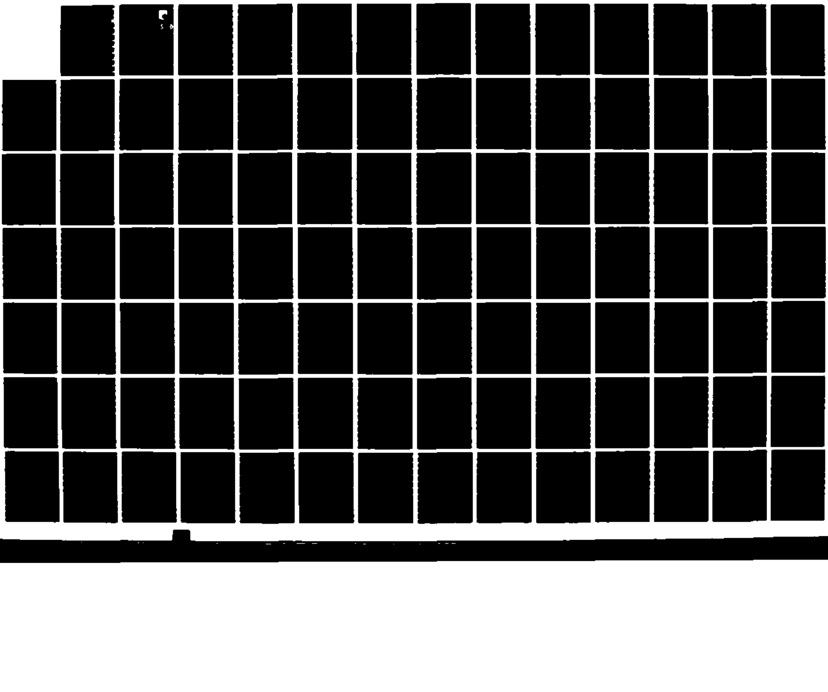
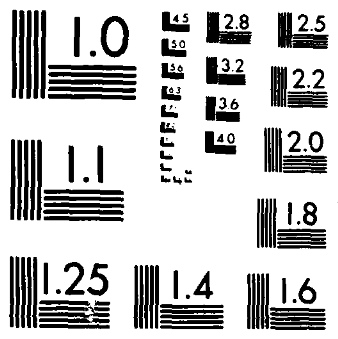
AD-A172 343   EVALUATION AND VALIDATION (E&V) TEAM PUBLIC REPORT   1/5
VOLUME 2(U) AIR FORCE WRIGHT AERONAUTICAL LABS
WRIGHT-PATTERSON AFB OH R SZYMANSKI 30 NOV 85

UNCLASSIFIED   AFWAL-TR-85-1016-VOL-2   F/G 14/2   NL




1.0
4.5
5.0
5.6
6.3
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

1

AFWAL-TR-85-1016
VOL II

EVALUATION AND VALIDATION (E&V)

TEAM PUBLIC REPORT
Volume II

RAYMOND SZYMANSKI
E&V TEAM CHAIRPERSON
AVIONICS LABORATORY
AIR FORCE WRIGHT AERONAUTICAL LABORATORIES
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433-6543

30 NOVEMBER 1985

Interim Technical Report for Period
1 October 1984 - 30 September 1985

APPROVED FOR PUBLIC RELEASE,
DISTRIBUTION UNLIMITED

PREPARED FOR:

ADA* JOINT PROGRAM OFFICE
3D139 (FERN ST/C107) PENTAGON
WASHINGTON, D.C. 20301

AVIONICS LABORATORY
AIR FORCE WRIGHT AERONAUTICAL LABORATORIES
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433-6543

[illegible] 087

for [illegible]
Raymond Szymanski
Project Engineer

Date 6 May 86

FOR THE COMMANDER

[illegible]
FRANKLIN C. GERKEN, Lt Col, USAF
Deputy Chief
System Avionics Division
Avionics Laboratory

Date 6 May 86

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

AD-A172 343

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for Public Release; Distribution Unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFWAL-TR-85-1016, Vol II	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Air Force Wright Aeronautical Laboratories 6b. OFFICE SYMBOL (If applicable) AFWAL/AAAF-2	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State and ZIP Code) Wright-Patterson Air Force Base, OH 45433-6543	7b. ADDRESS (City, State and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Ada*JOINT PROGRAM OFFICE 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State and ZIP Code) 3D139 (Fern Street/C107) Pentagon Washington, D.C. 20301	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO.: 63226 PROJECT NO.: AJPO TASK NO.: 28 WORK UNIT NO.: 53
11. TITLE (Include Security Classification) Evaluation and Validation (E&V) Team Public Report, Volume II (U)	
12. PERSONAL AUTHOR(S) Raymond Szymanski, E&V Team Chairman	
13a. TYPE OF REPORT Interim Technical 13b. TIME COVERED FROM 1 Oct 84 TO 30 Sep 85	14. DATE OF REPORT (Yr., Mo., Day) 1985 November 30 15. PAGE COUNT 419
16. SUPPLEMENTARY NOTATION *Ada is a Registered Trademark of the U.S. Government (Ada Joint Program Office)	

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)	
FIELD	GROUP	SUB. GR.		
09	02		Ada*	Common APSE Interface Set (CAIS)
			Evaluation	Ada Programming Support Environment (APSE)
			Validation	

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Activities and accomplishments of the Evaluation and Validation (E&V) Team are reported for FY1985. The purpose of the E&V Task, which is sponsored by the Ada Joint Program Office (AJPO), is to develop techniques and tools that will provide a capability to perform assessment of Ada Programming Support Environments (AJSEs) and to determine conformance of APSEs to the Common APSE Interface Set (CAIS). As this technology is developed, it is being made available to DoD components, industry, and academia.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED [✓] SAME AS RPT. ☐ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL Raymond Szymanski	22b. TELEPHONE NUMBER (Include Area Code) (513) 255-2446	22c. OFFICE SYMBOL AFWAL/AAAF-2

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

SECTION I

PROJECT TECHNICAL SUMMARY

1.1 Introduction

This report is the second in a series of annual technical reports to be published by the Evaluation and Validation (E&V) Team. The purpose of the E&V Public Report is to provide an overview of the many technical accomplishments of the E&V Team during the preceding fiscal year. This second report contains information resulting from E&V activities during fiscal year 1985 (October 1984 - September 1985) which is being made available for public review and comment. Contents of this report reflect an observation of the E&V Team progress during the fiscal year and should not be viewed as final representations of the technology being developed.

1.2 Background

In June 1983 the Ada Joint Program Office (AJPO) proposed the formation of the E&V Task and a tri-service E&V Team, with the Air Force designated as lead service. The purpose of the E&V Task is to develop the techniques and tools which will provide a capability to perform assessment of Ada Programming Support Environments (APSEs) and to determine conformance of APSEs to the Common APSE Interface Set (CAIS). As the E&V technology is developed, it will be made available to the community for use by DoD components, industry, and academia as deemed appropriate by the respective organizations. In October 1983, the Air Force officially accepted responsibility as lead service and designated the Air Force Wright Aeronautical Laboratories (AFWAL) at Wright-Patterson Air Force Base as lead organization. By November 1983, a comprehensive E&V Plan was developed, and by December 1983 an E&V Team had been established, with the first meeting held at Wright-Patterson Air Force Base. In April 1984, an E&V Workshop was held at Airlie, Virginia. The purpose of the Workshop was to solicit the participation of industry representatives in the E&V Task. Many of the participants in the E&V Workshop have chosen to remain involved in the E&V Task as Distinguished Reviewers, and have contributed significantly to the accomplishments of the E&V Team.

1.3 E&V Meetings

E&V Team meetings are held on a quarterly basis. The 5-7 December 1984, 5-7 June 1985, and 4-6 September 1985 meetings were held at the Wright-Patterson Air Force Base. The 6-8 March 1985 meeting was held in San Diego, California. The E&V Workshop was not held in 1985 due to the low response to the call for papers. As a substitute, the E&V Team hosted a Birds of a Feather session at the February 1986 SIGAda meeting in Los Angeles, California. Communication among E&V Team members throughout the year is accomplished primarily via the MILNET.

1.4 E&V Team Organization

The E&V Team is organized into the following four working groups:

a. Requirements Working Group (REQWG)

The REQWG is responsible for reviewing life-cycle methodology materials to determine life-cycle issues which should be addressed by the E&V Team; developing an E&V Requirements Document; providing analysis of E&V requirements to determine completeness, traceability, testability, consistency, and feasibility; identifying issues which may impact the development of E&V technology but which do not necessarily correlate to APSE components; and providing recommendations for development/acquisition of E&V tools/aids.

b. Coordination Working Group (COORDWG)

The COORDWG is responsible for performing a literature search for efforts relevant to the E&V Task; developing a Technical Coordination Strategy Document which documents the relationship of these efforts to the E&V Task; and profiding technical presentations to the E&V Team on these related efforts. They are also responsible for identifying professional organizations which are technically related to the E&V Task; developing a Public Coordination Strategy Document; preparing E&V Status Reports; and developing and maintaining an E&V project reference list.

c. APSE Working Group (APSEWG)

The APSEWG is responsible for providing expertise on DoD and commercial APSEs available within the DoD; providing presentations to the E&V Team on these APSEs; identifying existing capabilities/tests/tools associated with each APSE; and developing a DoD APSE Analysis Document.

d. Standards Evaluation and Validation Working Group (SEVWG)

The SEVWG is responsible for providing a forum for the development of methodologies for evaluating and validating current, proposed or future standards relating to APSEs; identifying issues affecting the evaluation or validation of standards; suggesting validation approaches or evaluation criteria; providing inputs to any supported efforts developing evaluators or validators for standards; and developing an APSE Validation Procedures Document and a CAIS Analysis Document.

1.5 Conclusion

This E&V Public Report is being made available by the E&V Team in order to solicit comments from those individuals who are not actively involved in the E&V Task. All comments should be addressed to:

Raymond Szymanski
AFWAL/AAAF
Wright-Patterson AFB, Ohio 45433-6543
(RSZYMANSKI@ADA20)

APPENDIX A

EVALUATION and VALIDATION

TECHNICAL COORDINATION STRATEGY DOCUMENT

VERSION 2.0

29 AUGUST 1985

Executive Summary

The purpose of the Evaluation and Validation (E&V) Task, which is sponsored by the Ada Joint Program Office (AJPO), is to develop the technology by which Ada Programming Support Environments (APSEs) will be evaluated and validated. As the E&V technology is developed, it will be made available to the user community for implementation by DoD components, industry, an academia as appropriate. The purpose of this document is to provide lines of communication between the E&V Task and other technically related DoD and industry efforts and organizations. With respect to identifying and interfacing with other efforts/organizations, the following areas have been addressed : 1) the name of the technically related effort/organization; 2) purpose; 3) relationship to the E&V Task; 4) benefits to the E&V Task; 5) benefits to the related effort or organization; 6) impact on E&V Task schedules; 7) impact on related effort or organization task schedule; 8) required level of coordination; 9) resolution of issues; and 10) focal point.

Table of Contents

1. INTRODUCTION

1.1 Objective of the Technical Coordination Strategy Document

The objective of the Technical Coordination Strategy Document (TCSD) is to provide a mechanism whereby both Department of Defense (DoD) and contractor technical efforts/organizations which are potentially related to the Evaluation and Validation (E&V) of Ada Programming Support Environments (APSEs) Task, may be identified. Specifically, the TCSD will identify : a) related technical efforts; b) relationships between the E&V Task and each related effort; c) areas of mutual benefit; d) impact of schedules; e) the level of coordination required between the E&V Task and each related effort; and f) issues which require resolution with respect to the mutual benefit of both the E&V Task and the particular related effort involved.

1.2 Background

The purpose of the E&V Task, which is sponsored by the Ada Joint Program Office (AJPO), is to develop the technology by which APSEs will be evaluated and validated. The term "evaluation" represents a qualitative assessment of an APSE component for which no objective standard exists. The term "validation" represents a quantitative measurement of an APSE component for which both a standard and metrics exist. Techniques and tools will be developed which will provide a capability to perform assessment of APSEs and to determine conformance of APSEs to the Common APSE Interface Set (CAIS), which is being developed by the Kernel Ada Programming Support Environment (KAPSE) Interface Team (KIT) and their companion organization, the KAPSE Interface Team from Industry and Academia (KITIA).

As the E&V technology is developed, it will be made available to the user community for implementation by the DoD components, industry, and academia as appropriate.

2. SCOPE

The overall goal of the TCSD is to establish lines of communication between the E&V Task and other related DoD and industry efforts/organizations. It is essential to the success and effectiveness of the E&V Task as a whole to coordinate with other related efforts. This type of coordination and communication will keep other organizations and efforts abreast of the E&V Task and its resulting technology and will identify those areas of evaluation and validation which are of mutual benefit.

During the December 1984 E&V Team meeting, the decision was made to combine two of the five working groups, specifically, the Public Coordination Working Group (PUBWG) and the Technical Coordination Working Group (TECWG). This combination resulted in the establishment of the Coordination Working Group (COORDWG), whose scope encompasses both communicating/interfacing with the public in terms of disseminating E&V information, as well as establishing/maintaining a technical interface between the E&V Task and other technically related efforts. It is the responsibility of the COORDWG to : 1) develop the TCSD; 2) provide technical presentations to the E&V Team on related technical efforts identified; and 3) provide position papers throughout the duration of the E&V Task which address particular aspects of the E&V Task with related tasks/efforts. The COORDWG is responsible for both providing and updating the status of these technically related efforts to the Team, as well as enhancing this document in future revisions with the identification of additional tasks/efforts and updated information on currently identified efforts. In addition to these tasks, the COORDWG is responsible for the information contained in the Public Coordination Strategy Document.

This version of the TCSD was developed by combining the various Technical Coordination Statements which were prepared by members of the E&V Team, who are presently involved or associated with the identified task/effort. The following represents the Technical Coordination Statement template used by each E&V Team member :

1. Name of the technically related effort
2. Purpose
3. Relationship to the E&V Task
4. Benefits to the E&V Task
5. Benefits to the related effort/organization
6. Impact on E&V Task schedules
7. Impact on related effort/organization schedules
8. Required level of coordination
9. Resolution of issues
10. Focal point

3. APPROACH

Currently, two primary methods to establish and promote coordination between the E&V Task and other related technical efforts/organizations have been identified and are outlined below.

3.1 Invited Briefings

Invitations will be extended to particular individuals to attend the quarterly E&V Team meetings as appropriate, for the purpose of briefing specific related efforts. These briefings will provide interactive communication and dialogue between the E&V Team members and the particular briefer with respect to exchange of technical information.

3.2 Technical Coordination Statements/COORDWG Briefings

Technical Coordination Statements will be used in conjunction with the method indicated in paragraph 3.1. The purpose of these statements is to identify a related effort (or organization), and elaborate upon various aspects of this relationship. Currently, ten specific relational aspects are identified on the Technical Coordination Statement, as indicated above.

In addition, the COORDWG Chairperson (or Vice-Chairperson) will update the E&V Team on the status of various related technical efforts at the quarterly E&V meetings. These briefings will adhere to the following format :

NAME OF RELATED EFFORT/ORGANIZATION

PROGRAM MANAGER (ADDRESS/PHONE)

PROGRESS STATEMENT (SIGNIFICANT EVENTS/MILESTONES) SINCE LAST UPDATE

DATE

4. IDENTIFICATION/ELABORATION OF RELATED TECHNICAL EFFORTS

The following technical efforts/organizations have been identified as being related to the E&V Task and are elaborated in the following paragraphs.

4.1 Ada C3I Test and Evaluation

4.1.1 Purpose

The purpose of this effort is to test and evaluate the effectiveness of using an integrated Ada programming environment in an Air Force command, control, communication and intelligence (C3I) system software development project. Areas to be evaluated include the use of Ada as a programming design language, documentation, the quality and quantity of the code produced, and compiler performance and productivity. In order to determine the effectiveness of Ada, the software for a selected Air Force system acquisition will be implemented in Ada as a parallel effort.

4.1.2 Relationship to the E&V Task

The results of this effort will be a technical report describing the results of the test and evaluation. The results of this effort may aid in the development of requirements and criteria for the evaluation and validation of APSEs.

4.1.3 Benefits to the E&V Task

The information gained as a result of this effort may assist the E&V Task in the development of APSE evaluation and validation requirements and criteria.

4.1.4 Benefits to the Related Effort/Organization

Any requirements and criteria for the evaluation of compilers developed before this contract is awarded will aid in the evaluation of the Ada Integrated Environment (AIE) Ada compiler.

4.1.5 Impact on E&V Task Schedules

No schedule impacts are currently identified.

4.1.6 Impact on Related Effort/Organization Schedules

No schedule impacts are currently identified.

4.1.7 Required Level of Coordination

At present, Elizabeth Kean is an active member of the E&V Team and will assist in the coordination of this effort and the E&V Task.

4.1.8 Resolution of Issues

Issues identified within the E&V Task will be handled within the E&V Task. Issues identified within the Ada C3I Test and Evaluation effort will be resolved through the Rome Air Development Center (RADC) chain of command, up to and including the AJPO.

4.1.9 Focal Point

The focal point is indicated below :

Elizabeth Kean

Rome Air Development Center

Commercial : (315) 330-4325

Autovon : 587-4325

4.2 Ada Integrated Environment

4.2.1 Purpose

The purpose of this Air Force-directed effort is to design and develop a Minimal Ada Programming Support Environment (MAPSE) including a state-of-the-art Ada compiler. The Ada compiler will be developed for rehosting and retargeting to a number of computers. The MAPSE will also consist of software tools and aids to assist programmers and project managers in the development of Ada software. Procedures for rehosting/retargeting the compiler and the MAPSE will be developed under this effort.

4.2.2 Relationship to the E&V Task

The product of this effort, a MAPSE, may eventually be evaluated and validated using the requirements and criteria developed under the E&V Task.

4.2.3 Benefits to the E&V Task

The AIE is the Air Force's implementation of a MAPSE. The AIE can be used as an aid in determining the requirements and criteria for evaluating and validating future APSEs.

4.2.4 Benefits to the Related Effort/Organization

The E&V technology developed under the E&V Task will aid in the assessment of future software tools to be incorporated in the AIE. The CAIS will eventually be implemented on the AIE. The CAIS and the CAIS validation capability will provide standardization of interfaces and a method for validating the implemented interfaces.

4.2.5 Impact on E&V Task Schedules

No schedule impacts are currently identified.

4.2.6 Impact on Related Effort/Organization Schedules

The CAIS will eventually be implemented on the AIE, therefore, the CAIS validation schedule may impact the AIE effort.

4.2.7 Required Level of Coordination

At present, Elizabeth Kean is an active member of the E&V Team and a technical evaluator on the AIE effort and will provide coordination between this effort and the E&V Task.

4.2.8 Resolution of Issues

Issues identified within the E&V Task will be handled within the E&V Task. Issues identified within the AIE effort will be resolved through the Rome Air Development Center (RADC) chain of command, up to and including the AJPO.

4.2.9 Focal Point

The focal point is indicated below :

Donald Mark

Rome Air Development Center

Commercial : (315) 330-3398

Autovon : 587-3398

4.3 Ada Joint Program Office

4.3.1 Purpose

The purpose of the AJPO, which was established on 12 December 1980 by the Under Secretary of Defense for Research and Engineering, is to manage the DoD's effort to implement, introduce and provide life-cycle support for Ada. The AJPO must ensure the implementation and maintenance of Ada as a consistent, unambiguous standard recognized by the DoD and also by the widest possible community. The AJPO must ensure the smooth introduction and acceptance of Ada in the DoD as early as possible, consistent with the needs of individual components. The AJPO must ensure the provision of life-cycle support for Ada through the development of a robust Ada Programming Support Environment (APSE) to improve productivity both in development and in continued evolution.

4.3.2 Relationship to the E&V Task

The AJPO is the sponsor of the E&V Task. The status of the E&V Task is briefed to the AJPO at the quarterly Tri-Service review meetings.

4.3.3 Benefits to the E&V Task

The AJPO oversees all of the E&V Task activities and provides managerial direction and funding to the E&V Task as necessary.

4.3.4 Benefits to the Related Effort/Organization

The development of the E&V technology by the E&V Task supports the AJPO objective of improving the productivity in development and continued evolution of APSEs.

4.3.5 Impact on E&V Task Schedules

No schedule impacts are currently identified.

4.3.6 Impact on Related Effort/Organization Schedules

No schedule impacts are currently identified.

4.3.7 Required Level of Coordination

The AJPO focal point (LCDR Philip Myers) attends E&V Team meetings and is on the distribution list for all E&V Team MILNET communication. In addition, the E&V Team Chairperson (Raymond Szymanski) is required to brief the AJPO on the status of the E&V Task at quarterly Ada Tri-Service Reviews.

4.3.8 Resolution of Issues

All such issues should be brought to the attention of the AJPO by the E&V Team Chairperson. The AJPO has final authority in the resolution of such issues.

4.3.9 Focal Point

The focal point is indicated below :

LCDR Philip Myers

Ada Joint Program Office

1211 South Fern St

RM C107

Arlington, VA 22202

Commercial : (202) 694-0208

Autovon : 224-0208

MILNET : MYERS@ECLB

4.4 Ada Language System

4.4.1 Purpose

The Ada Language System (ALS) is under the direction of the U.S. Army. The purpose of this effort is to develop an APSE on the VAX/VMS 11/780 with a MIL-STD-1815A host compiler targeted to the VAX. Other targets include the Military Computer Family (MCF) Nebula instruction set architecture (ISA), and the Intel 8086.

4.4.2 Relationship to the E&V Task

The technology developed through the E&V Task can be applied to the ALS development.

4.4.3 Benefits to the E&V Task

The ALS represents the Army's implementation of an APSE. The ALS can be used as an example in determining the criteria for performing evaluation and validation on future APSEs.

4.4.4 Benefits to the Related Effort/Organization

The technology which is developed by the E&V Team will provide input to the development of the ALS. Also, the CAIS, which is currently being developed by the KIT/KITIA, will be used with the ALS at a future time.

4.4.5 Impact on E&V Task Schedules

No schedule impacts are currently identified.

4.4.6 Impact on Related Effort/Organization Schedules

No schedule impacts are currently identified.

4.4.7 Required Level of Coordination

Coordination will be with the Army's ALS maintenance personnel. In addition, Mr James Williamson is currently participating as the Air Force representative on the Tri-Service ALS testing team.

4.4.8 Resolution of Issues

All such issues should be brought to the attention of the AJPO by the E&V Team Chairperson. The AJPO shall be responsible for informing the appropriate Army personnel and seeking resolution of these issues.

4.4.9 Focal Point

The focal point is indicated below :

Dennis Turner

DRSEL-TCS-Ada

U.S. Army/CECOM, Ft. Monmouth, New Jersey 07703

Commercial : (210) 544-4149

Autovon : 995-4149

4.5 Ada Program Design Language Evaluation Guidelines

4.5.1 Purpose

As part of a general plan to use the best available technology for software development and maintenance, the Facilities Engineering and Systems Development (FESD) branch of Transport Canada requires contractors to use an Ada program design language (PDL). The Ada PDL evaluation guidelines will be used by project managers to assess the compliance of software development proposals with the requirement to use an Ada PDL. The Ada PDL evaluation guidelines are also intended to provide industry with an outline of the requirements for an Ada PDL capability relevant to the needs of Transport Canada.

4.5.2 Relationship to the E&V Task

The evaluation guidelines produced will be publically available in early 1985. The E&V Team may acquire this document at that time.

4.5.3 Benefits to the E&V Task

The Transport Canada Ada PDL evaluation guidelines will contribute to the guidelines being prepared by the E&V Team for evaluating PDL capabilities.

4.5.4 Benefits to the Related Effort/Organization

The Transport Canada Ada PDL guidelines will be enhanced by any constructive criticism made by the E&V Team.

4.5.5 Impact on E&V Task Schedules

No schedule impacts are currently identified.

4.5.6 Impact on Related Effort/Organization Schedules

No impact is anticipated. The Transport Canada Ada PDL evaluation guidelines will be produced as a first version early in 1985. The production of an updated second version will depend on the feedback resulting from the first version. A second version is not currently anticipated in the near term.

4.5.7 Required Level of Coordination

At present, Randal Leavitt is an active member of the E&V Team representing Canadian interests and activities. He will provide any needed coordination between the E&V Team and Transport Canada.

4.5.8 Resolution of Issues

Both Transport Canada and the E&V Team will resolve issues related to Ada PDL evaluation guidelines independently. Transport Canada is interested, however, in any constructive criticism that the E&V Team members may care to offer.

4.5.9 Focal Point

The focal point is indicated below :

Randal Leatitt

PRIOR Data Sciences, Ltd.

39 Highway 7, Bell Mews Plaza

Nepean, Ontario

K2H 8R2

Commercial : (613) 820-7235

MILNET : MAGLIERI@ECLB

4.6 Ada Test and Verification System

4.6.1 Purpose

The purpose of this effort is to design an Ada Test and Verification System (ATVS) which can be implemented as a set of computer-based software tools to improve the reliability and maintainability of Ada software systems. It is intended that this system will be applied during the coding, testing, verification, and error detection/correction phases of software development. This effort will begin with a study to determine the most advanced techniques and capabilities to be included in the design. The ATVS will then be designed as an integral component of an APSE. As a minimum, the ATVS shall be designed for use with both the Air Force's AIE and the Army's ALS.

4.6.2 Relationship to the E&V Task

The ATVS will be a portable software tool residing on an APSE. Thus, the technology developed by the E&V Task can be applied.

4.6.3 Benefits to the E&V Task

If the CAIS is available at the time of implementation, it will be used as the interface.

4.6.4 Benefits to the Related Effort/Organization

The requirements and criteria for the evaluation and validation of APSEs ma result from the initial analysis study and the resulting implementation.

4.6.5 Impact on E&V Task Schedules

No schedule impacts are currently identified.

4.6.6 Impact on Related Effort/Organization Schedules

The development of the CAIS may impact the schedules of this effort.

4.6.7 Required Level of Coordination

At present, Elizabeth Kean will relay any information to and from the RADC focal point.

4.6.8 Resolution of Issues

Issues identified within the E&V Task will be handled within the E&V Task. Issues identified within the ATVS effort will be resolved through the RADC chain of command.

4.6.9 Focal Point

The focal point is indicated below :

Richard Evans

Rome Air Development Center

Commercial : (315) 330-3398

Autovon : 587-3398

4.7 Ada Validation Organization

4.7.1 Purpose

The Ada Validation Organization (AVO) is sponsored by the AJPO. Its purpose is to ensure that developers of Ada compilers have correctly implemented the standard Ada language (ANSI/MIL-STD-1815A-1983).

4.7.2 Relationship to the E&V Task

The AVO has been responsible for the development and implementation of an Ada Compiler Validation Capability (ACVC) in order to determine conformance of Ada compilers to the standard Ada language. The ACVC provides a capability to validate one particular tool within an APSE and, as such, will be incorporated within the E&V technology developed by the E&V Team.

4.7.3 Benefits to the E&V Task

The AVO has established formal procedures for validating Ada compilers and mechanisms by which the validation procedures are executed. The expertise gained through the development and implementation of these procedures will be beneficial to the E&V Team as it begins to establish recommendations for formal procedures for the implementation of E&V technology.

4.7.4 Benefits to the Related Effort/Organization

The E&V Task is responsible for developing evaluation capabilities, as well as validation capabilities. The determination of evaluation criteria for the assessment of compilers is but one of the many activities being performed by the E&V Task. The Ada compiler evaluation capability will be of particular benefit to the AVO.

4.7.5 Impact on E&V Task Schedules

No schedule impacts are currently identified.

4.7.6 Impact on Related Effort/Organization Schedules

No schedule impacts are currently identified.

4.7.7 Required Level of Coordination

At present, Capt Albert Deese of the Language Control Facility at WPAFB (ASD/ADOL) is an active member of the E&V Team, and coordination will be through him.

4.7.8 Resolution of Issues

Issues of concern should be raised to the AJPO level if necessary for resolution.

4.7.9 Focal Point

The focal point is indicated below :

Thomas Probert

Institute for Defense Analyses

1801 N. Beauregard St.

Alexandria, Virginia 22311

Commercial : (703) 845-2517

Autovon : 289-1948 (ext. 2517)

4.8 Air Force Computer Resource Management Technology

4.8.1 Purpose

The over-all objective of the Air Force Computer Resource Management Technology Program Element (64740F) effort is to apply advances in computer resource management technology to the development and acquisition of Air Force and other military systems.

4.8.2 Relationship to the E&V Task

This program element (PE) supports the development and application of techniques to increase the performance and reduce the costs of mission-critical computer resources. It includes proposed programs from several Air Force Systems Command Product Divisions to develop criteria for evaluating Ada compilers. In addition, this PE includes programs which plan for and support the introduction of the Ada programming language.

4.8.3 Benefits to the E&V Task

Various 64740F-sponsored programs may result in evaluation criteria for Ada compilers and other Ada tools that may be useful to the E&V Task. Also, new software tools developed under 64740F may expand the APSE functionality definition.

4.8.4 Benefits to the Related Effort/Organization

One project within the PE is concerned with software acquisition standards and mechanisms to improve the acquisition and support of computer resources. The E&V criteria developed by the E&V Task will directly contribute to the goal of this project.

4.8.5 Impact on E&V Task Schedules

No schedule impacts are currently identified.

4.8.6 Impact on Related Effort/Organization Schedules

No schedule impacts are currently identified.

4.8.7 Required Level of Coordination

At present, Debra Harto is an active member of the E&V Team and coordination will be accomplished through her.

4.8.8 Resolution of Issues

The focal point for coordination will assist in resolving any issues that arise which may adversely affect either effort.

4.8.9 Focal Point

The focal point is indicated below :

William Letendre

ESD/ALEE

Hanscom Air Force Base, Massachusetts 01731

Autovon : 478-5113

4.9 Common Ada Missile Packages

4.9.1 Purpose

The objective of the Common Ada Missile Packages (CAMP) program is to explore the feasibility of developing reusable software in Ada for armament applications, and an associated parts composition system. CAMP is sponsored by the Air Force Armament Laboratory (AFATL), the Air Force Munition and Ordnance Program Element (64602F), the Software Technology for Adaptable Reliable Systems (STARS) program, and the AJPO.

4.9.2 Relationship to the E&V Task

The product of this effort, APSE library components and associated parts composition system, may eventually be evaluated using the requirements and criteria developed under the E&V Task.

4.9.3 Benefits to the E&V Task

The reusable software components and associated parts composition system developed under the CAMP program, will result in new technology which may expand the requirements and criteria for evaluating future APSEs. In addition, the common armament functions identified by the CAMP program may serve as a basis for developing armament-specific compiler benchmarks for Ada compiler evaluation.

4.9.4 Benefits to the Related Effort/Organization

The CAIS and CAIS Validation Capability (CVC) will provide standard interfaces for future APSE libraries and supporting parts composition systems, such as are being developed by the CAMP effort.

4.9.5 Impact on E&V Task Schedules

No schedule impacts are currently identified.

4.9.6 Impact on Related Effort/Organization Schedules

The following E&V Task schedules may impact the schedules of the CAMP effort.

CAIS Validation Capability contract start - 2nd Quarter FY85

Draft CVC - 1st Quarter FY86

Version 1 CVC - 1st Quarter FY86

4.9.7 Required Level of Coordination

At present, Debra Harto is an active member of the E&V Team and coordination will be accomplished through her.

4.9.8 Resolution of Issues

Issues identified related to the E&V Task will be addressed within the E&V Task. Issues identified related to the CAMP program will be handled by the AFATL CAMP Program Manager.

4.9.9 Focal Point

The focal point is indicated below :

Chris Anderson

AFATL/DLCM

Eglin Air Force Base, Florida 32542

Commercial : (904) 882-2961

Autovon : 872-2961

4.10 Johnson Space Center Ada Project

4.10.1 Purpose

The Johnson Space Center (JSC) Testing and Analysis of DoD Ada Language Products for NASA (JSC Ada Project) effort is sponsored by National Aeronautics and Space Administration (NASA) Headquarters. The purpose of this effort is to perform testing and analyses of Ada software technology products being produced by the DoD, evaluate their applicability to future NASA projects (such as the Space Station) and develop a plan for their implementation in future NASA flight systems as a standard. The JSC Ada Project was established as a result of the Memorandum of Agreement (MOA) signed in June 1983 by Dr. Edith Martin, Deputy Under Secretary of Defense for Research and Development Technology, and Dr. Jack Kerrebrock, NASA Associate Administrator for Aeronautics and Space Technology. This agreement establishes NASA/DoD cooperation in the DoD STARS Program, and recognizes APSE Beta Testing at the JSC and the University of Houston (at Clear Lake City) as part of that cooperation.

4.10.2 Relationship to the E&V Task

Both tasks have a common goal of developing technology for use in evaluating APSEs. But in addition to using technology provided by the E&V Task, the JSC Ada Project will also develop specific evaluation criteria and tests based upon technology and tools used in current NASA spaceflight systems (e.g., the HAL/S programming support system currently used as a NASA standard).

4.10.3 Benefits to the E&V Task

The JSC Ada Project will primarily focus on use of APSEs in the development of prototype applications. Data and information from this activity will be used to develop standards and criteria later used in evaluating APSEs for NASA. This work, in conjunction with other studies and analyses at JSC, will identify additional APSE features and tools needed for support of NASA spaceflight applications projects. Information provided by the JSC Ada project should assist the E&V Task in its development of standards and criteria for use in evaluation of APSEs.

4.10.4 Benefits to the Related Effort/Organization

The technology developed by the E&V Task will be utilized in the JSC Ada Project to assist in evaluating APSEs for NASA.

4.10.5 Impact on E&V Task Schedules

No schedule impacts are currently identified.

4.10.6 Impact on Related Effort/Organization Schedules

An impact to the APSE evaluation criteria may occur.

4.10.7 Required Level of Coordination

At present, Mr. Terry D. Humphrey is an active member of the E&V Team, and is also the Steering Group Subcommittee Chairman for prototype applications development within the JSC Ada Project.

4.10.8 Resolution of Issues

Such issues should initially be addressed within the respective task in which they arise (i.e., the E&V Task or JSC Ada Project). Recommendations should then be developed within that task to resolve such issues. The issues and associated recommendations should then be presented to the other task leader. Task leaders should work together to obtain resolution. If resolution is unattainable at that level, both task leaders should elevate the issues for review by STARS personnel.

4.10.9 Focal Point

The focal point is indicated below :

Jack Garman

Johnson Space Center

Mail Code : FD

Houston, Texas 77058

(713) 483-4788

4.11 KAPSE Interface Team/KAPSE Interface Team from Industry and Academia

4.11.1 Purpose

The Kernel Ada Programming Support Environment (KAPSE) Interface Team (KIT) is a Navy-led organization sponsored by the Ada Joint Program Office (AJPO). Both the KIT and its companion Industry-Academia Team (KITIA), were formed by a MOA signed by the Assistant Secretary of each of the Services, and the Under Secretary of Defense in early 1982. The KITIA consists of volunteer representatives from industry and universities who provide technical expertise to the KIT. Their purpose is to contribute to the achievement of interoperability of applications databases and transportability of software development tools ("I&T"). In order to accomplish this objective, the KIT/KITIA is defining a Common APSE Interface Set (CAIS) to which all Ada-related tools can be written, thus assuring the ability to share tools and databases between conforming APSEs.

4.11.2 Relationship to the E&V Task

The CAIS is now a draft MIL-STD and is expected to become a full MIL-STD in the near future. As such, a CAIS Validation Capability (CVC) must be developed to enable determination of conformance to the MIL-STD by APSEs which implement the CAIS. One of the goals of the E&V Task is to develop the CVC.

4.11.3 Benefits to the E&V Task

In addition to the definition of the CAIS, the KIT/KITIA activities of developing requirements and criteria, improving upon the STONEMAN definition of an APSE, providing APSE-related terminology and definitions, examining the issue of determining compliance of the CAIS to the original requirements, etc., will provide useful inputs and obviate the need for repetitive activities by the E&V Task. In addition, E&V Tools developed in the future should be portable across CAIS implementations.

4.11.4 Benefits to the Related Effort/Organization

The process of developing a CVC in parallel with the definition of the CAIS will provide to the KIT/KITIA information related to problems encountered by the E&V Task in understanding CAIS semantics, such as ambiguities and inconsistencies, thus enabling the KIT/KITIA to modify the CAIS definition accordingly. The E&V Task will also help guide the KIT/KITIA in their choice of how to express the semantics and the specifications themselves, based upon experience with what can be validated best.

4.11.5 Impact on E&V Task Schedules

The following KIT/KITIA schedules may potentially impact the E&V Task schedules.

CAIS Draft Version 2 - January 1986

CAIS MIL-STD Version 2 - January 1987

4.11.6 Impact on Related Effort/Organization Schedules

The following E&V Task schedules may impact the schedules of the KIT/KITIA.

CVC contract start - 1st Quarter of FY86

CV^ Version 1 - 1st Quarter of FY87

CVC Version 2 - 4th Quarter of FY87

CVC Version 3 - 3rd Quarter of FY88

CVC Version 4 - 2nd Quarter of FY89

4.11.7 Required Level of Coordination

There are several E&V Team members who are also members of the KIT/KITIA. Areas of common interest are coordinated through these common representatives. The representatives are Virginia Castor, Elizabeth Kean, Timothy Lindquist, Patricia Oberndorf, Paul Reilly, and Guy Taylor.

4.11.8 Resolution of Issues

Issues of concern should be coordinated through the common E&V-KIT/KITIA representatives and raised to the level of team leaders if necessary for resolution.

4.11.9 Focal Point

The focal point for the KIT/KITIA is indicated below :

Patricia Oberndorf

Naval Ocean Systems Center (NOSC)

Code 8322

San Diego, California 92152

Commercial (619) 225-6682

Autovon 933-6682

4.12 Prototype Advanced Ada Programming Support Environment

4.12.1 Purpose

The purpose of the Prototype Advanced Ada Programming Support Environment (PA-APSE) project is to combine research into advanced APSE features, with support for the KIT.

4.12.2 Relationship to the E&V Task

The E&V Task is concerned with developing criteria for judging the quality and value of APSEs. The PA-APSE project investigates potential APSE features which may be found to be required or desirable, and so included in the features considered by E&V.

4.12.3 Benefits to the E&V Task

The PA-APSE project will identify APSE features of potential interest to the E&V Task, and the qualities of those features which are desirable.

4.12.4 Benefits to the Related Effort/Organization

The E&V Task may identify potential APSE features which should be further investigated by the PA-APSE project.

4.12.5 Impact on E&V Task Schedules

No schedule impacts are currently identified.

4.12.6 Impact on Related Effort/Organization Schedules

No schedule impacts are currently identified.

4.12.7 Required Level of Coordination

Mutual information flow is recommended. This can be provided via the communication paths already established between the KIT and the E&V Team.

4.12.8 Resolution of Issues

Such issues should be brought to the attention of the E&V Chairperson, and the PA-APSE contract monitor.

4.12.9 Focal Point

The focal points are indicated below :

Frank Belz (Project Manager at TRW)

TRW DSG

One Space Park

R2/1127

Redondo Beach, California 92078

Commercial : (213) 535-1623

Patricia Oberndorf (Contract Monitor)

Code 8322

NOSC

San Diego, California 92152

Commercial : (619) 225-6682/7401

4.13 Software Engineering Automation for Tactical Embedded Computer Systems

4.13.1 Purpose

The Software Engineering Automation for Tactical Embedded Computer Systems (SEATECS) project is an internal Naval Ocean Systems Center (NOSC) effort which conducts research into environment construction issues. A set of Top Level Requirements has been developed, and a proposed environment architecture will soon be published. The project also includes an experimental environment which is used to conduct investigations into various proposed environment features. Although SEATECS is not exclusively concerned with APSEs, all of the SEATECS work is applicable to APSEs.

4.13.2 Relationship to the E&V Task

SEATECS is involved in establishing and investigating potential environment features. Such features could be of interest to the E&V Task. In addition, SEATECS seeks to resolve various issues in environment construction which could be of importance to the E&V Task.

4.13.3 Benefits to the E&V Task

SEATECS will identify various aspects of environments which are important to a potential user, but which are often over-looked in current approaches to environment construction.

4.13.4 Benefits to the Related Effort/Organization

The E&V Task may identify issues which are appropriate for investigation using the SEATECS approach.

4.13.5 Impact on E&V Task Schedules

No schedule impacts are currently identified.

4.13.6 Impact on Related Effort/Organization Schedules

No schedule impacts are currently identified.

4.13.7 Required Level of Coordination

Mutual information flow is recommended. This can be provided via the communication paths already established between NOSC and the E&V Team.

4.13.8 Resolution of Issues

Such issues should be brought to the attention of the E&V Chairperson, and the SEATECS project manager, Howard Harvey. The KIT Chairperson, Patricia Oberndorf, will act as liaison as required.

4.13.9 Focal Point

The focal point is indicated below :

Howard Harvey

Code 8322

NOSC

San Diego, California 92152

Commercial : (619) 225-6682/7401

4.14 STARS - Application Thrust Area

4.14.1 Purpose

The STARS Application Thrust Area is three-phased in nature and can be summarized as follows : Phase I - Learning/Collection. Because of the need to get some early visability, this phase aims at accelerating the use of improved software tools and methods on DoD systems developments without creating undue start-up costs or schedule risks. While longer term competitive procurement activity will be initiated in this phase, selective application-specific prototyping will be initiated immediately where results and concepts can be applied for high immediate payoff for technology transition; Phase II - Major Development and Refinement. Some Phase I tasks will continue but the major thrust is in the development contracting initiative begun in Phase I to concentrate in acquisition of component repositories and insertion of components into system building harnesses and prototyping scenarios for a few important application areas; and Phase III - Completion and Transition to Services. Some of the basic technology work will continue but the main focus will be in technology transition of results into services and industry.

4.14.2 Relationship to the E&V Task

Like the E&V Task, the STARS Application Thrust Area is concerned with ensuring that quality software tools are acquired and/or developed, and implemented in mission critical software development efforts. Both tasks can complement each other in this respect. Further, both tasks are concerned with transitioning their respective technologies to the user community in a timely manner.

4.14.3 Benefits to the E&V Task

The STARS Application Thrust Area can benefit the E&V Task in terms of providing insight into the application of reusable software components (e.g., reusable libraries), as well as providing a perspective on software component composition, access methods, etc., and matching findings against Ada capabilities with respect to establishing guidelines for future development efforts.

4.14.4 Benefits to the Related Effort/Organization

The E&V Task can benefit the STARS Application Thrust Area by providing input into the required functionality of various software tools and resources which would ensure the development of quality components.

4.14.5 Impact on E&V Task Schedules

No schedule impacts are currently identified.

4.14.6 Impact on Related Effort/Organization Schedules

No schedule impacts are currently identified.

4.14.7 Required Level of Coordination

At present, lines of communications in terms of an E&V/STARS liaison are being investigated.

4.14.8 Resolution of Issues

Once such issues are identified, they should initially be addressed within the specific task in which the issue arose (i.e., E&V or STARS Application). Recommendations should then be developed within that task to resolve such issues. The issues and associated recommendations should then be presented to the other task leader. Task leaders should work together to obtain resolution. If resolution is unattainable at that level, both task leaders should elevate the issues for review by AJPO and STARS personnel.

4.14.9 Focal Point

The focal point is indicated below :

Robert D. Kolacki

Naval Electronic System Command

Elex 8141B NC #1

Room 5E40

Washington D.C. 20363

Commercial : (202) 692-8484

MILNET : KOLACKI@NRL

4.15 STARS - Business Practices Thrust Area

4.15.1 Purpose

The purpose of this effort is to improve the acquisition management of mission critical computer resources by the development of acquisition practice guidance based on past experience and by the integration and development of automated tools to facilitate acquisition/project tracking, monitoring, estimation, and interface with the development process. These improvements will be developed to be consistent and compatible with the defense system acquisition structure. Subgoals to improve business practices include : 1) identify and encourage the use of current effective practices which have not been institutionalized; 2) define better contracting mechanisms and incentives for defense systems software; 3) develop an integrated set of tools to facilitate software acquisition; 4) incentivise reuse of software; 5) apply expert systems technology to automate and facilitate software acquisition; and 6) interface the software acquisition tools to the software development environment.

4.15.2 Relationship to the E&V Task

One of the goals of the E&V Task is to develop evaluation criteria for APSE components with respect to the defined functionality of each tool. Since the objective of the STARS Business Practices Thrust Area is to develop and improve acquisition practices for mission critical computer resources, the E&V Task can provide useful input in terms of determining functional adequacy of various mission critical tools and resources which will be developed or acquired.

4.15.3 Benefits to the E&V Task

This effort can benefit the E&V Task by providing input in terms of project tracking, monitoring, estimation, etc., with respect to potential software tools and aids development efforts which may be required in order to perform evaluation and validation of APSE components.

4.15.4 Benefits to the Related Effort/Organization

The E&V Task can benefit the STARS Business Practices Thrust Area by providing input with respect to determining the appropriateness of various mission critical computer resources in terms of component functionality prior to a particular tool/resource development acquisition.

4.15.5 Impact on E&V Task Schedules

No schedule impacts are currently identified.

4.15.6 Impact on Related Effort/Organization Schedules

No schedule impacts are currently identified.

4.15.7 Required Level of Coordination

At present, lines of communications in terms of an E&V/STARS Business Practices liaison are being investigated.

4.15.8 Resolution of Issues

Once such issues are identified, they should initially be addressed within the specific task in which the issue arose (i.e., E&V or STARS Business Practices). Recommendations should then be developed within that task to resolve such issues. The issues and associated recommendations should then be presented to the other task leader. Task leaders should work together to obtain resolution. If resolution is unattainable at that level, both task leaders should elevate the issues for review by AJPO and STARS personnel.

4.15.9 Focal Point

The focal point is indicated below :

Philip S. Babel

ASD/EN (CRFP)

Wright-Patterson AFB, OH 45433-6543

Commercial : (513) 255-3656

Autovon : 785-3656

MILNET : BABELP@WPAFB-JALCF

4.16 STARS - Human Resources Thrust Area

4.16.1 Purpose

The objective of this effort is to provide the DoD by 1991 with established programs in career planning, training, and education which will provide and maintain the personnel resources needed to satisfy the STARS's goal. To meet this objective, the Human Resources Working Group (HRWG) recognizes the need for concurrent implementations of efforts in research, development, and insertion. Although the details of a Human Resources Program Plan have not been completed, the HRWG has definitized specific activities required to meet the thrust area objective. A major task of the HRWG is to determine what is presently being done in these areas and how this work can be adopted or redirected to support the STARS's goal.

4.16.2 Relationship to the E&V Task

One of the characteristics to be evaluated by the E&V Task is APSE usability. One of the major concerns of the Human Resources Thrust Area is the design of highly usable systems.

4.16.3 Benefits to the E&V Task

The Human Resources Thrust Area is focusing on steps that can be taken to improve the usability of future environments, while the E&V Task is focusing on developing the technology necessary to evaluate current and future environments. As a consequence of the STARS work, a greater understanding of the characteristics leading to the design of highly usable systems should be gained. This should, in turn, support an approach to evaluating usability which is based on an analysis of these characteristics (as compared to an approach which is purely empirical).

4.16.4 Benefits to the Related Effort/Organization

Progress within the Human Resources Thrust Area depends on an increas understanding of the characteristics leading to highly usable systems (includi APSEs). It is anticipated that the E&V Task will lead to the collection c empirical data that will be useful in identifying the e characteristics.

4.16.5 Impact on E&V Task Schedules

No schedule impacts are currently identified.

4.16.6 Impact on Related Effort/Organization Schedules

No schedule impacts are currently identified.

4.16.7 Required Level of Coordination

It is intended that a technical liaison be established between the E&V Tas and the STARS Human Resources Thrust Area in the near future.

4.16.8 Resolution of Issues

Issues should be addressed via coordination between the STARS Joint Progra Office and the E&V Team.

4.16.9 Focal Point

The focal point is indicated below :

Joseph Kernan

Commander

US Army Communications-Electronics Command

AMSEL-ICS-ED

Ft Monmouth, New Jersey 07703

4.17 STARS - Measurement Thrust Area

4.17.1 Purpose

This task, which is sponsored by the STARS program, is concerned with the development and use of measures to support evaluations and comparisons of software products, and of the processes associated with software development and support. The strategy for the Measurement Thrust Area includes establishing success criteria for the other task areas, performing cost/benefit analyses of various opportunities, collecting baseline data against which to measure progress, instrumenting automated support environments for data collection, and developing techniques for testing hypotheses and models related to software development and in-service support. Thus, this area is important not only for improving DoD programs, but also for assessing how well the STARS program is meeting its objectives.

4.17.2 Relationship to the E&V Task

The development of quantitative indicies to support comparision is key to both efforts. The Measurement Thrust Area of STARS is concerned with a broader area than the E&V Task.

4.17.3 Benefits to the E&V Task

One concern of the Measurement Thrust Area is the instrumentation of automated environments for data collection. Progress in this area will directly benefit the E&V Task.

4.17.4 Benefits to the Related Effort/Organization

The E&V Task is confronted with many of the same issues stemming from the effort to measure APSE characteristics of importance to potential users. Success in the E&V Task will require the development of objective, reliable procedures for measuring these characteristics, some of which (e.g., "usability") are difficult to pin down precisely. Many of the lessons learned, and measurement procedures resulting from the E&V Task will have direct relevance to the Measurement Thrust Area.

4.17.5 Impact on E&V Task Schedules

No schedule impacts are currently identified.

4.17.6 Impact on Related Effort/Organization Schedules

No schedule impacts are currently identified.

4.17.7 Required Level of Coordination

Plans are currently underway to establish a technical liaison between the E&V Task and the STARS Measurement Thrust Area.

4.17.8 Resolution of Issues

Issues should be addressed via coordination between the STARS Joint Program Office and the E&V Team.

4.17.9 Focal Point

The focal point is indicated below :

Joe Cavano

RADC/COEE

Griffis AFB, New York 13441

MILNET : CAVANO@RADC-MULTICS

4.18 STARS - Methodology Thrust Area

4.18.1 Purpose

The purpose of the STARS Methodology Thrust Area is to identify (specify) a set of methodologies from which a program manager can intelligently select a methodology for use in his/her project. To accomplish this objective, the specified set of methodologies can contain existing methodologies, new methodologies, or some combination of methodologies that, when integrated within a software engineering environment, satisfy the project manager's requirements. In other words, the Methodology Thrust Area will not define and develop a whole new set of methodologies for use on DoD programs; it will retain and apply as much as possible from those that already exist.

4.18.2 Relationship to the E&V Task

As technology is developed to evaluate life-cycle software methodologies, it should indicate which methodologies are better than others, and encourage tool set developers to implement those methodologies. At some point, the evaluation of the APSE should determine what methodologies are supported, evaluate them, and indicate to what extent the tool set supports them.

4.18.3 Benefits to the E&V Task

The E&V Task will benefit from the definition of technology to evaluate methodologies implemented on an APSE. Measures could possitly be developed which could be used on E&V tools and tool sets.

4.18.4 Benefits to the Related Effort/Organization

Even though a methodology is evaluated abstractly as good, bad, or somewhere in between, the tool set implementing that methodology can severely impact its usefulness. The E&V technology to evaluate tools and tool sets, should help in determining this aspect of the problem. Also, the E&V technology should assist in characterizing, evaluating, and selecting methodologies, and provide measures to accomplish this.

4.18.5 Impact on E&V Task Schedules

No schedule impacts are currently identified.

4.18.6 Impact on Related Effort/Organization Schedules

No schedule impacts are currently identified.

4.18.7 Required Level of Coordination

Plans are currently underway to establish a technical liaison between the E&V Task and the STARS Methodology Thrust Area.

4.18.8 Resolution of Issues

Issues should be identified, properly specified, and discussed within the team in which the issue originated. The issue should then be forwarded via the MILNET to the chairperson of the other team. The two chairpersons should determine how to address and resolve the issue.

4.18.9 Focal Point

The focal point is indicated below :

Peter Fonash

Army Material Command

AMCDE-SB

5001 Eisenhower Avenue

Alexandria, Virginia 22333-0001

Commercial : (202) 274-9318

MILNET : FONASH@ECLB

4.19 STARS - Software Engineering Enviroment Thrust Area

4.19.1 Purpose

The Software Engineering Environment (SEE) is sponsored by the Software Technology for Adaptable Reliable Systems (STARS) Program. A software engineering environment is an integrated system that supports mission-critical computer software over the entire life-cycle, from the initial statement of the requirements of the software to the support of the operational software. The three major objectives of the SEE Thrust Area are : 1) to define, design, and develop a production-quality SEE which can be used by all the Services; 2) to lay a solid foundation for the continuing evolution and technical advance of software engineering environments (beyond the STARS timeframe); and 3) to transition the 1990's environment to the services and to the Software Engineering Institute (SEI) for actual use by the services.

4.19.2 Relationship to the E&V Task

One of the goals of the SEE Thrust Area is to produce a high quality Joint Service SEE (JSSEE). As such, principles which will guide this thrust area include emphasis on production quality tools which reflect human engineering features, and which encourage good software engineering practices. The SEE and E&V Tasks will address common areas of interest and can benefit from one another in the research/technology common to both.

4.19.3 Benefits to the E&V Task

The SEE Task will result in the definition and preliminary design of a JSSEE, based upon careful review of life-cycle methodologies, tool functionalities, etc. The rationale which is used by the SEE Team to design the JSSEE, will provide useful requirements criteria to be addressed by the E&V Team.

4.19.4 Benefits to the Related Effort/Organization

The evaluation technology developed via the E&V Task will enable the JSSEE Team to assess the tools being incorporated within the JSSEE. The validation technology developed via the E&V Task will enable the JSSEE Team to determine JSSEE compliance with the CAIS , which is currently under development by the KIT/KITIA.

4.19.5 Impact on E&V Task Schedules

No schedule impacts are currently identified.

4.19.6 Impact on Related Effort/Organization Schedules

The following E&V Task schedules, with respect to the CAIS Validation Capability (CVC), may impact the JSSEE schedules.

Contract start - 1st Quarter FY86

CVC Version 1 - 1st Quarter FY87

CVC Version 2 - 4th Quarter FY87

CVC Version 3 - 4th Quarter FY88

CVC Version 4 - 4th Quarter FY89

4.19.7 Required Level of Coordination

At present, Ms. Ronnie Martin is an active member of both the E&V Team and the JSSEE Team. In addition, the E&V Team Chairperson, Raymond Szymanski, is on the distribution list for all JSSEE MILNET communications.

4.19.8 Resolution of Issues

Once such issues are identified, they should initially be addressed within the specific task in which the issue arose (i.e., E&V or SEE). Recommendations should then be developed within that task to resolve such issues. The issues, and associated recommendations, should then be presented to the other task leader. Task leaders should work together to obtain resolution. If resolution is unattainable at that level, both task leaders should elevate the issues for review by STARS and AJPO personnel.

4.19.9 Focal Point

The focal point is indicated below :

Phil J. Andrews

Naval Sea Systems Command

SEA 61R2

Washington DC 20362

Commercial : (202) 692-9761

Autovon : 222-9761

MILNET : PANDREWS@ECLB

4.20 STEP

4.20.1 Purpose

The Software Test and Evaluation Project (STEP), Phases III and IV, is sponsored by the Director Defense Test and Evaluation (DDT&E) and the STARS program. The purpose of STEP is to aid in the development and implementation of new DoD guidance and policy for the test and evaluation of computer software for mission critical applications. Principal subgoals include the stimulation of tool development, the support of policy development, and the identification of research issues and directions in the area of software testing. Principal recommendations from the previous STEP phases are intended to establish a chain of test planning, documentation, and evaluation criteria which starts at the most general planning document (the Test and Evaluation Master Plan, or TEMP) and proceeds through the plans and procedures implemented by the project offices, development organizations, and independent test organizations. Phases III and IV of STEP, which are currently underway, are designed to define the technology and provide implementation support for these recommendations.

4.20.2 Relationship to the E&V Task

There are at least three areas in which STEP and the E&V Task are related: a) STEP is tasked to develop new guidance statements, as needed, for software test and evaluation (T&E), as well as the necessary implementation methods. Work in this area is intended to address the policy-related issues so that the technology receives the support that is needed to put it into practice. This would include any modifications to DoDD 5000.3 and attendant Service regulations, etc., which would require TEMPs to report the results of the evaluation and validation of support software; b) STEP is tasked to produce T&E management and operating plans, and demonstration and qualification procedures. The procedures for inclusion of qualified tools in TEMP specifications and lower-level test plans will also be defined. These tasks address the technology-related problems involved in the qualification of software testing tools for DoD use. This is, in many ways, a subset of the work to develop the E&V technology; and c) STEP is tasked to provide functional requirements for APSE test environments. The requirements produced will need to be supported by the E&V technology.

4.20.3 Benefits to the E&V Task

The E&V Task will benefit from STEP's efforts in at least two ways. First of all, the E&V technology will receive the policy support needed which will accelerate its use. Secondly, efforts to develop E&V technology for application to testing tools should benefit from the qualification procedures developed by STEP.

4.20.4 Benefits to the Related Effort/Organization

The efforts of the E&V Task will allow the insertion of demonstrated risk reduction technology into the acquisition cycle. The qualification procedures developed by STEP will be elaborated and inserted into an environment where the standard operating procedures include the evaluation and validation of support software. Furthermore, the functional requirements for APSE test environments to be developed by STEP will be supported by a technology which ensures their implementation.

4.20.5 Impact on E&V Task Schedules

No schedule impacts are currently identified.

4.20.6 Impact on Related Effort/Organization Schedules

No schedule impacts are currently identified.

4.20.7 Required Level of Coordination

At present, Ms. Ronnie J. Martin is serving as STEP's liaison to the E&V Task. However, if both tasks are to capitalize upon the obvious opportunities for mutual benefit, additional mechanisms for increased coordination and support should be explored.

4.20.8 Resolution of Issues

Once such issues are identified, they should initially be addressed within the specific task in which the issues arose (i.e., STEP or E&V). Recommendations should then be developed within that task to resolve such issues. The issues, and associated recommendations, should then be presented to the other task. Task leaders should work together to obtain resolution. If resolution is unattainable at that level, both task leaders should elevate the issues for review by DDT&E, STARS, and AJPO personnel, as appropriate.

4.20.9 Focal Point

The DDT&E and STARS focal points, respectively, are indicated below :

Charles K. Watt

Director Defense Test and Evaluation

Room 3E (1060)

The Pentagon

Washington D.C. 20301

Commercial : (202) 695-7171

Joseph C. Batz

Acting Director, STARS Joint Program Office

Room 3D 139 (Fern St/C107)

The Pentagon, Washington D.C. 20301

Commercial : (202) 694-0208

Autovon : 224-0208

4.21 Tactical Ada Guidance

4.21.1 Purpose

The purpose of the Tactical Ada Guidance (TAG) program, which is sponsored by the Air Force Armament Laboratory (AFATL) and the Air Force Computer Resource Management Technology Program Element (64740F), is to demonstrate the use of Ada in a real-time armament system. Specifically, the software in the Medium Range Air-to-Surface Missile (MRASM) Test Instrumentation Controller (TIC) computer is being redesigned and implemented in Ada.

4.21.2 Relationship to the E&V Task

One of the by-products of this effort is the identification of Ada compiler implementation-dependent features that are particularly desirable for armament applications. These features may be useful in defining application-specific metrics for Ada compilers.

4.21.3 Benefits to the E&V Task

The TAG program will result in recommendations for application-specific (i.e., armament) evaluation criteria for Ada compilers.

4.21.4 Benefits to the Related Effort/Organization

No benefits are identified. The TAG program concluded in May 1985, prior to any E&V technology transition.

4.21.5 Impact on E&V Task Schedules

No schedule impacts are currently identified.

4.21.6 Impact on Related Effort/Organization Schedules

No schedule impacts are currently identified.

4.21.7 Required Level of Coordination

At present, Debra Harto is an active member of the E&V Team and coordinati
will be through her.

4.21.8 Resolution of Issues

Issues identified which relate to the E&V Task will be handled within the E&V
Task. Issues identified which relate to the TAG program will be resolved by the
AFATL TAG Program Manager.

4.21.9 Focal Point

The focal point is indicated below :

Chris Anderson

AFATL/DLCM

Eglin Air Force Base, Florida 32542

Commercial : (904) 882-2961

Autovon : 872-2961

4.22 Virginia Polytechnic Institute APSE Validation Effort

4.22.1 Purpose

The Virginia Polytechnic Institute (VPI) and State University APSE Validation
Effort is a research project conducted by the VPI Department of Computer Science
for the AJPO through the Office of Naval Research. The purpose of the effort is
to identify and address research issues related to, and supporting APSE
validation. Based on a position paper and proposal delivered to the KITIA
meeting in June of 1982 by Tim Lindquist, an initial effort addressing
validation needs in an APSE was conducted during the summer of 1982. This study
raised issues indicating the need for an APSE model able to accommodate
distributed and secure APSEs. It further indicated a need to address validation
of a kernel set of APSE facilities to achieve transportability of APSE tools.
Subsequent efforts on this project have detailed an APSE model based on the Open
Systems Interconnection (OSI) model, and have developed an Abstract Machine
approach to specifying the CAIS and a technique for developing a validation
suite from the specifications. The project is in the process of developing
specifications for the CAIS Node Model and Process Management sections, and will
undertake an operational definition of CAIS Version 1.4.

4.22.2 Relationship to the E&V Task

The KIT/KITIA-designed CAIS will become a MIL-STD in 1985. Further, a CAIS Validation Capability (CVC) will be developed through the E&V Task to determine conformance to the CAIS. The specification and validation techniques developed by the VPI APSE Validation Effort relate to both of these activities.

4.22.3 Benefits to the E&V Task

The VPI APSE Validation project specifications for the CAIS Node Model and Process Management sections, will serve as inputs to the development of a CVC. Whether the specifications generated are used for the CAIS, they isolate issues that must be addressed by the CVC. The Abstract Machine descriptions and the technique for generating test cases from the Abstract descriptions, can be used to identify areas the CVC must address. The operational definition of CAIS Version 1.4 will benefit the E&V Task by providing input to the CAIS Validation Capability (CVC) effort in constructing validation tests.

4.22.4 Benefits to the Related Effort/Organization

This project uses the E&V Team and the KIT/KITIA for review and feedback on its results.

4.22.5 Impact on E&V Task Schedules

The following VPI APSE Validation Effort schedules are of interest to the E&V Task :

September 1, 1984 -- Preliminary Abstract Description of CAIS Node Model

November 1, 1984 -- Preliminary Abstract Description of CAIS Process Management

4.22.6 Impact on Related Effort/Organization Schedules

The E&V Task schedules regarding the CVC impact this effort.

4.22.7 Required Level of Coordination

The Principal Investigator of the VPI APSE Validation Effort (Tim Lindquist), is a technical consultant to the E&V Task, and the E&V Team Chairperson (Raymon Szymanski), is the Project Monitor.

4.22.8 Resolution of Issues

The focal point for coordination will assist in resolving any issues that arise which may adversely affect either effort.

4.22.9 Focal Point

The focal point is indicated below :

*

Dr. Timothy E. Lindquist

Department of Computer Science

Virginia Polytechnic Institute and State University

Blacksburg, Virginia 24061

Commercial : (703) 961-7537 (961-6931 messages)

MILNET : LINDQUIST%VPI@RAND-RELAY

*

Note : Dr. lindquist has recently taken a position at Arizona State University. His new address is as follows:

Dr. Timothy E. Lindquist

Department of Computer Science

Arizona State University

Tempe, Arizona 85287

Commercial : (602) 965-2783

MILNET : LINDQUIS%ASU@CSNET-RELAY

4.23 WWMCCS Information System

4.23.1 Purpose

The World Wide Military Command and Control System (WWMCCS) Information System (WIS) effort is a Joint Service Program with the goal of modernizing the existing WWMCCS automatic data processing (ADP) and upgrading that system with new capabilities to satisfy developing requirements. The WIS program intends to use an Ada designed and implemented software first approach to the upgrade in order to make it portable to other hardware whenever it becomes necessary.

4.23.2 Relationship to the E&V Task

WIS is generating a very broad systems development and maintenance environment (SDME). This is to be active on multiple machines over a network. It will have control of code, data, and documentation at several applications levels. It is not a simple APSE, but may contain APSEs. The environment will be developed in Ada, starting from an existing program control and support system. A number of machine-independent tools are being assembled to provide an initial programming capability. Compilers will be evaluated and used as appropriate.

4.23.3 Benefits to the E&V Task

The WIS program will need to evaluate early in the program various tools and tool sets, as well as the tools they develop themselves. This will provide two opportunities for the E&V Team. First, the E&V Task should be able to adopt or learn from any technology for specific evaluations that WIS develops. Second, WIS can possibly be an early user of any E&V technology developed. Present WIS work includes compiler criteria and benchmarks to measure performance against those criteria.

4.23.4 Benefits to the Related Effort/Organization

The WIS program should be able to take advantage of any E&V technology that is developed.

4.23.5 Impact on E&V Task Schedules

The following WIS schedule may impact the E&V Task schedules.

Jan 86 - Internal delivery of SDME environment

4.23.6 Impact on Related Effort/Organization Schedules

No schedule impacts are currently identified.

4.23.7 Required Level of Coordination

The E&V Team should keep current electronic status from the WIS Joint Program Management Office (JPMO) on their plans and schedules. At present, Lt Patrick Sheridan is an active member of the E&V Team and will be responsible for coordination.

4.23.8 Resolution of Issues

Issues shall be addressed within the respective tasks (WIS or E&V). Recommendations should be developed within that task on how to resolve the issue and should be forwarded to the other team for consideration.

4.23.9 Focal Point

The focal point is indicated below :

Col William A. Whitaker

WIS JPMO/TE

Washington DC 20330

Commercial : (703) 285-5065

Autovon : 365-5065

MILNET : WWHITAKER@ECLB

1. Appendix A

1.1 Acronyms

ACVC	Ada Compiler Validation Capability
ADP	Automatic Data Processing
AFATL	Air Force Armament Laboratory
AFB	Air Force Base
AIE	Ada Integrated Environment
AJPO	Ada Joint Program Office
ALS	Ada Language System
ANSI	American National Standards Institute
APSE	Ada Programming Support Environment
ATVS	Ada Test and Verification System
AVO	Ada Validation Organization
CAIS	Common APSE Interface Set
CAMP	Common Ada Missile Packages
C3I	Command Control Communication and Intelligence
COORDWG	Coordination Working Group
CVC	CAIS Validation Capability
DDT&E	Director Defense Test and Evaluation
DoD	Department of Defense
DoDD	Department of Defense Directive
E&V	Evaluation and Validation

HRWG	Human Resources Working Group
IDA	Institute for Defense Analyses
IDAS	Integrated Design Automation System
IE&V	Independent Evaluation and Validation
ISA	Instruction Set Architecture
I&T	Interoperability and Transportability
JPMO	Joint Program Management Office
JSC	Johnson Space Center
JSSEE	Joint Service Software Engineering Environment
KAPSE	Kernel Ada Programming Support Environment
KIT	KAPSE Interface Team
KITIA	KAPSE Interface Team from Industry and Academia
MAPSE	Minimal Ada Programming Support Environment
MCF	Military Computer Family
MIL	Military
MOA	Memorandum of Agreement
MRASM	Medium Range Air-to-Surface Missile
NASA	National Aeronautics and Space Administration
NOSC	Naval Ocean Systems Center
OSI	Open Systems Interconnection
PA-APSE	Prototype Advanced Ada Programming Support Environment

PDL	Program Design Language
PE	Project Element
PPG	Program Planning Group
PUBWG	Public Coordination Working Group
RADC	Rome Air Development Center
SEATECS	Software Engineering Automation for Tactical Embedded Computer Systems
SEE	Software Engineering Environment
SEI	Software Engineering Institute
STARS	Software Technology for Adaptable Reliable Systems
STD	Standard
STEP	Software Test and Evaluation Project
TAG	Tactical Ada Guidance
TCSD	Technical Coordination Strategy Document
TECWG	Technical Coordination Working Group
T&E	Test and Evaluation
TEMP	Test and Evaluation Master Plan
TIC	Test Instrumentation Controller
USDRE	Under Secretary of Defense for Research and Engineering
VAX	Virtual Address Extension
VHSIC	Very High Speed Integrated Circuits
VMS	Virtual Memory System
VPI	Virginia Polytechnic Institute

WIS	WWMCCS Information System
WWMCCS	World Wide Military Command and Control System

11. Appendix B

11.1 COORDWG Members

Don Jennings, Chairperson
OC-ALC/MMECO
Tinker AFB, OK 73145-5990

James S. Williamson, Vice-Chairman
Air Force Wright Aeronautical Laboratories
WPAFB, OH 45433-6543

Debra Harto
AFATL/DLCM
Eglin AFB, FL

Patrick Maher (Distinguished Reviewer)
Magnavox Electronics Systems Co.
1313 Production Rd.
TC-10-C3
Fort Wayne, IN 46808

Jane Shirley
SYSTRAN Corp.
4126 Linden Ave.
Dayton, OH 45432

Betty Wills
CCSO/SKXD
Tinker AFB, OK

III. Appendix C

III.1 Related Technical Efforts Matrix

EVALUATION and VALIDATION

RELATED TECHNICAL EFFORTS MATRIX

(RTEM)

The following represents the Evaluation and Validation (E&V) Related Technical Efforts Matrix (RTEM), indicating members of the E&V Team, along with potentially related technical efforts/organizations with which each indicated member is involved. This matrix will reside in <EV-INFORMATION> and be continually updated as appropriate by the Coordination Working Group (COORDWG).

RELATED TECHNICAL EFFORTS MATRIX

(RTEM)

E & V TEAM MEMBER	1 Ada C3I TEST AND EVAL	2 Ada INTEGRATED ENVRMNT	3 Ada JOINT PROGRAM OFF	4 Ada LANGUAGE SYSTEM	5 Ada PDL EVAL GUIDELINE	6 Ada TEST AND VER SYS	7 Ada VALIDATION ORG	8 AIR FORCE COMP RES MAN	9 CAMP	10 JSC Ada PROJECT	11 KIT/KITIA	12 PA APSE	13 SEA TECS	14 STARS/APP THRUST AREA	15 ** STARS/BUS PRA THR AREA	16 ** STARS/HUMAN RES THR AR	17 ** STARS/MEA THRUST AREA
DEESE, A CAPT							X										
HARTO, D *								X	X								
HUMPHREY, T *										X							
KIRKPATRIC LT														X			
KEAN, L	X	X				X					X						
LINDQUIST, T *																	
MARTIN, R *																	
MYERS, P LCDR			X														
OBERNDORF, P											X	X	X				
SHERIDAN,P LT*																	
SZYMANSKI, R											X						
TAYLOR, G																	
WILLIAMSON, J				X													

* SEE RTEM (CONTINUED)

** E&V TEAM POINT OF CONTACT TO BE DETERMINED

	<> RELATED TECHNICAL EFFORT/ORGANIZATION <>																
E&V TEAM MEMBER	18 STARS/METH THRUST AR	19 ** STARS/SEE THRUST AR	20 STEP	21 TACTICAL Ada GUIDANCE	22 VPI APSE VALIDATION	23 WIS	24	25	26	27	28	29	30	31	32	33	34
	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
HARTO, D				X													
HENNE, M	X																
HUMPHREY, T																	
LINDQUIST, T					X												
MARTIN, R		X	X														
SHERIDAN, P LT						X											

** E&V TEAM POINT OF CONTACT TO BE DETERMINED

The following provides an appendix of acronyms and abbreviations as used in the above Related Technical Efforts Matrix (RTEM) :

APPENDIX

RTEM ACRONYMS/ABBREVIATIONS

APP	Application
APSE	Ada Programming Support Environment
AR	Area
BUS	Business
CAMP	Common Ada Missile Packages
COMP	Computer
C3I	Command Control Communication and Intelligence
ENVRMNT	Environment
EVAL	Evaluation
JSC	Johnson Space Center
KIT	KAPSE Interface Team
KITIA	KAPSE Interface Team from Industry and Academia
MAN	Management
MEA	Measurement
METH	Methodology
OFF	Office
ORG	Organization
PA	Prototype Advanced
PDL	Program Design Language
PRA	Practices
RES	Resource
SEATECS	Software Engineering Automation for Tactical Embedded Computer Systems
SEE	Software Engineering Environment
STARS	Software Technology for Adaptable Reliable Systems
STEP	Software Test and Evaluation Project
SYS	System
THR	Thrust
VER	Verification
VPI	Virginia Polytechnic Institute
WIS	WWMCCS Information System (Note : "WWMCCS" stands for Worldwide Military Command and Control System.)

APPENDIX B

EVALUATION AND VALIDATION
PUBLIC COORDINATION STRATEGY

VERSION 2.0

1 JULY 1985

TABLE OF CONTENTS

EXECUTIVE SUMMARY

The overall Evaluation and Validation(E&V) Task objective is to develop the technology for the evaluation and validation of APSEs. As the E&V technology is developed, it will be made available to the community for use by DOD components, industry, and academia as deemed appropriate by the respective organizations. In order to accomplish the open channel of communication to the public, the E&V Coordination Working Group (COORDWG) was established. This document describes the strategy for accomplishing the communications need for public awareness. The mechanisms for communication with the Public include:

(1) Briefing; (2) Papers; (3) E&V Workshops; (4) Quarterly E&V Team Meeting minutes; (5) E&V Status Reports; (6) Project Reference List; (7) E&V Public Report; and (8) Milnet E&V Directory.

1. INTRODUCTION

1.1 Objective

The purpose of the Evaluation and Validation (E&V) Task is to develop the techniques and tools which will provide a capability to perform assessment of Ada Programming Support Environments (APSEs) and to determine conformance of APSEs to the Common APSE Interface Set (CAIS). "Evaluation" represents a method of assessing the performance of APSE components. "Validation" represents a method of determining conformance to a standard which a method of determining conformance to a standard which is is applicable to an APSE(e.g., MIL-STD-1815A, CAIS, etc.) As the E&V technology is developed, it will be made available to the community for use by DoD components, industry, and academia as deemed appropriate by the respective organizations. The objective of the E&V Coordination Working Group (COORDWG) is to facilitate the transition of this technology to the public as it becomes available. This document describes the strategy for accomplishing this transition.

1.2 Background

Currently there is little data available on correlating specific APSE capabilities with project requirements. As stated in section 1.1, one of the goals of the E&V Task is to provide an evaluation capability for APSE components for which there exists neither a standard nor a method of determining conformance to that standard (e.g., Ada compiler implementation dependent features, run-time system characteristics, etc.) Since this information is critical to the success of the software development process, it is essential that emerging evaluation techniques resulting from the E&V Task be provided to the public as soon as they become available.

Similarly, in order to ascertain whether data and tools from one APSE will be able to be transported to another APSE, a metrics capability must be developed to enable the determination of conformance to the CAIS, which is expected to become a Military Standard in 1985. Again, this information is vital to software program managers and designers who plan to achieve maximum software portability and should be made available to the public as it emerges.

2. SCOPE

In order to accomplish the objectives described in section 1.1, it is essential that the E&V Team maintain open channels of communication to the public. The public coordination strategy outlined in this document primarily focuses on communication originating from the E&V Team to the public and associated feedback from this communication. Technical information related to the E&V Task originating outside the team will be monitored by the E&V Coordination Working Group (COORDWG) and transmitted to the appropriate E&V working groups. The mechanisms supporting the outflow of information from the team to the public are described in section 3 and appendices C-G.

3. APPROACH

Several mechanisms for communication with the public have been identified to assist team members in the public exchange process. These mechanisms are outlined in the following subsections.

3.1 Briefings

It may be appropriate for team members to occasionally present briefings to the public. A current set of "official" E&V vugraphs will be maintained by the COORDWG for use by any team member. Briefing from a standard set of vugraphs will ensure that each briefer will be presenting the E&V Task in an appropriate, consistent and accurate manner. However, following a presentation submission of the "Public Exchange Record" to the Chairperson, with a copy to the team is required. The format for this is listed in Appendix E and on MILNET USC-ECLB directory <EV-INFORMATION>.

Briefings may be resented to DoD organizations, committees and conferences as well as to industry and academia. A list of candidate organizations is presented in Appendix C, along with the primary mission and point of contact.

An E&V update will be presented by the Chairperson or designated alternate at the AdaJUG Government Corner and the SIGAda Environment/Standards subcommittee session, as warranted by the emergence of E&V technology.

3.2 Papers

Various team members may wish to submit E&V Task related papers to professional journals. The paper must be submitted to the E&V Chairperson for review and approval prior to publication. Since these papers will be subject to Government public release approval, substantial lead time should be allowed (4-6 weeks). A copy of the approved paper should be forwarded to the E&V Team Chairperson for inclusion in the annual E&V Team Public Report. A list of candidate publications is given in Appendix D, along with procedures for document submittal.

Also the author of the paper should complete the COORDWG form listed in Appendix E for inclusion in the E&V Project Reference List maintained by the COORDWG on the MILNET USC-ECLB directory <EV-INFORMATION>.

3.3 E&V Workshop

An E&V Workshop will be conducted on an annual basis throughout the duration of the E&V Task. The purpose of the E&V Workshop is to encourage industry/academia participation in the E&V effort, and participation will be limited. Those selected to participate will be extended an invitation to join the team as E&V Distinguished reviewers. Information on the proposed E&V Workshop will be made publicly available and participants will be selected on the basis of position papers which are written relevant to the technical aspects of the E&V Task.

3.4 Quarterly E&V Team Meeting Minutes

The minutes of each quarterly E&V Team meeting will be recorded by the COORDWG and submitted for team comment and approval prior to entry in the <EV-INFORMATION> directory under file names "MINUTES-MONTH-YEAR" with MONTH being three letters, YEAR being two digits beginning in "DEC-83" and continuing at three month intervals. A hardcopy of the minutes will also be incorporated in the annual E&V Team Public Report. The format for the minutes is presented in Appendix F.

3.5 E&V Status Report

The COORDWG will prepare a brief E&V Status Report base' on the E&V Minutes. After being reviewed by the team and approved by the E&V Chairperson, the report will be published in the Ada Information Clearinghouse Newsletter, the LCF Newsletter, put on the MILNET USC-ECLB <EV-INFORMATION directory and be available for publication or for "handouts" at appropriate conferences.

3.6 Project Reference List

A list of E&V related documents will be kept in the Project Reference List on the MILNET USC-ECLB <EV-INFORMATION> directory. This list will be maintained by the COORDWG. Team members should contribute to the list by filling out the template listed in Appendix E and sending it to the Chairperson with a copy to the team. The list will not only inform the public about various E&V related studies, but also keep the team up to date on any related technology.

3.7 E&V Public Report

An E&V Team Public Report will be published annually in order to provide the public with information on the activities of the E&V Team. The E&V Team Public Report will contain the minutes of all E&V Team meetings as well as all position papers prepared by E&V Team members. The E&V Team Public Report will also contain position papers written by industry/academia participants in the annual E&V Workshop, as well as all documentation which results from the E&V Workshop.

3.8 MILNET E&V Directory

Another mechanism for communication with the public is via the MILNET USC-ECLB <EV-INFORMATION>directory. Use of this directory (with password "EV") is available to anyone with Milnet access. Included in this directory are the minutes from the quarterly E&V Team meetings, and abbreviated E&V Quarterly Report, the E&V Project Reference List, E&V Team member listing, and other pertinent E&V related information.

Files which are referenced through the <EV-INFORMATION> directory are actually stored in another directory, <EV-INFO>, to which access is not available. All files in <EV-INFO> have an extension of ".HLP". The public may access files by one of the following methods: (1) loggin into the <EV-INFORMATION> directory and using the HELP facility to read the files; and (2) copying the HELP facility to read the files; or (2) copying files (specifying the ".HLP" extension) using the TYPE or FTP facilities directly from the <EV-INFO> directory without actually logging into the directory.

APPENDIX A

ACM	Association for Computing Machinery
AdaIC	Ada Information Clearinghouse
AdaJUG	Ada Jovial Users Group
AFLC	Air Force Logistic Command
AFSC	Air Force Systems Command
AIAA	American Institute of Aeronautics and Astrcnautics
AJPO	Ada Joint Program Office
APSE	Ada Programming Support Environments
CAIS	Common APSE Interface Set
COORDWG	E & V Coordination Working Group
ECR	Embedded Computer Resource
EIA	Electronics Industries Association
IEEE	Institute of Electrical and Electronics Engineers
LCF	Language Control Facility
NSIA	National Security Industrial Accociation
NTIS	National Technical Information Service
SIGAda	Special Interest Group Ada

APPENDIX B

MEMBERS

Don Jennings OC-ALC/MMECO Tinker AFB, OK 73145-5990	Chairperson
Jimmy Williamson AFWAL/AAF-2 Wright-Patterson AFB, OH 45433	Vice-Chairperson
Debra Harto AFATL/DLCM Eglin AFB, FL	
Betty Wills CCSO/SKXD Tinker AFB, OK 73145	
Patrick Maher Magnovox Electronics Systems Co. 1313 Production Rd Ft Wayne, IN 46808	
Jane Shirley Systran Corporation Dayton, OH	

APPENDIX C

Organizations

The following organizations have been identified as possible candidates for E&V related presentations. DoD and industry/academia organizations are listed separately.

DoD Organizations and Conferences

Air Force Systems Command (AFSC) Embedded Computer Resource (ECR) Focal Group

This group consists of representatives from the AFSC laboratories and product divisions associated with embedded computer resources. Meetings are held approximately three times yearly. Attendance is usually limited to members. To present a special briefing of interest contact your AFSC ECR focal point or Maj Chuck Lillie, AFSC/ALR, Wright-Patterson AFB, OH, AV 785-6941, (513) 255-6941.

AFSC Software Technology Coordinating Group

This group consists of representatives associated with software technology from the AFSC laboratories. Meetings are held approximately four times per year. Only official members may attend. To present a special briefing contact your AFSC representative or Capt. Sunny Riley, AFSC/DLA, Andrews AFB, DC 20334, AV 858-2482, (301) 981-2482.

Armament/Avionics Standardization Conference

This annual conference (usually in Sept) is jointly sponsored by AFSC and AFLC. Candidate presentations should conform to panel issues. The chairperson of the Standardization Panel is Robert Earnest ASD-AFALC/AXTS, Wright-Patterson AFB, AV 785-5945.

Ada Information Clearinghouse (AdaIC)

Under the direction of the AJPO, the AdaIC was established to facilitate the transfer of information to the Ada user community. The Clearinghouse is expected to serve as the focal point for collecting and disseminating information. The Information Clearinghouse announces activities including upcoming conferences, seminars, classes, and general information on Ada via <ADA-INFORMATION> an on-line file accessible through ARPANET or TELENET. The IC also offers reference services via Net mail, telephone, or postal mail. Pofessional information scientists, knowledgeable of Ada activities, update the <ADA-INFORMATION> files an respond to inquiries. AJPO invites inquiries through one of the following channels:

Net Mail:	<ADA-INFORMATION>
Postal Mail:	Ada Information Clearinghouse P.O. Box 849 Rome, NY 13440
Telephone:	(315) 336-2359
Postal Mail:	Ada Information Clearinghouse 1211 S Fern RMC107 Arrlington, VA 22202 3d139 (400 AN)
Telephone:	(202) 694-0210 (703) 685-1477 AV 224-0210

Industry/Academia Organizations and Conferences

SIGAda

This professional association meets three times yearly. Technical topics associated with the use or implementation of Ada are welcome. For more information concerning the appropriate session in which to present a briefing, contact Jean Whitaker, Hughes Aircraft Co., (714) 732-9231.

AdaJUG

This professional organization brings together representatives from industry, academia and the Government, interested in standardization and language control activities, compilers and tools; applications and development efforts associated with JOVIAL and Ada. The Government Corner is an appropriate session to brief short presentations (15-20 min) concerning Government sponsored Ada activities. The Chairperson of this session is Lt Col Joseph W. Dangerfield, ASD/AXT, Wright-Patterson AFB, OH 45433, AV 785-5941, (513) 255-5941. For longer presentations contact the AdaJUG Chairperson Donna Gant at (916) 920-3663.

National Security Industrial Association (NSIA)

Members of the association are defense contractors. Open national conferences focusing on special topics are held several times a year. For further information contact W. M. McMurray, General Dynamics, at (314) 851-8910.

Institute of Electrical and Electronics Engineers (IEEE)

This professional organization has over 210,000 engineers and scientist members. There are numerous meetings and special technical conferences held annually. For more information contact Eric Herz, Executive Director, at (212) 705-7900, or write 345 East 47th Street, New York, NY 10017.

Association for Computing Machinery (ACM)

This professional organization has over 53,000 members associated with computing and data processing. There are over 31 special interest groups. National conferences are held annually (usually in October). For more information contact Sidney Weinstein, Executive Director, at (212) 869-7440 or write 11 West 42nd Street, 3rd Floor, New York, NY 10036.

Electronics Industries Association (EIA)

American Institute of Aeronautics and Astronautics (AIAA)

APPENDIX D

Publications

The following publications have been identified as candidates for publishing E&V related papers. Procedures for document submittal are also included.

Computer Magazine (IEEE)

Articles that cover all aspects of computer science are welcome. Articles are usually survey or tutorial in nature. Submit six copies of the manuscript including illustrations, references, and authors' biographies to the Editor-in-Chief:

Stephen S. Yau
Dept. of EE and Computer Science
Northwestern University
Evanston, IL 60201
Telephone: (312) 492-3641

Defense Electronics

Articles covering aspects of computer science that are relevant to the DoD community are welcome. Send to:

EW Communications, Inc.
1170 East Meadow Drive
Palo Alto, CA 94303-4275
Telephone: (415) 494-2800

Ada Letters (ACM)

Information dealing with all aspects of Ada are welcome. "Short Notices" announce meetings or publications. "Letters to the Editor" raise issues or answer them. Articles typically deal with indepth technical topics related to the use of Ada. "Ada Events" announce significant events of major interest to the Ada community.

Short Notices
Letters to the Editor

Dr. Kaye Grau
Harris Corporation (GISD)
150 Wickham Road
Melbourne, FL 32901
Telephone: (305) 676-6446

Articles

Ronald F. Brender
DEC
110 Spit Brook Rd
ZK02 - 3/N30
Nashua, New Hampshire 03062
Telephone: (603) 881-2088

Ada Events

Robert I. Eachus
Honeywell SSPD
300 Concord Road
Billerica, MA 01821
Telephone: (617) 671-2907

Submissions should be single-spaced with no page numbers, and may be printed two-up. Submission deadlines Aug 31, Oct 31, Dec 31, Feb 28/29, Apr 30 and Jun 30.

Communications of the ACM

This publication serves as a newsletter to members about activities of the Association of Computing Machinery and as a publication medium for contributed technical papers and other material of interest. Papers on all aspects related to computing science are solicited. In particular, research contributions in all areas are welcomed. Manuscript format is provided in the July 1982 issue, pages 507 - 508. Submit to:

Nicolas Mokhoff
ACM Headquarters
11 West 42nd Street
New York, NY 10036
Telephone: (212) 869-7440

JOVIAL Language Control Facility Newsletter

Brief articles or announcements related to Ada activities are welcome. Submit to:

ASD/ADOL
Wright-Patterson AFB, OH 45433
(513) 255-4472/4473
AV 785-4472
LCF at WPAFB-JALCF

Ada Joint Program Office Newsletter

Brief articles or announcements related to Ada activities are welcome. Submit to:

Technical Research Institute
c/o Ada IC
P.O. Box 849
Rome, New York 13440

Appendix E

There are three forms for the submission of data to the team: the Public Exchange Record, the Project Reference List Submittal Form, and the E&V Working Group Status Report.

Public Exchange Record*

TYPE OF EXCHANGE: (briefing, paper, etc.)

SPECIFIC TOPIC:

DATE:

PLACE OR PUBLICATION:

ATTENDEES:

PRESENTER:

MATERIAL PRESENTED: (Brief synopsis of technical content of presentation)

FEEDBACK:

*This form shall be completed within one week of the presentation. Send a copy to the entire E&V Team for information purposes.

Project Reference List Submission Form*

TITLE:

DATE:

AUTHOR(S):

AFFILIATE:

SPONSOR:

ABSTRACT:

RELATIONSHIP TO E&V:

TO ORDER:

*This form should be submitted by individual who has found the relevant material and desires to have it included in the list. Cc to the E&V Team membership.

APPENDIX F

E&V Minutes Format

The minutes of E&V quarterly meetings are presented on the MILNET USC-ECLB <EV-INFORMATION> directory in files "MINUTES-MONTH-YEAR" with MONTH being three letters, YEAR being two digits, beginning in "DEC-83" and continuing at three month intervals.

Minutes

of the

EVALUATION & Validation (E&V) MEETING

Date

1. Date

1.1 Topic 1

1.2 Topic 2

1.N Topic N

2. Date

2.1 Topic 1

2.2 Topic 2

2.N Topic N

3. Action Items

Appendix A Acronyms

Appendix B E&V Meeting Attendance

E&V WORKING GROUP STATUS REPORT

This report shall be submitted by working group chairs to the E&V Chairperson at the quarterly E&V meeting. This report will be incorporated as part of the E&V Team minutes.

WORKING GROUP:

DATE:

PERSONNEL: (List Chair, Vice Chair, and members. Note changes.)

DELIVERABLES DUE THIS QUARTER:

ACCOMPLISHMENTS THIS QUARTER:

KEY ISSUES ADDRESSED DURING THIS QUARTER:

UNRESOLVED PROBLEMS OR ACTION ITEMS:

PROJECTED WORK FOR NEXT QUARTER:

DELIVERABLES DUE NEXT QUARTER:

PRESENTATIONS PLANNED FOR NEXT MEETING:

OTHER SIGNIFICANT INFORMATION:

technical aspects of the E&V Task.

B-5

Appendix G
Document Format

This is the format to be used in all documents produced by the E&V Team.

E&V Title Sheet

Version X.X

Date

WORKING PAPER _ NOT APPROVED

This is an unapproved draft and subject to change. Do not specify or claim conformance to this document.

The Task for the Evaluation & Validation of Ada* Programming Support Environments (APSEs) is sponsored by the Ada Joint Program Office(AJPO)

(This version is for disatribution to E&V Team members only)

*Ada is a Registered Trademark of the US Government (Ada Joint Program Office).

Table of Contents

APPENDIX C

MINUTES

of the

EVALUATION & VALIDATION (E&V) MEETING

5-7 December 1984

TABLE OF CONTENTS

1.0 Wednesday, 5 December 1984

1.1 Welcome and General Business

The E&V meeting began with a welcome by the Chairperson, Jinny Castor. Her new supporting secretary, Ms. Rilla Pezzopane, was introduced, followed by self-introductions of all E&V Team Members.

A list of files on the <ev-information> directory was discussed. The newest addition to the directory is the file <AIMS-PHASE-I> which describes 3 aspects of Ada's application to real-time avionics software.

It was announced that Systran Corporation, a local Dayton company is performing documentation and administrative tasks in support of the E&V Team. Systran is helping to organize the 1985 E&V Workshop, and they are consolidating all of the E&V deliverable documents in order to compile the E&V Public Report.

The 1985 E&V Workshop will be held in Airlie, Virginia from 8-12 July. An announcement soliciting position papers from industry will appear in the Commerce Business Daily sometime in March.

1.2 STARS - Software Technology For Adaptable, Reliable Systems

Major Charles W. Lillie from Headquarters Air Force Systems Command gave a presentation concerning Software For Adaptable, Reliable Systems (STARS). STARS is an OSD program directed by Dr. Robert Mathis. STARS is supported by all three Services as well as by other government agencies such as NASA.

Major Lillie began by discussing the need for the STARS program. He outlined the following topics to be discussed during the remainder of his presentation: (1) the development of software system standards, (2) the organization of the STARS committee, (3) the support STARS will provide for related projects.

1.2.1 STARS Objective

The objective of the STARS organization is to establish the development and transition of mission critical software technology in order to increase defense system software productivity. STARS is interested in the total system development including requirements analysis, design, implementation, and maintenance. It is also interested in improving the software acquisition process and software program management.

STARS is striving to correct the problems of inefficient cost estimates, low software reliability and low software productivity. Its goal is to achieve an increase in productivity of 10 to 1 in mission critical computer software. STARS proposes to define a set of software standards to increase software productivity, system reliability and adaptability.

1.2.2 Software Standards

Since it has been proven that it is less costly and less timely to make a change in the software than to manipulate the hardware to achieve the same result, it is imperative that high quality software is produced. In order to improve the quality of software, one needs to apply standards to the requirements analysis and design phases of software development. If more time was spent in the front-end development of software systems (such as requirements analysis and design) the development time would be reduced. Therefore, the need exists to develop standards for software systems. Standards will decrease development time, improve productivity, and reduce maintenance costs. Standards will also aid in transporting software packages across various systems and programs. The results achieved by developing standards will help to attain the projected order of magnitude increase in productivity.

1.2.3 Organization of STARS

The STARS program is headed by Dr. Robert Mathis. The Air Force STARS Program Manager is Col. K. Nidiffer; the Army STARS Program Manager is Col. R. Stanley; the Navy STARS Program Manager is Ms. C. Morgan. Each Service has two thrust areas that it supervises. The six thrust areas or area coordinating teams are as follows: Measurement and Metrics, Business Practices, Methodology, Human Resources, Software Engineering Environments, and Applications.

1.2.3.1 Measurement and Metrics

One of the Air Force thrust areas, the Measurement and Metrics area coordinating team is chaired by Joe Cavano. This team's task is to develop the techniques for measuring software. The team investigates software production methods, and methods for measuring software quality and software reliability. Another area of interest is the productivity of the STARS program. The team follows the STARS program to determine if it attains its projected goals, and to determine if the 10 to 1 increase in productivity is reached.

1.2.3.2 Business Practices

The second Air Force thrust area is the Business Practices area coordinating team. The chairperson of this team is Phil Babel. The task for this team is to define methods to manage software. Areas of concern include software acquisition, software program management, and automated work stations. The team is also investigating tools to automate acquisition management.

1.2.3.3 Methodology

The methodology area coordinating team is supervised by the Army and is chaired by Pete Fonash. This team's objective is to investigate the development methods for software. They are researching software life cycle models to improve existing development methods.

1.2.3.4 Human Resources

The second thrust area for the Army is the Human Resources area coordinating team. The chairperson of this team is Joe Kernan. The team's objective is to investigate the methods employed to educate people in software engineering, and the career paths for military and civil service personnel involved with mission critical computer systems. This team investigates the military's definition of software engineering and the methods used to categorize a software engineer.

1.2.3.5 Software Engineering Environments

The Software Engineering Environments area coordinating team is supervised by the Navy. The chairperson is Phil Andrews. The objective of this group is to investigate the development of software engineering environments. The team is involved mainly with the Joint Services Software Engineering Environment (JSSEE), although they follow other environment development efforts also.

1.2.3.6 Applications

The second thrust area for the Navy is the Applications area coordinating team which is chaired by Bob Kolacki. The team is interested in the development and management of generic software. They are studying the concept of generic libraries, the methods for documenting generic software, and the problem of tracking the changes made to a generic software package.

1.2.4 STARS Support

STARS will provide support and funding to projects that can show that they are satisfying the objectives of one of the thrust areas. STARS is interested in projects that can be used across the DoD and are likely to be submitted for competitive bids. If one has a proposal to be submitted, they can send it to HQ AFSC/ALR, or send it to Major Lillie at the following address:

Major Charles Lillie
HQ AFSC/ALR
Andrews AFB, MD 20335-5000

If anyone has questions about STARS or requires further information they can contact Major Lillie at Av 858-6941, commercial (301) 981-6941, or on the Net at LILLIE%AFSC-HQ.

1.3 Joint Services Software Engineering Environment (JSSEE) Operational Concept Document (OCD)

Mr. Daniel Green from the Naval Surface Weapons Center (NSWC) in Dahlgren, Virginia, discussed the Joint Services Software Engineering Environment (JSSEE) Operational Concept Document (OCD). JSSEE is the major product of the STARS Software Engineering Environment task area.

The topics Mr. Green outlined for his presentation are the following: (1) describe the DoD-STD-SDS, (2) define JSSEE, and (3) discuss OCD-JSSEE.

1.3.1 DoD-STD-SDS

JSSEE is heavily based on the DoD Standard on Defense System Software Development (DoD-STD-SDS). DoD-STD-SDS establishes a uniform software development process which is applicable throughout the system life cycle. The DoD-STD-SDS covers the total system lifecycle starting with concept exploration, through demonstration and validation, and including full-scale development followed by production and deployment. DoD-STD-SDS provides the overall methodology that JSSEE is designed to support.

1.3.2 JSSEE

JSSEE is defined as an integrated set of methods, procedures, and supporting computer programs that are needed to develop and support mission critical computer resources (MCCR) software. JSSEE methods for carrying out software development and support process designed to allow the Services to add methods that are unique to their projects. The JSSEE methods are provided as standard/default for each particular activity. JSSEE's procedures refer to the documentation that describes the implementation model. The supporting computer programs are the software that automates or supports a particular method. JSSEE is designed to support any piece of software which is critical to the primary missions of the three Services and of the other supporting agencies.

1.3.2.1 JSSEE Goals

JSSEE goals are closely integrated into the goals of the STARS project. The goals of the implementation of JSSEE are to improve personnel productivity, to improve software quality, and to decrease the development/change time. JSSEE is also expected to help increase the predictability of cost and schedule.

1.3.3 OCD-JSSEE

The OCD is one of three major documents that are required for the full-scale development of JSSEE. The OCD describes the conceptual operation of the system. The second document, the System Segment Specification (SSS), defines the specifications to which JSSEE will be built. The last major document is the Computer Resources Life Cycle Management Plan (CRLCMP). The CRLCMP indicates how the government plans to provide support over the life cycle of the system.

The OCD takes the place of the formal mission statement for JSSEE. It was developed as a user-oriented document. It provides a description of JSSEE as it appears to users and a description of how users will interact with JSSEE. It also presents the assumptions for the operation of tasks.

1.3.3.1 Scope

The scope of the OCD is to express the software engineering environment as it will be required for large projects. Particular attention is being focused on defining functions that need to be formed and deciding the extent to which those functions can be automated. Once the feasibility of automating functions is determined, JSSEE will attempt to provide the tools required to perform the automation.

1.3.3.2 Mission

The JSSEE mission is divided into two segments. The primary mission is to support the MIL-STD-SDS development approach, all software development activities, and post deployment use and development. Software management and the production of all software products are also included in the primary mission. The secondary design goals are to support the following: (1) selected system design activities, (2) selected system management activities, and (3) part of the system test and integration.

1.3.3.3 Installation Organization

In order for JSSEE to operate, it presumes that it is built upon a set of computers, probably a distributed system. These computers have their own operating systems with terminals connected by a communication network.

JSSEE provides a set of methods that it supports (such as the Harness method), the documentation describing the methods, and a set of tools with associated documentation describing how to use the tools to implement the models in that particular work area. JSSEE tools will include a word processor, database management system, and an Ada compiler. JSSEE was developed with the idea that installation managers could add methods unique to their installation. Therefore, although JSSEE provides an Ada compiler, if another compiler was more fitted to a particular installation, the manager could substitute one for the other without restructuring the entire system.

1.3.3.4 Characteristics

JSSEE has some unique system characteristics. It is based on Ada, but since many larger systems were developed in other languages such as CMS-2, JOVIAL or Fortran, JSSEE has multilingual capabilities. Since JSSEE was designed for multiple hosts and multiple target computers, many JSSEE installations will develop each tailored to specific Services. Another JSSEE characteristic is that it was designed to support 1 to 50 projects and 10 - 1000 users per installation.

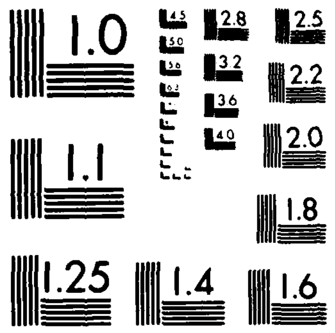
1.4 Ada Language System / Navy (ALS/N)

Mr. Tom Conrad from the Naval Underwater Systems Center (NUSC) presented an overview of the Ada Language System / Navy (ALS/N). The ALS/N is a minimal programming support environment based on the Army's Ada Language System (ALS). The ALS/N goal is to provide a full Ada capability for the Navy standard target computers, the AN/UYK-43, AN/UYK-44, and AN/AYK-14. This full capability includes support for both program development activities and run time activities. The topics Mr. Conrad outlined for his presentation are the following: (1) the status of the ALS/N, (2) the basis of ALS/N on the ALS, (3) the structure of the ALS/N, (4) the ALS/N design goals, and (5) the characteristics of the ALS/N.

1.4.1 Status of ALS/N

The system specification for the ALS/N is complete. A draft dated September 1983 is available. The program performance specifications (PPS) are in final review, and the ALS/N is currently undergoing a FY85 competitive procurement. In addition, an effort is underway to develop a pilot production capability for the UYK-44 which includes certain of the ALS/N components. This information was based upon currently public status (i.e., 9/83 public disclosure).

AD-A172 343 EVALUATION AND VALIDATION (E&V) TEAM PUBLIC REPORT 2/5
VOLUME 2(U) AIR FORCE WRIGHT AERONAUTICAL LABS
WRIGHT-PATTERSON AFB OH R SZYMANSKI 30 NOV 85
UNCLASSIFIED AFWAL-TR-85-1016-VOL-2 F/G 14/2 NL


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3

1.4.2 Basis of ALS/N on the ALS

The ALS/N is built upon the basic ideas and components of the ALS. The ALS/N includes the same database structure, the same user interface with some extensions and the same KAPSE with additional extensions. The ALS/N will capture 7 ALS functions intact and will include another 10 ALS functions that will need some modifications.

1.4.3 The Structure of the ALS/N

The ALS/N consists of a programming support environment (MAPSE) on the host computer, and a run time environment (RTE) that resides on the target computer.

1.4.3.1 Programming Support Environment

The MAPSE user interacts with the user access support system. This system consists of the command language processor, the file administrator, the HELP facility, and the environment data manager. The user access support system interacts with four other support systems via the command language processor interface. The four support systems are the following: (1) language processor support, (2) separate compilation support, (3) code manipulation support, and (4) MTASS interface support. These four systems communicate with the MAPSE run time environment (RTE) via a KAPSE interface, and the MAPSE RTE communicates directly to the host operating system. A third communication interface is the Container Data Manager which connects the language processor system to the separate compilation system, and the separate compilation system to the code manipulation system.

1.4.3.2 Run-time Environment

The run-time environment has three parts. One section is the user-written Ada programs. The second part is the run-time application support which consists of the run-time performance measurement aids, the run-time debugger, and the run-time loader. The third part of the environment is the run-time operating system. This system consists of the run-time support library and the run-time executive.

1.4.4 Design Goals of the ALS/N

One of the design goals of the ALS/N is to provide a built-in capability to transmit an application program from the MAPSE environment on a host computer to the RTE on a target computer for the purposes of execution or debugging. This will be accomplished by using the text_io package to achieve communication via the embedded target computer interface. This will be used to support single and multiple target environments based on UYK-44, UYK-43, and AYK-14.

A second design goal is to provide a built-in capability for direct exchange of environment database components with remote sites also employing ALS/N. This will be accomplished with an interhost telecommunications interface that will support transmit and receive commands of the file administrator. It will also support direct host-to-host links for ARPANET and DECNET file transfer protocols.

1.4.5 Characteristics of the ALS/N

Some distinguishing characteristics of the ALS/N are its approach to multi-lingual support, multi-level security, extensibility, and run-time support.

1.4.5.1 Multi-lingual Support

The ALS/N provides full support for Ada program development, and it includes partial support for earlier Navy languages such as CMS-2. This was built-in to provide some transition capability from currently used languages to Ada. The ALS/N supports separate CMS-2 code development using existing MTASS/L tools within the ALS/N MAPSE. It also permits importation of MACRO/L and MACRO/M code, developed according to certain interface specifications, to be linked with Ada code.

1.4.5.2 Multi-level Security (MLS)

Due to the cost, the expected performance impact, and the technical risk assessment, the decision was to defer MLS. Therefore, ALS/N does not support multi-level security.

1.4.5.3 Extensibility

Since the ALS/N is heavily based on the ALS, if the ALS/N succeeds then it proves the extensibility of the ALS core elements such as the database structure, and the chief interfaces to the user system and to the operating system. The notion of "Project Interface" is incorporated into the ALS/N such that project-specific tools and command scripts for enforcing project-specific methodologies are expected to be overlaid on the basic ALS/N tool set. The practical limits on extensibility may be the ability of the host computer system to provide enough processing power and storage capacity to support particular methodologies.

1.4.5.4 Run-time Support

The ALS/N run-time support is considered as critical to the system as the compile time support. The ALS/N must support all configurations of the Navy standard target computers and must provide a user-configurable run-time environment. It must include provisions for user programs to directly access some run-time executive services. The ALS/N run-time support is also required to provide a direct link between the host and target systems to permit loading and debugging.

The general session of the E&V meeting was adjourned so that working groups could meet separately. Working groups met separately through Thursday.

2.0 Friday, 7 December, 1984

2.1 Working Group Status Reports

2.1.1 REQWG Status Report

The REQWG Status Report was given by Dr. Tim Lindquist, chairperson of the group. No personnel changes were noted. No deliverables were due this quarter. Work is proceeding on Version 2.0 of the Requirements Document. The major change in the document is that version 2.0 will deal with a functionality-based taxonomy rather than the tool-oriented taxonomy presented in version 1.0. A draft version 2.0 of the Requirements Document is planned for release during the E&V meeting in June 1985. The group also started work on the Tools and Aids Requirements Document. Key issues addressed were the audience and purpose of the document. Suggestions for the purpose and contents of the document were discussed.

2.1.2 APSEWG Status Report

The APSEWG Status Report was presented by Liz Kean, the chairperson. No personnel changes were noted. No deliverables were due this quarter. Projected work includes evaluating the ALS, ALS/N and the AIE environments against the SEE taxonomy. The APSEWG will use the REQWG attributes to determine the strengths and weaknesses of the three environments. Due to the relationship between the SEE taxonomy and the REQWG attributes, Liz expects the APSEWG and REQWG will be working closely during the next quarter. It was decided that during the review of the three environments no comments will be publicly released. A complete report of the conclusions reached will be publicly released when the review process is completed.

2.1.3 COORDWG Status Report

The personnel from the PUBWG and TECWG were combined to form a new working group, the COORDWG. Chairperson and Vice-chairperson of the group are Don Jennings and Jimmy Williamson, respectively. The team members include Paul Dobbs, Debra Harto, Don Jennings, Randal Leavitt, Patrick Maher, Mark Mears, Capt. John Taylor, Jimmy Williamson, and Betty Wills. This group is responsible for all public and technical coordination efforts. The deliverables due this quarter were the minutes of the last E&V meeting, and the E&V Status Report. Work was accomplished on updating the Technical Coordination Strategy Document. It was announced that there will no longer be any condensed minutes. There will be only two forms of minutes available, the status report and the full minutes. Randal Leavitt is consolidating a list of tools and techniques for software evaluation. These tools can be for any language, not only Ada. He requests that any helpful information be sent to him.

2.1.4 CAISWG Status Report

The CAISWG Status Report was given by Lt. Darleen Sobota, the chairperson. No personnel changes were noted. However, Tim Lindquist sat in on part of the working session and provided valuable comments. Darleen encouraged him to continue to offer any helpful ideas he has. The group is working on the updated draft version 1.0 of the APSE Validation Procedures Document. The group is spending many hours reading and trying to understand version 1.4 of the CAIS Draft Military Specification. In order to help overcome the complexities of the CAIS Mil Spec, the group is working on a dependencies graph for CAIS models/nodes. A presentation is planned for the next E&V meeting to discuss either the CAIS 1.4 document or the APSE Validation Procedures Document.

2.2 Individual Presentations

Due to the large turnover of E&V team members, each member was encouraged to present a short briefing on himself. From these presentations, it is hoped that team members will become better acquainted and that the Team will become more united.

2.2.1 Christine M. Anderson (Reported by Debra Harto)

Chris Anderson was the representative from the Air Force Armament Laboratory (AFATL) at Eglin AFB. She was chairperson of the PUBWG and was responsible for writing the E&V meeting minutes. Chris participated in the E&V Workshop by acting as the chairperson of the E&V Workshop Recommendations Working Group. Chris consolidated the information exchanged at the Workshop and produced the Workshop Recommendations Document.

2.2.2 Michael Bridges / Jim Parlier

Michael Bridges is a distinguished reviewer representing General Dynamics (GD). He is the alternate for Jim Parlier who is a member of the REQWG. Michael's section of GD is the Ada focal point, and his participation in the E&V team allows him to provide better consulting services on Ada environments to other GD organizations.

2.2.3 Bard S. Crawford

Bard Crawford is a distinguished reviewer representing TASC. At the E&V Workshop, Bard participated in the REQWG and helped produce the E&V Workshop Requirement Document. Currently, Bard is an active member of the APSEWG. At TASC, Bard is the focal point for Ada and STARS activities.

2.2.4 Capt. Albert Deese, Jr.

Capt. Deese is from the Language Control Facility (LCF) at WPAFB where he has been involved in validating Jovial and Ada compilers. He is currently a member of the APSEWG and is helping to evaluate the ALS against the JSSEE taxonomy. From his past experience in the software development and performance analysis of WWMCCS (World Wide Military Command and Control System), Capt. Deese has an excellent background to provide helpful inputs to the APSEWG.

2.2.5 Nelson Estes

Mr. Estes is a past CAISWG chairperson. His current E&V interest is on Ada compiler performance evaluation as well as participating on the CAISWG. His normal job function is program manager for the Phase II Ada-1750A production quality compiler. He has additionally spent time on Ada transportability moving the Mathlib packages to the Data General APSE and encouraging independent Ada-1750 compilation systems. Additional future work will include changing MIL-STD-1750 so that it better supports Ada.

2.2.6 Richard Fleming

Richard Fleming is from the Space Division/ALR, a product division involved in the acquisition and development of space systems. As a member of the REQWG, Richard contributed the Command Language Interpreter Section of the Requirements Document. Since his work at Space Division includes analyzing the real-time performance of Ada compilers, Richard provides useful feedback to acquisition and development issues.

2.2.7 Kathleen A. Gilroy

Kathleen Gilroy is a representative of Software Productivity Solutions, Inc. (SPS), active in the areas of software engineering, Ada, and support environments. Planned products include a proprietary environment supporting Ada projects, and Ada software components for a variety of application domains. As a distinguished reviewer and member of the REQWG, Kathy contributed to the development of the E&V Requirements Document. From her participation on the E&V Team, Kathy is able to provide guidance on in-house and customer selection of environments, and guidance on development of proprietary products.

2.2.8 Bud Hammons

Bud Hammons is a distinguished reviewer representing Texas Instruments. Bud is a member of the CAISWG, and his major contribution to the group was the organization of the E&V Reference Manual. Bud's future work will involve studying the CAIS. He feels that his interaction with the E&V team helps to promote the transfer of technology within his home organization.

2.2.9 W. W. Happ / John Miller

Bill Happ is the alternate for John Miller who represents McClellan AFB. As a member of the REQWG, Bill has received information that is helpful to his efforts at McClellan. Bill and John organized and taught a 40 hour course in Ada for embedded computer systems. Bill also acts as a coordinator between activities at McClellan and the E&V team.

2.2.10 Marlene Hazle

Marlene Hazle is a representative of the MITRE Corporation, a Federal Contract Research Center (FCRC). Marlene is a member of the REQWG and was responsible for the Configuration Management and Acronym Sections of the Requirements Document. As a result of her participation in the E&V team, Marlene disseminates information obtained at the meetings and E&V products to her home organization, where she is involved in Ada technology work and SPO support, and to ESD/ALS. Some of the products distributed were compiler questionaires, documents, and benchmark tests.

2.2.11 Marlow Henne

Marlow Henne is a distinguished reviewer representing Harris Corporation. He is a member of the APSEWG. Marlow acts as an interface point between the NATO Working Group on Ada Environments for Guidance & Control, and he can assist the E&V team in collecting information on commercial APSEs. Marlow feels that his participation in the E&V team assists Harris's in-house APSE Evaluation Task.

2.2.12 Don Jennings

Don Jennings is from the Embedded Computer Systems Support Section of the Oklahoma City Air Logistics Center at Tinker AFB. Don is the chairperson of the COORDWG, and he is responsible for producing and distributing the E&V Status Report. At his home organization, Don is the Ada focal point.

2.2.13 Elizabeth S. Kean

Liz Kean is a representative from the Rome Air Development Center (RADC). She is a principal evaluator of the Air Force's Ada Integrated Environment (AIE) effort. As the chairperson of the APSEWG, Liz contributed the AIE and ALS write-ups to the APSE Analysis Document. She developed the Compiler Criteria Document. Through her participation in the E&V team Liz has available technology that she uses to evaluate the AIE Ada Compiler.

2.2.14 Randal Leavitt

Randal Leavitt is a distinguished reviewer representing a Canadian Company, Prior Data Sciences, Ltd. Randal is the chairperson of the Ada Working Group (AWG) for the Canadian Standards Association. He is a member of the COORDWG and is conducting a survey of current software evaluation methods.

2.2.15 Dr. Tim Lindquist

Dr. Tim Lindquist is a professor at the Virginia Polytechnic Institute Department of Computer Science. He is a member of the KIT/KITIA, and he is the chairperson of the REQWG. From his participation in the E&V team, Tim has acquired information on the following: research topics for students, software interface specifications, and an operational semantic definition of the CAIS. Tim emphasized the idea that the requirements defined by E&V are an excellent focus for research projects.

2.2.16 Patrick J. Maher

Pat Maher is from the F-16 Avionics Integration Support Facility at Hill AFB. He is a member of the COORDWG and is responsible for publishing the E&V meeting minutes. Pat also assisted in the publication of the Public Coordination Strategy (PCS) and Project Reference List (PRL) Documents. At his home organization, Pat is the Ada focal point.

2.2.17 Ronnie J. Martin

Ronnie Martin is a representative of the Georgia Institute of Technology School of Information and Computer Science. She is working on the Software Test and Evaluation Project (STEP) whose mission is to improve the practice of DoD software test and evaluation (DDT&E). While participating in the REQWG, Ronnie contributed to the Requirements Document and has served as a coordination point between the E&V team and the STEP/DDT&E activities. Ronnie also participated in the E&V Workshop.

2.2.18 Gary McKee

Gary McKee is a distinguished reviewer representing Martin Marietta. Gary is a member of the CAISWG. Gary's participation in the E&V team provides input to Martin Marietta's in-house Comprehensive Software Development Environment to aid the environment design. His participation also provides a source for the state-of-the-art E&V information relative to Ada technology to disseminate to Martin Marietta's Ada training program.

2.2.19 Mike Meirink

Mike Meirink is a distinguished reviewer representing the Sperry Corporation. Mike is a member of the REQWG. He led the subgroup that drafted the outline for the Tools and Aids Document and has made the Sperry tool taxonomy data available to the group. Due to his participation in the E&V team, Mike has gained insight into building and evaluating environments. The coordination between E&V and Sperry Corporation helps to sustain harmony with the DoD environment efforts and those efforts supported by Sperry.

2.2.20 Mike Mills

Mike Mills is a representative of the ECSPO at WPAFB. He currently is maintaining their Jovial compiler and is participating in the Ada/1750A compiler project. Mike feels that his experience with compilers is beneficial to his working group, the CAISWG. The information he has collected as a member of the E&V team has aided the development of the Ada/1750A compiler project at WPAFB.

2.2.21 Lt. Douglas M. Olson

Lt. Olson represents Hq AFCMD/SID from Kirtland AFB. At his home organization, Doug oversees, manages and administers contractor performance. As a member of the APSEWG, Doug has taken an active role in the second draft of the APSE Validation Procedures Document. Due to his participation in the E&V team, Doug feels that he brings increased Ada knowledge to his home organization.

2.2.22 Paul Reilly

Paul Reilly is a distinguished reviewer representing the Data General Corp. He is involved in ADE development and software support. Paul is a member of the APSEWG and feels that one of the major benefits of being a part of the E&V team is the access to the ARPANET and all the information that it holds.

2.2.23 Ray Sandborgh

Ray Sandborgh is a distinguished reviewer representing the Sperry Corporation. Ray is a member of the REQWG and has been working on the Reference Manual Guidebook. He is interested in providing contractor perspective on measurement theory, experimental design, clinical evaluation models, and software testing. As a benefit from his participation in the E&V team, Ray has learned about evaluation efforts which may help the development of the Sperry 1100 series compiler and has obtained information on the development status of the AIE, ALS, and the ALS/N.

2.2.24 Lt. Darleen Avery Sobota

Lt. Sobota is a representative of AFWAL at WPAFB. She is the chairperson of the CAISWG and has made significant contributions to her group's validation documents. Through her participation in the E&V team, Darleen keeps abreast of new technology and research advances. She is currently involved in the installation of an Ada compiler for her home organization.

2.2.25 Jimmy Williamson

Jimmy Williamson is a representative of AFWAL at WPAFB. Jimmy is the Vice-chairperson of the COORDWG and is responsible for maintaining an up-to-date list of technical coordination activities. Through his participation in the E&V team, Jimmy provides AFWAL with relevant information regarding the relationship of the E&V Task with other related efforts. Jimmy is a member of the Tri-Service ALS test team and is the Avionics Laboratory focal point for the ALS.

2.2.26 Betty Wills

Betty Wills represents the Command and Control Systems Office (CCSO) at Tinker AFB. Betty is a member of the COORDWG and was responsible for the reorganization of the Project Reference List. Betty feels that her participation in the E&V team allows her home organization to keep abreast of the changes in the Ada environment.

2.2.27 Doug Yarborough

Doug Yarborough is from GTE representing WIS. WIS is the WWMCCS Information System. WWMCCS is an acronym for World Wide Military Command and Control System. As a member of the APSEWG, Doug coordinates the activities of the GTE-sponsored APSE with the activities of the APSEWG.

2.3 General Discussion

2.3.1 ECLB Disk Space

Too many team members are running out of disk space. To save some of your messages, you can put them in a file and store the file on tape. In order to do this: (1) save the messages in a file, (2) execute the command ARCHIVE filename. Once a file is archived it will no longer appear in your directory. In order to recover the file from the tape, execute the command RETRIEVE filename. It takes overnight for a file to be retrieved. Since archived files do not appear in the directory, it is the team member's responsibility to remember the files he/she has archived.

2.3.2 Distinguished Reviewers

If you presented a position paper at the 1984 E&V Workshop, then you are considered a distinguished reviewer (DR). As a DR, you are invited to join the E&V team and to participate in its activities.

If you are a DR and you have left the organization that supported you at the 1984 E&V Workshop, but your new organization is willing to support you as a DR, then you can remain on the E&V team. You can stay on the Team for as long as your new company is willing to support you. The original company also has the right to continue to send a representative because they supported the 1984 E&V Workshop.

If you are a DR, but you do not take an active part in the Team and its activities then you will be dropped as a DR. Your name will be removed from the list of distinguished reviewers, and your ARPANET account will be deleted.

If you are presently a DR as a result of the 1984 E&V Workshop, you are not required to participate in the 1985 E&V Workshop. If you wish to submit a position paper, that is fine. If you choose not to participate, then you will still keep your DR status.

It should be understood that those people whose papers do get accepted for the 1985 E&V Workshop will be extended the opportunity to become a DR and will have all of the benefits and responsibilities that it involves.

2.3.3 March 1985 Meeting

If Jinny can find a host, the next E&V meeting will be held in California from 4-6 March. Otherwise, the meeting will default to March 6-8 at WPAFB in Dayton, Ohio.

The E&V meeting was then adjourned.

2.4 Action Items

The format for the action items is the following:
AI-date-number: name of person responsible, topic of AI, date due.
The date due will be left off if no date was provided.

AI-12-5-84-1: Jinny Castor. Complete revised E&V Plan. 31 Dec 84.

AI-12-5-84-2: Jinny Castor. Request MILNET accounts for Kathleen Gilroy and Debra Harto. 14 Dec 84.

AI-12-5-84-3: Jinny Castor. Request STARS list of point of contacts for area coordinating teams from Major C. Lillie. 14 Dec 84.

AI-12-7-84-1: Jinny Castor. Request information regarding the E&V CAISWG participation at the KIT/KITIA meeting in January 1985. 14 Dec 84.

AI-12-7-84-2: Jinny Castor. Establish meeting dates and location for the March 1985 E&V meeting.

AI-12-7-84-3: Jinny Castor. Provide copies of presentation slides to E&V members.

AI-12-7-84-4: Jinny Castor. Update the EV-TEAM-INFO file and the EV-TEAM-MAIL file.

AI-12-7-84-5: Jinny Castor. Send copies of the ALS & ALS/N documents to Bard Crawford, and a copy of the AIE document to Paul Riley.

AI-12-7-84-6: Don Jennings. Get information on the Asilomar Conference Center as a possible site for the 1986 E&V Workshop.

AI-12-7-84-7: Don Jennings. Investigate the requirements for the public release of the E&V Status Report to publications such as IEEE.

AI-12-7-84-8: Gary McKee. Send a NET message to Trish Oberndorf to discuss questions on CAIS 1.4.

AI-12-7-84-9: Gary McKee. Develop the General Node Management breakdown for the CAIS, section 5.1.

AI-12-7-84-10: Nelson Estes and Mike Mills. Develop the breakdown of the CAIS Process Nodes, section 5.2.

AI-12-7-84-11: Bud Hammons and Doug Olson. Develop the breakdown of the CAIS I/O Facilities, section 5.3.

AI-12-7-84-12: Darlene Sobota. Develop the breakdown of the CAIS utilities, section 5.4.

AI-12-7-84-13: Marlene Hazle. Map Configuration Management Section into a functional taxonomy. Jan 85.

AI-12-7-84-14: Rich Fleming. Map Command Language Interpreter Section into the functional taxonomy. Jan 85.

AI-12-7-84-15: Kathy Gilroy. Map the Compiler Section into the functional taxonomy. Jan 85.

AI-12-7-84-16: Ronnie Martin. Consolidate the ideas on Product quality guidance. 31 Dec 84.

AI-12-7-84-17: Ray Sandborgh. Generate a decision model for tool support. Feb 85.

AI-12-7-84-18: Mike Meirink. Develop a refined outline of the Tools/ Aids Document. Mar 85.

AI-12-7-84-19: REQWG. Consider Liz Kean's definitions and how they will fit into the functional taxonomy. Jan 85.

AI-12-7-84-20: REQWG. Consider outstanding comments. Mar 85.

ATTENDANCE LIST
E&V Team Meeting, 5-7 December, 1984

Michael Bridges
General Dynamics/DSD
San Diego, California

Jeff Brunson, Jr.
AFALC/PTEC
Wright-Patterson AFB OH

Jinny Castor
AFWAL/AAAF
Wright-Patterson AFB OH

Thomas Conrad
NUSC
Newport, Rhode Island

Bard Crawford
TASC
Reading, Massachusetts

Capt. Albert Deese, Jr.
ASK/ADOL
Wright-Patterson AFB OH

Richard Fleming
Space Division/ALR
c/o The Aerospace Corporation
Los Angeles, California

Kathleen A. Gilroy
Software Productivitiy
Solutions, Inc.
Melbourne, Florida

Daniel Green
NSWC
Dahlgren, Virginia

Charles Hammons
Texas Instrumments
North Texas State University
McKinney, Texas

W. W. Happ
SM-ALC/MMEHP
McClellan AFB CA

Debra Harto
AFATL/DLCM
Eglin AFB FL

Marlene Hazle
MITRE Corporation
Bedford MA

Marlow Henne
Harris Corporation
Melbourne FL

Don Jennings
OC-ALC/MMECE
Tinker AFB OK

Elizabeth Kean
RADC/COES
Griffiss AFB NY

Randal Leavitt
PRIOR Data Sci/P.N.D. Canada
39 Highway 7
Nepean, Ontario
K2H 8R6

Maj. Charles Lillie
HQ AFSC/ALR
Andrews AFB MD

Tim Lindquist
Virginia Polytechnic Institute
Blacksburg, Virginia

Patrick Maher
OO-ALC/MMECF
Hill AFB UT

Ronnie J. Martin
Georgia Institute of Technology
Atlanta, Georgia

Gary McKee
Martin Marietta
Denver, Colorado

Mike Meirink
Sperry Corporation
St. Paul, Minnesota

Lt. Douglas M. Olson
HQ AFCMD/SID
Kirtland AFB NM

Ray Sandborg
Sperry Corporation
St. Paul, Minnesota

Lt. Darleen Avery Sobota
AFWAL/FIGR
Wright-Patterson AFB OH

Jimmy Williamson
AFWAL/AAAF-2
Wright-Patterson AFB OH

Doug Yarborough
GTE Government Systems
Billerica, Massachusetts

Mike Mills
ASD-AFALC/AXTS
Wright-Patterson AFB OH

Paul Reilly
Data General Corporation
4400 Computer Drive
Westboro, Massachusetts

Jane Shirley
Systran Corporation
Dayton, OH

Lori Walton
Systran Corporation
Dayton, OH

Betty Wills
CCSO/SKXD
Tinker AFB OK

APPENDIX D

MINUTES

of the

EVALUATION & VALIDATION (E&V) MEETING

4-7 March 1985

TABLE OF CONTENTS

3.2.4 Future Plans

3.3 General Discussion

3.3.1 Stars Representatives From The E&V Team

3.3.2 Policy For Working Group Chairpersons

3.3.3 Requests For A Copy Of The ALS

3.3.4 E&V Glossary

3.4 Action Items From E&V Meeting 5-7 December 84 Not Accomplished

3.5 Action Items From E&V Meeting 4-7 March 1985

1.1 Welcome

The E&V meeting began with a welcome by the chairperson, Jinny Castor. Before the introduction of Mr. W. M. Murray, each Team member introduced himself.

Mr. W. M. Murray, director of technical software for Data Systems Division of General Dynamics (GD), welcomed the E&V Team to GD's San Diego site. He stressed that GD is committed to Ada development. GD is active in many Ada projects including the following: the use of Ada in mission critical systems, the development of an Ada/1750 compiler system, and the amount of training required by systems and software engineers to become proficient in Ada.

1.2 General Business/Action Items

1.2.1 Evaluation of APSEs

Jinny emphasized that it appears to other DoD organizations that the E&V Team is investigating only whole APSEs and not the individual components that make up an APSE. She encouraged the Team to make it understood that the E&V Team is analyzing each component of an APSE individually as well as evaluating an APSE as an entire entity.

1.2.2 E&V Workshop

The E&V Workshop will be 8-12 July 1985 at Airlie, Virginia. Unfortunately, this is the same week as the KIT/KITIA meeting. The main thrust of the workshop is to enumerate the evaluation criteria of APSEs associated with a particular application software development. Any interested industry representative is invited to submit a position paper in response to the Workshop's Commerce Business Daily announcement.

1.2.3 Action Items From December Meeting

The action items (AI) generated at the December 1984 E&V meeting were reviewed. The AI that were not completed will be included in the minutes of the next meeting and will be carried forward until they are fulfilled. Approximately 7 out of 20 AI from the December 1984 meeting were not completed.

1.3 Real-Time Programming With Ada

Mr. John DaGraca from General Dynamics presented a briefing on the current status of Real-Time Programming with Ada. Mr. DaGraca presented a description of some Ada compiler implementation-dependent features of the run-time support environment in order to support Ada tasking. The topics that outline Mr. DaGraca's presentation are the following: (1) objective, (2) run-time executive, and (3) conclusions.

1.3.1 Objective

The objective of this GD program was to apply software engineering principles to obtain an efficient system with respect to timing and memory utilization. The project was constructed in such a way that one team of people would design the system, then another team would maintain it. This allowed for an evaluation of the maintainability of the Ada code produced by the design team.

They designed and implemented a real-time executive whose basic function was to serve as a process scheduler. Telesoft, Rolm, NYU and DEC compilers were used to obtain comparison data for run-time systems.

It was determined that a strong development methodology is essential in order for the maintenance personnel to have a model with which to work. A second conclusion is that thorough understanding of the run-time support environment is crucial.

1.3.2 Run-Time Executive

The team used Ada tasking features to implement the run-time executive. They found that it is important to have a thorough understanding of the run-time support environment and the run-time implementation-dependent features.

1.3.2.1 Run-Time Support Environment

The old method used to design a run-time executive was to design an interrupt-driven scheduler. It was written in assembly code and was driven by each process's priority. This design method allowed the programmer to decide when an interrupt would occur, how it would be handled, and what priority each process would be assigned. The programmer knew that any process would be in one of the following three states: active, suspended, or queued.

The new method used to design a run-time executive is to use Ada tasking. Ada's tasking feature is difficult to understand because one can use a task to implement a pure process, a monitor, a buffer, a semaphore, a message, or an interface unit. The run-time support environment decides when an interrupt will occur, how it will be handled, and what priority each process will be assigned. It is important that the programmer understands how the run-time support environment is implemented because the tasking results depend on the programmer's interpretation of this environment.

1.3.2.1.1 Results

It was determined that some Ada compilers implemented tasking using time slicing and some used run/until blocks. When the Ada tasking program was re-hosted on other machines, these various run-time implementations caused the program to obtain different results.

It was suggested that since the Ada Compiler Validation Capability (ACVC) cannot determine how run-time systems are implemented, a standard run-time model for compilers should be developed. However, when a run-time system is implemented, it should be thoroughly documented especially the implementation-dependent features of tasking.

1.3.2.2 Implementation-Dependent Features

The two implementation-dependent features that gave the team the most trouble were order of elaboration, and synchronized entry points.

1.3.2.2.1 Order of Elaboration

The order that data is initialized in Ada tasking is not specified. It is important for a programmer to know when and in what order data is initialized.

The Language Reference Manual (LRM) does not specify in what order one can assume that tasks will be initiated. Yet, it is important for a programmer to be able to predict in what order tasks will execute. It is important because when something is designed, the programmer wants operations to occur in the order that is expected and not in an arbitrary manner. This was a problem for the people programming the run-time executive.

1.3.2.2.2 Synchronized Entry Points

It is difficult to program tasks when rendezvous are performed at unspecified times during execution. In one case where a critical section was required, one compiler implemented it using a signal/wait scheme, and the other compiler used tasking. These variations in implementations cause a programmer problems when he is concerned with timing and memory constraints.

1.3.3 Conclusions

It was concluded that run-time specifications and implementations need to be standardized. Without standards, portable software can be generated, but memory and timing faults will be a problem.

A second conclusion is that there are constructs which are not fully explained in the LRM and are sometinmes left as implementation-dependent features. Programmers who are unaware of these implementation-dependent features could encounter numerous problems with real-time programming with Ada.

A third conclusion is that a strong methodology is required when programming in Ada because it provides the people tasked to maintain the software with a path to trace back to the design level.

1.4 WIS Software Development And Maintenance Environment

Captain Percy Saunders from Hanscom AFB, Massachusetts discussed the development of the all-Services software development and maintenance environment (SDME). SDME is a program targeted to updating the computer programming capabilities using the Ada language for the World Wide Military Command and Control System (WWMCCS) Information System, or WIS. Captain Saunders' presentation was divided into the following sections: (1) overview of SDME, (2) requirements for SDME, (3) history of SDME, (4) SDME program, (5) architecture, (6) user view, and (7) summary.

1.4.1 Overview Of SDME

The SDME is a software system resident on standard WIS hardware such as the WIS workstation, Common User Processor, and Joint Mission Hardware. Its purpose is to enhance the effectiveness of WIS personnel whose mission is the development and maintenance of WIS systems. The SDME accomplishes this through an evolvable set of tools, embedded in a portable environment, with a common user interface to all SDME functionality.

1.4.2 Requirements For SDME

SDME aims to provide powerful Ada programming tools to WIS sites for support of software development activities. It is designed to manage programs from 20 to 20 million lines of code and to respond to the changing conditions of a software development process. SDME is also responsible for providing the individual user with access to a WIS workstation in order for him to develop software without degrading the entire system's performance.

1.4.2.1 Design Requirements

SDME will be designed and implemented in Ada. It will be portable over a variety of hardware and will be capable of operating in a distributed environment. SDME provides evolutionary growth capabilities and supports reusable software components.

1.4.3 History Of SDME

1.4.3.1 WIS Precursors

WIS is based on the Distributed Software Engineering and Control Process (DCP). DCP demonstrated the capabiltiy to develop software in a distributed environment and the use of an Ada command language interpreter. This effort was active from June 1983 to December 1984.

1.4.3.2 NOSC Tools

The WIS Joint Program Management Office (JPMO) authorized the Naval Ocean Systems Center (NOSC) to contract to approximately 25 vendors to require existing software tools in Ada. These tools will provide the foundation of the SDME; they include reusable software packages such as mathematical functions and larger individual tools such as editors and data base managers. These tools were delivered in the first quarter of FY85 and are currently being beta-tested by GTE, the support contractor for the WIS SDME effort.

1.4.3.3 Functional Model

A functional model description that provides an architectural basis for the SDME was delivered in January 1985. This document identifies the content of the major components of the SDME, especially the data base.

1.4.3.4 Compiler Guidelines

The WIS JPMO has received a document that details WIS's need for Ada compilers. The document covers the entire compilation system from the compiler to the run-time support environment. This document will be used to communicate to industry WIS's unique needs for Ada compilers and associated support tools.

1.4.3.5 Strategies

WIS intends to develop an evolvable, portable system that meets the needs of the WIS user community. This will be accomplished by integrating WIS precursor efforts into the mainline WIS program and by incorporating feasible environmental and Ada technological advances of industry and government. A dialogue with WIS site personnel will be established in order to inform them of SDME and to integrate their feedback into the SDME design.

1.4.4 SDME Program

The SDME program includes the following task areas: (1) procured component evaluation, (2) proof of concept and prototyping, (3) SDME design specification, (4) design, development, and integration, (5) test and evaluation, (6) installation, (7) maintenance, (8) compiler acquisition, and (9) site integration.

1.4.4.1 Procured Component Evaluation

This area is managed by NOSC, and its aim is to analyze industry and government Ada tools. These tools will be evaluated as to each tool's usefulness in the SDME.

1.4.4.2 Proof Of Concept And Prototyping

The purpose of this task is to research and prototype the outstanding design issues for the SDME. Areas of special interest are Ada design, data base design, user interfaces, and requirements tracking. A document, the Complete Conceptual Manual (CCM), will be a result of this phase of the program.

1.4.4.3 SDME Design Specification

This task area will specify the required functionality of tools, tool interfaces, and the user interface.

1.4.4.4 Design, Development, And Integration

This area will involve the design of the environment's systemic components such as the interfaces and data base objects, the development necessary to integrate all specific components, and the interaction with the WIS user community in order to evaluate and tune the design.

1.4.4.5 Test And Evaluation

This task will include rigorously stressing the system in order to uncover errors prior to deployment. This testing will take place at GTE's software development installation.

1.4.4.6 Installation

The purpose of this task is to install SDME in WIS operational support facilities and to conduct acceptance testing. This task also includes the training of WIS site personnel.

1.4.4.7 Maintenance

This process will involve on-site support for the SDME, including error correction and interim enhancements. A staff from GTE will be provided at the site to handle maintenance issues.

1.4.4.8 Compiler Acquisition

This area includes development of compiler benchmark programs for performance and functionality testing. An evaluation of compilers will be conducted, and appropriate compilers for use in SDME will be recommended to the government.

1.4.4.9 Site Integration

The purpose of this task area is to insure the utility of SDME to site personnel. It involves hands-on training, consolidating and evaluating user feedback, and providing input to t'e SDME design group.

1.4.5 Architecture

The architecture of the SDME is based on layered levels of increasing abstraction which provide portability, evolvability, and maintainability. The main components of the SDME are the following: (1) SDME core, (2) Portable APSE Interface Set (PAIS), (3) object library, (4) tool interface, and (5) user interface.

1.4.5.1 SDME Core

The SDME core is machine dependent and includes the operating system, network interface, and relational data base management system.

1.4.5.2 PAIS

The PAIS insulates the rest of the SDME from the core. It is compatable with the Common APSE Interface Set (CAIS), and other interface standards such as the graphic kernel system. The PAIS is designed to permit Ada package specifications to remain stable as package bodies are reimplemented for rehosting.

1.4.5.3 Object Library

The object library is designed as a user view of objects. The library includes an encyclopedia that contains object information, libraries that contain collections of objects, and a director that handles the logical to physical distributed mapping.

1.4.5.4 Tool Interface

The tool interface serves to isolate each tool from the SDME data organization. This interface permits different implementations of similar tools while preserving a stable user interface.

1.4.5.5 User Interface

The user interface is separated from the SDME core by the PAIS and is separated from the tools by the tool interface. The user interface provides a stable boundary when the SDME is rehosted, the core component changes, or the tools change. The user interface implements the full Ada command language with menus, graphic systems and windows.

1.4.6 User View

The user interface permits access to SDME functions and controlled objects which reflect the user's work. This provides the user with only the information that is relevant to his program.

The user view of the system is adaptable to different project phases and work domains. It provides support for joint software maintenance and site-unique software development. The user view of the SDME is consistent from host-to-host, and site-to-site; it is designed in such a way as to remain constant while remaining flexible enough to incorporate new tool technology in the future.

1.4.7 Summary

The SDME focuses industry technology on WIS support needs and was designed to be evolvable through use of externally developed components. Although WIS is explicitely oriented toward a distributed network environment, requirements of the WIS user community are an important design issue. A significant design factor of the SDME is that it was designed to be able to respond to a changing environment, and to evolve as technology advances.

1.5 Open Discussion

1.5.1 E&V Information

Jinny Castor reminded the E&V Team that information made available to the Team is primarily considered "for the E&V Team only", unless explicitely stated otherwise. It should be reiterated that information sent to the Team should be kept to the Team and not distributed further.

1.5.2 Deliverables

A schedule of deliverables due for the remainder of FY85 was reviewed. Each working group has deliverables due sometime during the year and should keep working toward completing these documents.

The general session of the E&V meeting was adjourned so that working groups could meet separately. Working groups met separately for the remainder of the day.

2.0 Tuesday, 5 March 1985

2.1 Interface Standardization

Ms. Tricia Oberndorf, chairperson of the Kernel Ada Programming Support Environment (KAPSE) Interface Team (KIT)/KAPSE Interface Team from Industry and Academia (KITIA) presented a briefing on KAPSE interface standardization. The objective of the Common Ada Interface Set (CAIS) is to provide a standard interface to a KAPSE, and the KIT/KITIA is responsible for the development of the CAIS. The topics Ms. Oberndorf outlined for her presentation are the following: (1) describe the background for the CAIS, (2) discuss the KIT/KITIA goal, (3) discuss the KIT/KITIA progress and plans, (4) list work products, and (5) discuss CAIS prototyping.

2.1.1 Background

The two basic motivations for common interfaces at the KAPSE level are the following: (1) to reduce the cost of maintenance by having fewer tools to accept and use, and (2) to reduce the cost of development by having a common set of tools which everyone can use. Basically, we want to be able to share tools and databases between APSEs.

2.1.2 KIT/KITIA Goal

The goal of the KIT/KITIA is to develop a set of common interfaces which will form a boundary between the tailored implementation of the underlying host computer and the tools developed to be common to APSEs. This set of common interfaces is the CAIS, and it is planned that the CAIS will become the interface level of the KAPSE itself and not an extra layer on top of the KAPSE.

Since tools will be written to satisfy the common interface (the CAIS) and not to satisfy the host computer's implementation dependencies, the development of the CAIS will encourage a sharing of resources throughout the industry/government community. This sharing could take place through channels such as DoD-industry information exchanges, IEEE or ACM conferences, and project interchanges. The sharing of tools will result in CAIS-compatible tool libraries. Since this software can be reused, government projects will cost less than if this software had to be rewritten.

2.1.3 KIT/KITIA Progress and Plans

2.1.3.1 CAIS Documents and Comment Reviews

The KIT/KITIA has proposed a KAPSE interface set; this set is documented in CAIS Document versions 1.2, 1.3, and 1.4. The draft CAIS versions 1.3 and 1.4 were distributed to over 500 reviewers in the United States and Europe. Two CAIS public reviews were conducted, one in August 1984, and the second in November 1984. The KIT/KITIA continues to review comments from these activities, and answers will appear soon on the MILNET.

2.1.3.2 ALS/CAIS Comparison Study

An ALS/CAIS comparison study was conducted, and a report containing the conclusions of this investigation is available. The study considered the work required to transition the ALS to the CAIS and dealt with the topics of how one would approach such a task, and what are the similarities and differences between the ALS and the CAIS.

2.1.3.3 Interoperability and Transportability

Interoperability (I) is the ability to share databases; transportability (T) is the ability to move tools. The KIT/KITIA recently reached concurrence on the requirements and design criteria for interface sets which achieve I&T. This set of requirements will be imposed on the CAIS Version 2.0 contractor who will design a set of interfaces which meet this set of requirements and design criteria, and which are upward compatible with CAIS Version 1.0. The CAIS Version 2.0 contract will be awarded in May 1985.

2.1.3.4 MIL-STD-CAIS

The KIT/KITIA delivered a MIL-STD-CAIS to the AJPO. The MIL-STD-CAIS will be distributed for review to the three Services, to other DoD organizations, and to major industry associations. A public review of the comments on MIL-STD-CAIS will be conducted, and MIL-STD-CAIS will be revised to reflect these comments.

2.1.4 Work Products

2.1.4.1 Requirements and Design Criteria (RAC)

The RAC were designed in the context of STONEMAN and with the idea that it would be implemented in approximately five years. The RAC Document is directed toward any interface set which will achieve transportability of tools and interoperability of databases. The five main sections of the RAC Document are the following: (1) General Design Objectives, (2) General Syntax and Semantics, (3) Entity Management, (4) Program Management Facilities, and (5) Input and Output.

2.1.4.2 Common APSE Interface Set

2.1.4.2.1 CAIS Document

Founded on the ALS and AIE, the CAIS was written for life-cycle environments for the support of mission critical computer systems. It was designed for Ada support and was not designed for use on target machines. The CAIS document describes the interfaces of the CAIS; the document's main section provides the interfaces described as Ada package specifications. For each interface the following details are addressed: purpose, parameters, exception handling, and additional interfaces. The document also includes four appendices - predefined entities in the CAIS, comp'lable package specifications, compilable package bodies, and an overview of the interfaces by package and function.

2.1.4.2.2 CAIS Utilities

The CAIS standardizes those aspects of writing tools which commonly cause problems when moving tools between hosts. The CAIS provides the following: (1) naming conventions for users, files, processes, and devices, (2) a hierarchical structure to retain information about processes and files, (3) interfaces to start processes and tools, (4) capability to create background processes, (5) support for creating, deleting, opening, closing, reading, and writing files, and (6) support for 3 kinds of terminals and support for magnetic tape drives. Some topics deferred until the initial CAIS is implemented are the following: (1) configuration management, (2) explicit controls for distributed environments, and (3) inter-tool interfaces.

2.1.4.2.3 Important Observations

Achievements noted during the development of the CAIS are: (1) the CAIS remained as true to Ada as possible. In many instances this included trying to cope with decisions which Ada did not make because of the difficulties of making them in a transportable way; (2) the CAIS was designed to operate with a wide range of existing operating systems; (3) the CAIS had to accommodate new ideas about what is required to support a software engineering environment; and (4) the CAIS was developed in an extremely public forum, and therefore represents a commitment to achieve concensus from the Ada community.

2.1.5 CAIS Prototype

2.1.5.1 Prototype Definition

A CAIS prototype will be developed to prove whether the CAIS design is implementable. The prototype will include a combination of experimental implementations, experiments designed to produce information using those implementations (including criteria for measurement), and reports evaluating the performance of the implementations with respect to the criteria. Besides justifying the CAIS design, the CAIS prototype will provide some insight as to whether useful tools can be implemented using the CAIS, and whether the CAIS will achieve its transportability objective.

2.1.5.2 Kinds of Prototypes

The four basic kinds of prototype implementations are the following: "quick and dirty", tuned, full or partial implementation of individual tools, and full implementation of toolsets. A "quick and dirty" implementation is designed to be completed as quickly as possible. This prototype is useful for proving implementability and provides a base for tool experiments. A tuned implementation is designed to be as efficient as possible; it provides useful insights into tailorability and the overall tool potential for efficiency. A full or partial implementation of individual tools is useful for insights into tool usability and applicability. The last type of prototype, a full implementation of toolsets, is designed to provide a full APSE content for experiments. Full implementations are also useful for determining the ability of the CAIS to satisfy the CAIS design requirements.

2.1.5.3 Identified Prototypes

There are nine prototypes currently in development. TRW is developing a full CAIS Version 1.0 implementation under a KIT-support contract. TRW's implementation is based on UNIX/ARCTURUS hosted on a VAX. The MITRE effort is currently in the design phase. The only totally in-house effort to prototype a CAIS is sponsored by Gould. Other CAIS prototype work is being conducted by Texas Instruments, Virginia Polytechnic Institute, a government-sponsored WIS project, the RADC-sponsored AIE effort, a STARS-sponsored Los Alamos project, and a VHSIC- and AJPO-sponsored very high order development language (VHDL) program.

2.2 The TRW Software Productivity System

Mr. Imad Bitar from TRW's Redondo Beach, California facility gave a presentation on TRW's Software Productivity System (SPS). The SPS has been an ongoing project at TRW for four years, and its aim is to increase the productivity of software engineers. The topics that outline Mr. Bitar's presentation are the following: (1) background, () the SPS, (3) lessons learned, (4) measurement data, and (5) conclusions.

2.2.1 Background

2.2.1.1 Productivity Assessment

In 1980, TRW conducted a study to determine how the company could increase its software productivity. The study involved an internal assessment of the industry's shift to better software productivity.

2.2.1.1.1 Internal Assessment

TRW's internal assessment consisted of a questionnaire that inquired "If there were only two or three things you could get TRW to do to improve software productivity, what would they be?" The four areas of improvement cited most often were management actions, work environment and compensation, education, and software tools. An interesting result of the survey was that while most upper and middle management personnel felt that improved management actions would improve productivity, the software engineers and programmers emphasized the need for more software tools.

2.2.1.1.2 Quantitative Assessment

TRW studied some companies to determine what actions they were taking to improve software productivity. Two of the companies TRW investigated were IBM and Bell Labs. TRW received input from some educational institutions such as Harvard and Carnegie Mellon, and they received ideas from Dr. Barry Boehm's book "Software Engineering Economics." One of the results of these studies was that if one had nominal software tools in a project and only the software tools were improved, then a 150 percent increase in productivity could be expected. Another result was that if one improved the quality of personnel on a software project then a 400 percent increase in productivity could be expected.

2.2.1.3 TRW Productivity Goals

TRW's near-term goal (1985) is to increase the 1980 average productivity by a factor of two; TRW's long-term goal (1990) is to increase the 1980 average productivity by a factor of four. These goals imply that in 1985 (1990) TRW software projects will bid 1/2 (1/4) the number of labor months in the 1980 time frame for the development of similar applications software while maintaining the required quality.

2.2.1.4 Chronology

The SPS was established in 1981, and its goal was to build a software development environment (SDE).

The initial facility was system operational in 1982. It supported one project, had nominal tool integration, and the software tools were basically those provided with UNIX.

In 1983, the first SPS VAX was installed, and the number of users was expanded to 190. TRW found that the users requested more documentation and management aids than were currently available on the system.

In 1984, two VAXs and two Pyramid computers were added to the system. The system supported 400 users, and the user support program was expanded by implementing additional software tools.

The goals for 1985 are to add five more computers and to increase the number of users to 1000.

2.2.2 The SPS

2.2.2.1 Operational Concept

The operational environment consists of source computers, to support the development environment, and target computers, to host the final product. The system is connected by a LAN, and any source computer can be reached from any terminal/workstation.

2.2.2.2 Objectives

TRW wanted to automate all of the activities of a project life cycle. These activities include requirements and design, coding, testing, and document preparation. TRW planned to implement an environment that would automate one or more methodology. The result of not forcing all projects to use a single methodology is that software methodologies will support technical methods and management procedures. TRW's environment was designed to support all users, from managers and programmers to secretaries. The company implemented a method of storing and relating data so that all data pertinent to a project could be accessed.

2.2.2.3 SPS Software

Some design tools available on the SPS are a program design language, and a requirements traceability tool. Development software includes compilers for Ada, Fortran, C, and Pascal, debuggers, and an automated unit development folder. Management tools include an electronic spread sheet containing functions for cost-to-completion and proposal pricing, and a capability for milestone charts. SPS software also includes automated office tools, user interface utilities, general purpose utilities, and software to accomplish file transfers.

2.2.2.4 SPS Hardware

The SPS LAN contains 9 central processing units (CPU's), 262 terminals (with 132 more on order), and 160 bus interface units. The SPS also includes two IDM 500 Data Base Machines, 6 Imagen laser printers, and numerous portable terminals.

2.2.3 Lessons Learned

2.2.3.1 Software

TRW found that selecting UNIX as the basis for SPS was an excellent choice. UNIX is available on a variety of different computers, and this proved beneficial because TRW is not tied to a single manufacturer. TRW discovered that one should exercise caution not to obtain a hybrid version of UNIX because porting problems will result.

TRW learned that evolutionary SPS development has been beneficial. User feedback and tool utilization measurement data provided useful input to the project as it has developed over the past four years.

TRW found that the core of an automated SDE is an integrated project master data base. This data base avoids retaining redundant data, promotes data flow between tools, and provides a consistent means of storing data.

2.2.3.2 Hardware

TRW found that LANs are very beneficial; they are reliable and cost effective. Through the LAN, performance has been measured as high as 19.2K baud.

TRW concluded that a centralized environment is not an ultimate solution to user support problems. It was discovered that 20 percent of the users consume 70 percent of CPU resources. TRW plans to off-load the "heavy" resource users to personal workstations in order to make the system more cost effective.

Another interesting conclusion is that user acceptance of the SPS was enhanced through access to modern technology, such as laser printers, graphics facilities, and data base machines. It was noted that some users learned the system faster because of their enthusiasm to use these high technology devices.

2.2.3.3 Technology Transfer

TRW learned that user support functions were consuming 50 percent of SPS resources. In order to provide better user training and consulting, TRW created a User Support Organization. This organization is responsible for a reference library, and hardware maintenance support, as well as user consulting and training.

2.2.3.4 Measurement

Measurement data was collected on tool utilization, system resources utilization, and user/system interfaces. Whenever possible, subjective measurement was validated with real data.

2.2.3.4.1 Tool Utilization

The measurement data indicated that twice as much CPU time was spent for documentation services as was spent for software development. Documentation services include preparing viewgraphs and presentations, and updating requirements and design documents.

2.2.3.4.2 System Resources Utilization

Tools were analyzed based on the frequency of invocation. This data is useful for determining which tools should be moved to workstations in order to remove the load from the system. The ten tools that most heavily tasked the CPU are the following: (1) vi - screen editor, (2) csh - command interpreter, (3) ipr - formatter for printer, (4) ips - printer status monitoring, (5) troff - text formatter, (6) emacs - screen editor, (7) scribe - text formatter, (8) query - forms manager, (9) viewcomp - electronic spreadsheet, and (10) ada pdl - program design language.

2.2.3.4.3 User Function Data

Data was collected to determine which group of company personnel used the system most often. It was concluded that the software engineers and programmers occupied 65 percent of the CPU for development, and secretaries used 11 percent of the CPU for data entry. The personnel who used the system the least were managers and senior staff. Personnel who were surveyed felt that their productivity increased almost 40 percent due to the SPS.

2.2.4 Conclusions

TRW found that software development environments should be extensible, uniform, and customizable. The environment should support friendly interactive facilities, rapid prototyping, and reuse of internal components. It is also important to include a data base for use as a central information repository.

TRW believes that a software development environment can increase the productivity of project personnel, but a SDE is a large system and requires corporate commitments. Mature support tools are very important, and the man-machine interface must accommodate all classes of users and must be consistent across all tools. The final point that Mr. Bitar presented was that user involvement and acceptance of SPS was crucial to the success of the SDE.

The general session of the E&V meeting was adjourned so that working groups could meet separately. Working groups met separately for the remainder of the day.

3.0 Wednesday, 6 March 1985

3.1 Working Group Status Reports

3.1.1 APSEWG Status Report

The APSEWG status report was presented by Liz Kean, the chairperson. One personnel change was noted: Lt. Jim Kirkpatrick, Gina Burt's replacement to the E&V Team, moved from the APSEWG to the CAISWG. No deliverables were due this quarter. Accomplishments this quarter include establishing the format to describe the functions of the ALS, AIE, and ALS/N; the tool breakdown of these environments has begun. Projected work for next quarter includes listing the tools in the ALS, AIE, and ALS/N, and providing inputs, processing, and outputs for each tool. This work is an effort to map the environments into the SEE taxonomy. No deliverables are due next quarter.

3.1.2 CAISWG Status Report

The CAISWG status report was presented by the chairperson Darleen Sobota. Personnel changes noted include a new member, Lt. Jim Kirkpatrick, who will be transitioned into the position of CAISWG chairperson. The new vice-chairperson is Gary McKee, and the transitioning chairperson is Lt. Doug Olson. Having Trisha Oberndorf attend the CAISWG working group sessions helped them resolve many CAIS-related questions that the CAISWG had generated. The deliverables due this quarter is the APSE Validation Procedures Document. Accomplishments this quarter include incorporating the comments on the APSE Validation Procedures Document, reviewing the draft CAIS-MIL-STD version 1.4, and providing input to the CAIS working group of the KIT/KITIA. Projected work for next quarter is to continue the development for the breakdown of sections 5.1, 5.2, 5.3, and 5.4 of the CAIS document. No deliverables are due next quarter.

3.1.3 COORDWG Status Report

The COORDWG status report was presented by the chairperson Don Jennings. One personnel change was noted; Randal Leavitt moved from the COORDWG to the REQWG. No deliverables were due this quarter. Accomplishments this quarter include the E&V meeting minutes and status report, a draft Public Coordination Strategy Document version 2.0, and proposed changes to the Technical Coordination Strategy Document. A format for all E&V documents was prepared at the meeting and will be reviewed by the Team. Projected work for next quarter includes Public Coordination Strategy Document version 2.0, the E&V meeting minutes and status report, and the draft Technical Coordination Strategy Document version 2.0. The Public Coordination Strategy Document version 2.0 is the COORDWG deliverable for next quarter.

3.1.4 REQWG Status Report

The REQWG status report was presented by the chairperson Tim Lindquist. Personnel changes noted were the following: (1) Pat Lawlis will become REQWG chairperson, and (2) Marlene Hazle will fill the position of vice-chairperson. No deliverables were due this quarter. Accomplishments this quarter include the move to the SEE taxonomy, the outline for the Tools and Aids Requirements Document, and resolving the comments on the Requirements Document version 1.0. Deliverables due next quarter are the draft Requirements Document version 2.0, the draft Tools and Aids Requirements Document version 1.0.

3.2 Life Cycle Software Engineering Environment Taxonomy

Liz Kean from Rome Air Development Center (RADC) gave a presentation on the Life Cycle Software Engineering Environment (SEE) taxonomy. Liz covered the following topics: (1) background, (2) NBS/ICST taxonomy, (3) SEE taxonomy, and (4) future plans.

3.2.1 Background

The E&V Team and the Software Technology for Adaptable, Reliable Software (STARS) Joint Service Software Engineering Environment (JSSEE) task needed a SEE taxonomy. Analysis of the National Bureau of Standards Institute for Computer Sciences and Technology (NBS/ICST) taxonomy determined that a major fault of the NBS/ICST taxonomy is that it does not cover all of the tools required for a life cycle; it includes tools for the coding and unit test phases only. It was decided that the E&V Team and the JSSEE task needed to develop a generic taxonomy of functions for a SEE that included all phases of the life cycle. Three people from RADC, Richard Evans, Elizabeth Kean, and Frank LaMonica, were tasked to develop this taxonomy.

3.2.2 NBS/ICST Taxonomy

The NBS/ICST taxonomy is a basis for the SEE taxonomy. The NBS/ICST taxonomy is a hierarchical arrangement of software tool features; the taxonomy covers the basic processes of a tool: input, function, and output. There are three categories of tool functions; they are the following: transformation, management, and analysis. A transformation tool is defined to be any tool whose input is in a different form than its output. An example of a transformation tool is a compiler; the input is source code, and the output is object code. A management tool is identified by the control of data; a data base is a management tool. Analysis tools are categorized as static or dynamic. An example of a static analysis tool is a consistency checker which is a tool that determines whether or not an entity is internally consistent in the sense that it contains uniform notation and terminology. A dynamic analysis tool i. an assertion checker which is a tool that tests the validity of assertions as the program is executing.

3.2.3 SEE Taxonomy

The SEE taxonomy is a functionally-based, expanded version of the NBBS/ICST taxonomy. Each of the tool categories are subdivided into the following sections: (1) global, (2) system/software requirement, (3) preliminary design, (4) detailed design, (5) code/unit testing, (6) software integrati . ı testing, (7) software performance testing, and (8) post-deployment support. This breakdown is based on the NBS/ICST and on MIL-STD-SDS, and each of the eight subsections is further divided into tool features.

3.2.4 Future Plans

Minimal feedback on the SEE taxonomy has been received by the SEE design team at RADC; all constructive comments are appreciated. Future work planned for the taxonomy is to condense as much of the repetition as is possible. RADC plans to publish the SEE taxonomy as a technical report in the near future.

3.3 General Discussion

3.3.1 Stars Representatives From The E&V Team

Any E&V Team member interested in being the E&V representative to a STARS task area should contact Jinny Castor. Distinguished reviewers will be allowed to participate if their support organization agrees to pay travel expenses, and if the STARS working group agrees to permit an industry representative to attend its sessions. Government personnel who are interested in representing the E&V Team will have their expenses paid by the E&V Team budget. The E&V Team is most interested in sending representatives to the STARS Metrics and Measurement, and the STARS Applications task areas. The main responsibility of the E&V representative is to serve as a liason between the STARS task area and the E&V Team; each representative will report issues that impact the E&V Team and issues that the E&V Team could impact for his particular task area.

3.3.2 Policy For Working Group Chairpersons

Distinguished reviewers may hold the vice-chairperson position in working groups; this position was previously reserved for government personnel only. Each working group chairperson is still required to be a government representative.

3.3.3 Requests For A Copy Of The Als

Any government E&V Team member who wants a copy of the ALS must submit a written request to Jinny Castor. The letter must state the following points: (1) the person requesting the ALS is an E&V Team member, (2) the ALS will be used in support of E&V activities only, and (3) the ALS will not be released to a third party. This letter must be accompanied by three blank magnetic tapes which will be used to mail the software.

3.3.4 E&V Glossary

A need has arisen for an E&V glossary of terms. The COORDWG will adopt the KIT/KITIA glossary and add to it terms that are related to the E&V task. This glossary will be updated and maintained by the COORDWG.

The E&V Team meeting was then adjourned.

3.4 Action Items From E&V Meeting 5-7 December 84 Not Accomplished

AI-12-7-84-1:	Don Jennings. Get information on the Asilomar Conference Center as a possible site for the 1986 E&V Workshop.
AI-12-7-84-2:	Don Jennings. Investigate the requirements for the public release of the E&V Status Report to publications such as IEEE.
AI-12-7-84-3:	Gary McKee. Develop the General Node Management breakdown for the CAIS, section 5.1.
AI-12-7-84-4:	Nelson Estes and Jim Kirkpatrick. Develop the breakdown of the CAIS Process Nodes, section 5. 2.
AI-12-7-84-5:	Bud Hammons and Doug Olson. Develop the breakdown of the CAIS I/O Facilities, section 5.3.
AI-12-7-84-6:	Ray Sandborgh. Generate a decision model for tool support.
AI-12-7-84-7:	REQWG. Consider outstanding comments on draft 1 of the Requirements Document.

3.5 Action Items From E&V Meeting 4-7 March 1985

AI-3-7-85-1:	Jinny Castor. Obtain WIS Compiler Guidelines from Capt. Percy Saunders and put them on the EV-INFO directory.
AI-3-7-85-2:	Jinny Castor. Send message to E&V Team indicating how to edit the HERMES template to allow CC:.
AI-3-7-85-3:	Jinny Castor. Update EV-TEAM.INFO.HLP file.
AI-3-7-85-4:	Jinny Castor. Update EV-TEAM-MAIL.HLP file.
AI-3-7-85-5:	Jinny Castor. Prepare public exchange records for E&V briefing - SIGAda in San Diego, CA, and E&V briefing - Local SIGAda San Diego, CA.
AI-3-7-85-6:	Jinny Castor. Modify EV-INFORMATION login file to send comments to EV-INFO.
AI-3-7-85-7:	Jinny Castor. Contact ECLB administrator to request that mail sent to EV-INFORMATION be redirected to EV-INFO.

AI-3-7-85-8:	Jinny Castor. Send copy of E&V Plan to Ronnie Martin.
AI-3-7-85-9:	Jinny Castor. Modify master copy of E&V Plan to reflect: 1. Tools/Aids Requirements Document is draft in FY85 and version 1.0 in FY86. 2. Changes to definition of E&V: E - performance measurement V - conformance measurement
AI-3-7-85-10:	Jinny Castor. Modify E&V viewgraph to reflect above E&V definitions.
AI-3-7-85-11:	Jinny Castor. Send travel voucher/cost form to E&V members.
AI-3-7-85-12:	Jinny Castor. Coordinate with Jimmy Williamson to develop 6 STARS Task Area descriptions for Tech Coord Strategy Document.
AI-3-7-85-13:	Jinny Castor. Notify E&V Team when Public Report is available through DTIC.
AI-3-7-85-14:	Jinny Castor. Change E&V Plan schedule to reflect that the draft Tools and Aids Requirements Document version 1.0 is due next quarter.
AI-3-7-85-15:	Greg Gicca. Confirm availability for presentation at June E&V meeting.
AI-3-7-85-16:	Greg Gicca. Draft of SDME into SEE taxonomy.
AI-3-7-85-17:	Al Deese. Draft of ALS into SEE taxonomy.
AI-3-7-85-18:	Bard Crawford and Marlow Henne. Draft of ALS/N into SEE taxonomy.
AI-3-7-85-19:	Paul Reilly and Stacy Reddan. Draft AIE into SEE taxonomy.
AI-3-7-85-20:	Liz Kean. Update the project reference list for those documents referenced in the APSEWG Document.
AI-3-7-85-21:	Marlene Hazle. Draft attribute definitions for Requirements Document.
AI-3-7-85-22:	Ronnie Martin. Draft Q/A section of Requirements Document.
AI-3-7-85-23:	Mike Meirink, Rick Contreras, and Bob Fritz. Draft the Tools & Aids Requirements Document.

AI-3-7-85-24:	Rich Fleming. Review issues & prepare a draft rewrite of section 4.0 of the Requirements Document.
AI-3-7-85-25:	Pat Lawlis. Ensure Requirements Document conforms to document format set up by COORDWG.
AI-3-7-85-26:	Gary McKee. Provide overview/introduction of CAIS-MIL-STD comments by 20 March 85.
AI-3-7-85-27:	Gary McKee. Maintain all team members' comments of the CAIS-MIL-STD.
AI-3-7-85-28:	Darlene Sobota. Send Tricia a copy of APSE Validation Procedures Document version 1.0.
AI-3-7-85-29:	Darlene Sobota. Draft a letter for Gary McKee to be new CAISWG chairperson.
AI-3-7-85-30:	John Reddan. Send article on simulation of Ada processes to Tricia.
AI-3-7-85-31:	John Reddan. Develop the breakdown of the CAIS utilities, section 5.4.
AI-3-7-85-32:	E&V Team. Provide inputs to COORDWG for Project Reference List.
AI-3-7-85-33:	COORDWG. Obtain the KIT/KITIA glossary of terms and use it to create an E&V glossary.
AI-3-7-85-34:	COORDWG. Forward Tools/Aids questionnaire and obtain responses.
AI-3-7-85-35:	Don Jennings. Prepare boilerplate for working groups document's section 1.2 (background).
AI-3-7-85-36:	Don Jennings. Put a copy of the E&V document format on the NET.
AI-3-7-85-37:	Kathy Gilroy. Draft the requirements questionnaire for NET distribution.
AI-3-7-85-38:	Kathy Gilroy. Draft the compiler implementation dependencies for the Requirements Document.
AI-3-7-85-39:	Helen Romanowsky. Human/computer interfaces of Requirements Document.
AI-3-7-85-40:	Betty Wills. Update the E&V Project Reference List.
AI-3-7-85-41:	Rick Contreras. Put on the NET a message soliciting input for a software evaluator for computer resources of a delivered Air Force system.

ATTENDANCE LIST
E&V Team Meeting, 4-6 March, 1985

Stowe Boyd
GTE Gov't Systems
1 Federal St.
Billerica, Ma 01821

Michael Bridges
General Dynamics
Data Systems Division
PO Box 85808, MZ VP 5300
San Diego, Ca 92138

Jinny Castor
AFWAL/AAAF
Wright-Patterson AFB, Oh 45433-6543

Capt. Ricardo Contreras
Hq AFOTEC/LG5S
Kirtland AFB, NM 87117

Bard Crawford
TASC
One Jacob Way
Reading, Ma 01867

Capt. Al Deese
ASD/SIOL
Wright-Patterson AFB, Oh
45433-6543

Nelson Estes
ASD-AFALC/AXTS
Wright-Patterson AFB, Oh 45433-6543

Richard Fleming
Aerospace Corp.
M1/112
P.O. Box 92957
Los Angeles, Ca 90009

Capt. Ken Frankovich

Robert Fritz
CSC
4045 Hancock St.
San Diego, Ca 92110

Gregory Gicca
GTE Gov't Systems
1 Federal St.
Billerica, Ma 01821

Kathleen Gilroy
SPS
P.O. Box 361697
Melbourne, Fl 32936

Debra Harto
AFATL/DLCM
Eglin AFB, Fl 32542-5000

Howard Harvey

Marlene Hazle
MITRE Corp.
Burlington Rd.
Bedford, Ma 01730

Marlow Henne
Harris Corp.
GISD
505 John Rhodes Blvd.
Bldg. 1
Melbourne, Fl 32901

Don Jennings
OC-ALC/MMECE
Tinker AFB, Ok 73145-5990

James Johnson

Elizabeth Kean
RADC/COES
Griffiss AFB, NY 13441

Lt. James Kirkpatrick
AFALC/PTEC
Wright-Patterson AFB, Oh
45433

Maj. Allan Kopp
AJPO
Rm 3D139
(Fern St/C107)
The Pentagon
Washington, DC 20301-3081

Capt. Patricia Lawles
AFIT/ENC
Wright-Patterson AFB, Oh
45433

Randal Leavitt
PRIOR Data Sciences
39 Highway 7
Nepean, Ontario
K2H 8R6

Tim Lindquist
VPI and State Univ.
562 McBryde Hall
Blacksburg, Va 24061

Patrick Maher
OO-ALC/MMECF
Hill AFB, Ut 84056

Ronnie Martin
Georgia Institute of Tech.
Atlanta, Ga 30332

Gary McKee
Martin Marietta Aerospace
M/S 0423, P.O. Box 179
Denver, Co 80201

Michael Meirink
Sperry Corp.
DPG
P.O. Box 64525
St. Paul, Mn 55164

John Miller
SM-ALC/MMEHD
McClellan AFB, Ca 95652

Trisha Oberndorf
NOSC
Code 423
San Diego, Ca 92152-5000

Lt. Douglas Olson
HQ AFCMD/SI
Kirtland AFB, NM 87117

John Reddon
Syscon Corp.
3990 Sherman Way
San Diego, Ca 92110

Stacy Reddan

Helen Romanowsky
Rockwell International
400 Collins Rd NE
Cedar Rapids, Io 52498

Lt. Darlene Sabota
AFWAL/FIGRB
Wright-Patterson AFB, Oh 45433

Ray Sandborgh
Sperry Corp.
Knowledge System Ctr.
3001 Metro Parkway, Suite 223
Bloomington, Mn 55420

Capt. Percy Saunders

Lt. Patrick Sheridan
WIS Program Office
Hanscom AFB, Ma 01731

Jimmy Williamson
AFWAL/AAAF-2
Wright-Patterson AFB, Oh 45433

Betty Wills
CCSO/SKXD
TINKER AFB, OK 73145

APPENDIX E

MINUTES

of the

EVALUATION & VALIDATION (E&V) MEETING

5-7 June 1985

TABLE OF CONTENTS

SECTION PAGE

1.0 Wednesday, 5 June 1985

1.1 Welcome, Introductions and General Business

The E&V Team meeting opened with welcoming remarks by Virginia Castor, chairperson. Introduced to the team were Maj. Allen Kopp, AJPO; Maj. Kenneth Schoonover, HQ Systems Command; Jerry Brookshire, distinguished reviewer from Texas Instruments; Manda Suri, distinguished reviewer from Lockheed; Thomas Leavitt, distinguished reviewer from Boeing.

It was announced that:

- The E&V Technical Support Contract has not yet been awarded.
- The CAIS Validation Capability contract is in Procurement.
- The Ada Compiler Evaluation Capability announcement of request for proposal is scheduled to appear in the Commerce Business Daily (CBD) on 30 May. The original of the announcement will be available over ARPANET.
- Minutes of the March meeting are available on the NET under <DHARTO>MIN.TEXT.

Action items carried over from the December and March meetings were reviewed.

General Business included the following items:

- A list of E&V Team members was distributed and updates/corrections were requested.
- It was requested that all mail messages go to all team members.
- The schedule of future meetings was reviewed.
- KIT/KITIA review was discussed.
- It was announced that Jinny Castor has taken a position with the AJPO and will be moving to Washington D.C. The new E&V chairperson will be Raymond Szymanski.
- The STARS glossary reflects conflict in terms of Evaluation and Validation. The E&V Requirements Document is to be used as a baseline for the definition of these terms.
- The E&V Status Report is due for the Language Control Facility (LCF) Newsletter the week of 9 June. The COORDWG will address this during its working group sessions.
- The Schedule of Deliverables was reviewed.

1.2 Functional Capabilities Required to Support Software Test and Evaluation in Ada Programming Support Environments (APSEs)

Dr. Richard DeMillo, Project Director of STEP, Georgia Institute of Technology

The Software Test and Evaluation Project (STEP) was initiated by the Director Defense Test and Evaluation (DDT&E) in 1981, and is administered by the Under Secretary of Defense for Research and Engineering, whose office is responsible for large weapon system acquisitions.

The major goal of STEP is to improve the practice of software test and evaluation. The approach used, which is top down in nature, is to make policy changes and implement the changes, moving down to the technology hat supports software test and evaluation.

A study that began in 1981 at the request of DDT&E provided an overview of the state-of-the-art in software testing and reported the current view in the practice of software testing. Recommendations prompted by the study were:

- Upgrade the automated technology available
- Insert the technology into APSEs

The STEP program was tasked by STARS (Software Technology for Adaptable, Reliable Systems) to identify the functional requirements for technology to be inserted in Ada development work. Requirements were identified as:

- Capabilities to remove barriers which prevent adequate testing
- Capabilities to facilitate improved testing
- Implementation of the defined capabilities, facilitating interface with STARS efforts, assuring a two way flow of information.

The STEP view of the environment for testing operational software recognizes that:

- Testers use software differently from other users
- Programmers view testers negatively
- Testing may delay delivery of software
- The testing life cycle is not identical to the development life cycle
- Testing needs cut across needs of the rest of the development community.

Requirements for test and evaluation include identification of and response to constraints placed on design environments that support the testing community. These requirements are either general or specific in nature.

General requirements are those imposed on the tester by external requirements. They are not always technical in nature and have little impact on specific software design. General requirements state that:

1. Test and evaluation capabilities should be available at all points in the software life cycle where test and evaluation is needed, should be a component of existing decision support systems, and should feed into a life cycle-based information repository.

2. The system should be accessable via multiple interfaces to meet the variety of needs that exist among the various users. Different modes of operation are needed by different classes of users, including users who are not testers.

3. Integration of tools is needed in order to eliminate duplication of capabilities and allow the building of tools. Integration also promotes uniformity of interfaces and efficient context switching.

4. Isolation of testers is necessary because there is a need for integrity, reliablility, security and efficiency in the test process. In addition, isolation will prevent the software failures that occur during testing from affecting the rest of the user community.

5. Stability of evolution must coincide with what is important to the test community.

6. Host/target selection must be specified.

7. Testing applications are customized in the areas of life cycle phases, project-specific management, application-specific technology, classes of users, applicable policy, and applicable contracts, regulations and standards.

8. Evolution of test capabilities includes easy inclusion of new tools and methodologies.

Specific requirements may take the form of a wish list. They are set forth to insure technical feasibility, and may include the comment, "it will not work the way it should unless..." The wish list requires:

1. Classification of capabilities by function and Joint Services Software Engineering Environment (JSSEE) structural features. This classification is accomplished with respect to life cycle, level of information required, and level of services required.

2. Types of development functions provided. A tool building capability is better than providing specific tools and increases the possibility of compatibility. A test building capability is more flexible than providing specific tests. Other services include test description/preparation, test analysis, and degree of summarization services.

3. Ability to control test related processes.

4. Human Factors that make the test process more pleasant, thereby increasing productivity and effectiveness of the testers.

A final draft of the report describing requirements and rationale for tests is available this summer. Copies can be obtained bv contacting Dr. DeMillo at Georgia Institute of Technology.

1.3 VHSIC Hardware Description Language (VHDL) Program
Dr. John Hines, Very High Speed Integrated Circuits (VHSIC) Program Office

VHDL is the government effort to produce a good standard hardware description language. There are currently fifty seven different non-standard hardware description languages in use. VHDL is to hardware documentation language what Ada is to software documentation language.

The VHDL contract was awarded in August 1983. Intermetrics is the prime contractor with IBM and Texas Instruments as subcontractors. Intermetrics, together with IBM is responsible for language definition. Intermetrics is solely responsible for implementation of the compiler parts of the language system, and Texas Instruments is responsible for the simulator.

Phase A, the definition phase of the project, produced a series of language definition documents. This phase ended in July 1984 with version five of the language definition. This document underwent major industry review and tool analysis, which resulted in version seven. This document will be the baseline for the implementation of the VHDL. A final document, which will incorporate minor changes to version seven, is due December 1985. Phase B, the implementation phase, has been in progress since October 1984.

The VHDL project, a part of the VHSIC program within the Department of Defense, is required to have close Ada ties. The VHDL implemented language structure is very similar to Ada to the extent that VHDL could be considered an extended subset of Ada, extended in the sense that some user defined procedures, functions and packages have been added. It uses Intermediate Design Language (IDL) to define the form of the language, which is somewhat like a data base schema. CAIS Version 1.3 was used in the design library.

Hardware design deals with an object-oriented philosophy, structure and behavior. VHDL attempts to provide the levels of abstraction necessary to depict high level design and maintain the structural aspects of the designer's viewpoint. Entity models allow the designer to decompose to any level of the design hierarchy. The system allows the designer to configure things as desired, and a smart linker performs efficient simulation.

The compilation takes place in an analyzer which produces an intermediate file similar to Diana. This file is dumped into a design library. A simulation can then be performed to verify the description.

The simulator generates a simulation model, called the simulation kernel, which is analogous to a run-time system on a computer. Intermediate VHDL Annotated Notation (IVAN) generates an Ada program which is compiled and executed.

In addition to components mentioned above, the Virtual Memory Management (VMM) system has been developed, and the final version delivered.

1.4 Distinguished Reviewers

1.4.1 Single Project/Multiple APSE's
Jerry Brookshire, Texas Instruments
(subject assigned to Requirements Working Group)

The fundamental underlying concern in this area is in regard to distributed APSEs and their communication mechanisms. Large scale system development includes components of mission critical software requiring concurrent design and development, potentially involving a variety of target computers. Overall system operational requirements could dictate a large volume of multi-way concurrent communications between functional nodes.

Approaches to addressing the problems inherent in a single project using multiple APSEs are:

- Begin with a single large central development APSE and move toward the development of smaller remote versions. In this instance, it is possible to insure compatibility by designing it in.

- Multiple APSEs are required at the beginning of many projects. This situation requires that incompatibilities be addressed as they are identified.

Issues that need to be resolved include host/development station compatibility, database handling, and multilevel software development.

Considerations in addressing these issues could include: providing similar facilities across different targets, initially developing all software at a large central facility to avoid database problems, and working with a project database not resident at smaller remote facilities. Host-to-host compilation for debugging and early testing during development is a possible requirement. Host-to-target code generation may require a simulator to do target debugging in the beginning phases of a project when all development is confined to the host machine.

1.4.2 Ada Program Library System
Thomas Leavitt, Boeing
(Subject assigned to Requirements Working Group)

The library system is the structure in which program development occurs and is an important factor in usability. It is overlooked in some evaluation taxonomies.

In selecting a library system from those available, evaluation criteria must be established. These tend to be subjective in nature and include ease of use, parsimony, and elegance. Modifiable aspects of systems must be related to the context of the whole programming system. A very important aspect of the library system is support of controlled sharing of units in the library.

A means of evaluating library systems under consideration for a proposed Ada system is to require each supplier to describe how his system will perform each of the following scenarios:

1. Create a local executable file from a new main program unit which incorporates preexisting library units.
2. Create a new library unit.
3. Create a local executable file from a modified library unit without impacting existing files.
4. Replace a shared library unit with a new version.
5. Test the currency of a unit in a program without recompiling.
6. List all units which depend on a specified unit and all the units which a specified unit depends on in the context of a named program.
7. Delete a unit.
8. List the units in a library.
9. Given a target machine address, determine the corresponding Ada unit.
10. Copy units between libraries on different APSEs at different sites.
11. What version support is provided?
12. Identify the source text files, by version and date, which were linked together to form a specified executable program.
13. Estimate the learning time to know how to do all the above operations from the documentation provided.

1.4.3 Security in APSE
Manda Sury, Lockheed Missiles and Space Company
(Subject assigned to CAIS Working Group)

Software security is an essential feature of mission critical applications. Therefore the E&V task should include security considerations in evaluating APSE components and interfaces.

Security affects all the software attributes defined in E&V documentation: robustness, reliability, integrity, correctness and completeness. Security can be defined as:

- Protection of valuable assets

- Prevention, protection and correction of vulnerabilities.

In determining what to protect, the value of the item under consideration should be compared to the cost of protecting it. Not everything warrants security. Items (assets) to be protected include information, objects, and programs.

Techniques for providing security include isolation and mediation. Isolation keeps threat of security breaches away from the item being protected, whereas a mediator acts as a go between for the object and would-be users. Basic components of a protection mechanism are:

- organizational policy stating who can access each item
- person or process that implements the policy
- monitoring capability

The National Bureau of Standards defines a hierarchy of users as ownership rights, delegation rights, and access rights. Security is in conflict with user-friendliness and interoperability, particularly when users are allowed to act in different role options. A balance between these entities must be achieved.

A recommendation is that E&V consider that the access control mechanism described in the proposed CAIS document (version 1.4) serve as a preliminary standard for evaluating the security of software in APSE components and interface sets.

1.5 Open Discussion

The open discussion was lead by Major Allan Kopp and took the form of a question and answer session.

The first question concerned the personnel makeup of AJPO. It was explained that the director, technical director, and secretary are representatives of the Department of Defense. The new Technical Director is Paul Cohen. Deputy program directors represent the Army, Navy and Air Force.

A series of questions were asked concerning the status of Ada compiler validation standards. Answers to these questions and related issues were covered in a briefing presented Thursday, 6 June 1985 during the general session of the E&V Team meeting.

Discussion ensued concerning milestones and goals of the AJPO. Issues of particular interest are the insertion of CAIS into the STARS program, the Ada insertion policy, and Ada education. An education working group is organized (Software Engineering Education Working Group - SEEDWG) that focuses on education in the area of transitioning technology.

Following the open discussion and a lunch break, working groups met for the remainder of the day.

2.0 Thursday, 6 June 1985

2.1 Announcements

1. The E&V Team INFO-HELP file listing was again distributed for corrections.

2. The E&V Team Public Report will be mailed to the distinguished reviewers.

3. The agenda for Thursday morning was modified to include a briefing by Major Allan Kopp and the ARTEWG Status Report.

4. Virginia Castor announced that she would be availabl to conduct training sessions on the ARPANET during the lunch hour.

5. The draft Technical Coordination Strategy Document, Version 2.0 is completed and copies are available for team members.

2.2 Component Evaluation Criteria
Greg Gicca, G.T.E Government Systems
WIS Program

One of the major effort of the Software Development and Maintenance Environment (SDME) project at GTE will be to build an environment using existing Ada technologies. The objective of the evaluation criteria is to determine which software products currently available are acceptable.

Components are evaluated according to an established list of criteria. Information is gathered through a four phawse review process that determines whether or not the component should be procured.

General categories of evaluation criteria include functional applicability, understandability, testability, evolvability, efficiency, portability, and human engineering. In evaluating functional applicability, issues of usability, methodology independence, procurement cost, data rights, and correctness of output are considered.

In evaluating understandability, source features such as source code modularity, granularity, descriptiveness of documentation, traceability, programming methodology, and completeness of documentation are considered.

In evaluating testability (ease of verification), descriptiveness, modularity, and instrumentation are examined.

Evolvability (the potential of modifying or expanding a component) is evaluated in terms of flexibility, reusability of source code modules, interoperability, descriptiveness, granularity, machine independence, and system independence.

Efficiency of a component is rated in regard to data storage and system response.

Portability is determined by the programming language used and the degree of machine independence and system independence inherent in the component.

Evaluation in terms of human engineering considers the issues of user interaction, error recovery, error messages, on-line help facilities, completeness of documentation, system response, integrity, and the degree of training required.

The evaluation process that gathers the information takes place in four phases: a high level initial component review, an initial component evaluation, a more detailed component evaluation, and a specific component evaluation. The Initial Component Review evaluates initial information at a high level regarding functional applicability, evolvability, portability and human engineering. Reviewers' comments and basic component/vendor information is input at this time. The initial component evaluation decomposes these four areas of evaluation, then evaluates the component in these areas with further comments and recommendations from reviewers' specialized perspectives. The first two phases of evaluation are top-down in nature. The third stage of review further decomposes the functions and characteristics to be evaluated and produces more detailed comments and recommendations from reviewers, who evaluate only from their area of expertise. The final phase, specific component evaluation, rates the component as acceptable, unacceptable, or comments that there is not sufficient information upon which to make a determination. Components are compared, so that the most acceptable item will be procured.

Current requirements and evaluation criteria address technology that is available now, but may not address technology available a few years into the future. The fourth phase of evaluation has been designed to allow for the addition, subtraction, and changing of present evaluation categories.

2.3 Ada Validation Policy "Policeman"

Major Allan H. Kopp
Air Force Deputy Director, Ada Program

Although the Ada community needs a written policy, at the present time there is no formal validation policy in effect. The current unwritten policy addresses only part of the problem that exists. At present, a yearly revalidation of compilers is specified.

A DoD Ad Hoc committee has been formed to draft a policy statement for validation. The committee includes Maj. Allan Kopp and Paul Cohen of the AJPO, Pete Fonash of the Army, Bill Wilder of the Navy, and Ken Schoonover of the Air Force. Policy formation is envisioned as taking place on three levels. The policy will be maintained at the OSD (AJPO) level. Procedures will be developed at the DoD component and AVO level. Validations are to be conducted, reported and administered at the AVF level.

The draft Validation Policy defines responsibilities, identifies the scope of validation, and addresses validation schedules, revalidation, domain of validation, validation reporting, and compiler maintenance.

The scope of validation includes reporting conformance to standards of the tested compiler, but does not indicate the suitability of a compiler for a particular purpose, nor does it replace a set of application-specific requirements. Validation activities do not measure performance or capacity of a compiler, but do include preparation, maintenance, and distribution of the validation test suite.

Validation schedules allow for periodic validation. Initially, validation of a compiler has been for a one year period; however, validation periods may be extended for the convenience of the government. Revalidation has been required yearly or when major software upgrades are made. A problem occurs when validation certificates expire before revalidation can take place. New policy considers that a validated compiler remains validated for a given mission critical project.

The item being validated is the compiler in its executable form. The host machine is termed a virtual machine. This allows the recognition of derived validations of equivalents to the virtual machine. The "equivalent" definition is the responsibility of the vendor. These derived validations expire at the same time that the original validation expires.

Validation reporting addresses not only the tracking and approval of Validation Summary Reports (VSR) but also publication of VSR results and of reasonable challenges. No policy has been established concerning the rescinding of challenged validations; however, placing a deadline on vendor corrections to the stated problem is being considered as a possible solution. Several issues in the area of revalidation are in need of resolution.

At the present time, the old Ada Compiler Validation Certification (ACVC) system is being used. The new policy is in the process of being "sold" to the validation community. It has been presented in briefings, but the printed form is being witheld, pending Tri-Service review.

2.4 ARTEWG Status Report
Kathleen A. Gilroy, SPS, Inc.

The Ada Run Time Environments Working Group (ARTEWG) is a merger of the Run-Time Planning Group established by KIT/KITIA, the Common Run Time Interfaces Working Group of the user subcommittee, and the Distributed Systems Working Group of the user subcommittee. The purpose of this group is to develop products and services for the Ada community. Goals and objectives stated in the ARTEWG charter are:

- Establish conventions, criteria, and guidelines that promote reusability of Ada components, improve the performance of Ada components, and provide for the evaluation and selection of RTEs.

- Provide interface mechanisms between members of the community that will promote quality RTE implementations and identify/resolve Ada RTE issues.

A plan of action for achieving the stated goals involves these tasks:

- Elaborate Ada implementation dependencies

- Examine approaches used in building RTEs and categorize Ada implementation approaches

- Categorize applications and their RTE requirements

- Map the requirements onto implementations

- Derive commonality of RTE interfaces

A list of products to be used in implementing the identified tasks include a catalogue of implementation dependencies designed for use by application programmers, a catalogue of implementation approaches for RTEs, classification of application requirements, guidelines for use of Ada RTE for application programmers, and a catalogue of RTE interface options. The list of products reflects needs that exist in the Ada community. These needs generally address cross-target compilers and military applications.

It is anticipated that by-products to be generated will include a file of Ada RTE issues, Appendix F documentation requirements, a dictionary of RTE terminology, guidelines for evaluating RTEs and possibly an Ada run-time "Transportability Handbook."

The ARTEWG is sponsored by SIGAda and endorsed by AdaJUG and the AJPO. The group consists of twenty principal members and a larger number of advisory members who contribute their time on specific issues that are in line with their areas of expertise. Within the ARTEWG, there are three working groups that deal with implementation dependencies and approaches, application requirements, and common RTE interfaces.

Areas of interest to ARTEWG that are not currently being addressed include:

- A feasibility demo of "minimal RTE"

- Programming support environment/RTE relationship

- Definition of pragmas for RTE configurability

- RTE extensions

- Data transfer protocols

- Planning and/or conducting classes on fundamentals of the Ada RTE for program managers or other novices

- Forums and workshops

- Formulating training/education guidelines

The principal link between ARTEWG and the E&V Team is one of communication.

At the conclusion of the above presentations, working groups convened.

3.0 Friday, 7 June 1985

3.1 CAIS Operational Definition Status Report
Dr. Timothy Lindquist, Virginia Polytechnic Institute

The project objective is to create an operational semantic definition of CAIS written primarily in Ada. The operational definition will provide for design refinements for the current version of CAIS and serve as input to version 2 of CAIS. It will provide a vehicle for tool transportability studies, for the examination of CAIS functionality, and for tool retargetability studies. In addition, it will provide input to the development of the CAIS validation capability by developing validation tests, identifying and resolving specification gaps, and by operationally testing validation tests. Theoretically, the operational definition will be the next step in a sequence of more formal specifications.

A progress report stated that preliminary versions of CAIS List Utilities, CAIS Node Management, and CAIS Process Control should be available in August 1985. At the present time, the Dynamic String Package is 90 percent complete, CAIS private routines to support node management and process management are 80 percent complete. Node management done in MIL-STD CAIS is 33 percent complete, and is totally completed in non-compilable code. List Utilities are 80 percent complete, and Process Management, updated to the proposed MIL-STD, is 25 percent complete.

Some CAIS/Ada issues in need of resolution have been identified. When using the node management routines to manipulate attributes, one finds that the value of the attribute has to be a list type. Therefore the tool must use TO_LIST to convert a string to a list before creating an attribute, and TO_TEXT to convert a value back to a string after a "get". It is recommended that overloads should be defined for attribute routines. A second issue involves LIST_UTILITIES. Compiling the specifications introduced an inconsistency Text explaining LIST_UTILITIES defines NAME_STRING, but the appendix does not. The package NODE_DEFINITIONS does not constrain NAME_STRING. A problem arises because NAME_STRING is returned as the value of a function, causing difficulties for the CAIS user and the implementer. Therefore, either the tool must invoke functions returning NAME_STRING in a cumbersome manner, or implementation must constrain NAME_STRING. The latter solution violates the specification. Steps will be taken to insure adherence to the specification. Solutions to the problem are being evaluated.

Work is being done in the area of generating validation tests from abstract machine descriptions. The tests are to be administered in a black box manner using a white box technique to develop the tests. An Ada-based abstract machine description of the CAIS is used as input to the technique. Paths are isolated by using symbolic execution to identify the number of paths to test in a validation suite and to isolate input/output pairs that would cause those paths to execute. The input/output pairs are then converted into validation tests. The ultimate result of using the technique presented will be a more complete set of validation tests.

Preliminary work is being done in the area of establishing a definition of CAIS input/output. Efforts are being made to distinguish between input/output that can be written in Ada only and those portions that are machine dependent.

The final effort to be reported concerns Ada packages for encapsulating CAIS elements. A problem regarding "conflict of interest" exists in that an appendix to an earlier version describes two approaches to defining packages for the CAIS in a compilable way

- Access to the tool using CAIS is limited

- Access to the CAIS implementation packages is unlimited.

No resolution on this issue has been reached.

3.2 STARS Status Report
Lt. James Kirkpatrick, AFALC/PTEC

The report presented highlights of the STARS Application Workshop which was held 9 - 12 April 1985 at the Naval Research Laboratory, Washington, D.C. Focus of the workshop was software reusability. Working groups addressed the issues of taxonomy, incentives, libraries, design/integration, and metrics.

The taxonomy working group concentrated on the issues of common terminology and defining levels of reusability of software.

The working group on incentives identified problems in the proposal (bid) evaluation process that discourage the use and development of reusable software. Proposals that involve reuse of software are eliminated because they seem unrealistically low, whereas proposals incorporating development of reusable software seem unrealistically high. Possible resolutions to this problem were outlined. Discussions regarding the testing of reusable software brought forth reference to MIL-STD 2167 (Tri-Service), MIL-STD 1679 (Navy) and MIL-STD 1679A (DoD).

The library working group addressed issues of Configuration Management (CM) and maintenance of reusable software. The library is critical to reusability.

The design/integration working group dealt with assessing the current status of, and determining future approaches to, the design and integration of reusable software. Two areas examined were the designing of reusable parts, and the development of designs using reusable parts. Two dimensions of reuse are vertical (reusing software associated with a specific application) and horizontal (reusing software across application areas). It was noted that sociological problems exist with software reuse.

The working group on metrics highlighted methodology and measurements to be used in the development of reusable parts. RADC methodology, RADC metric work sheets, and Software Evaluation Reports of STARS Measurement DIDs were recommended. In using reusable parts, information concerning development, quality levels, functionality, operational history, and interface should be obtained for all parts under consideration by the user.

3.2.1 STARS RADC Quality Metrics
Marlow Henne

The STARS Quality Metrics represents a process one can use to evaluate a program. Thirteen entities are evaluated, such as transportability, modularity, etc. A contract with Defense Mapping Agency (DMA) funding was awarded a year ago. This contract will automate the tools used in this process. STARS has expanded this effort to include Ada software. The result is anticipated in 1986.

3.3 Working Group Status Reports

3.3.1 Requirements Working Group (REQWG) Status Report

The Requirements Working Group Report was presented by chairperson, Pat Lawlis. Three new members of the working group were announced: Karyl Adams, Jerry Brookshire, and Tom Leavitt. Deliverables due included a draft Version 2.0 of the Requirements Document and a draft Version 1.0 of the Tools and Aids Requirements Document. Accomplishments reported included completion of updates to the Requirements Document in the areas of Attributes Definitions, Required APSE Evaluations and Validations, and Quality Guidance. Other accomplishments were the conducting of a Tools and Aids Survey, drafting of a Tools and Aids Requirements Document, and the writing of a document on assessment of Availability of Tools and Aids. Key issues addressed concerned a need for public awareness of the availability of tools and aids, prioritization of tools and aids needed, and consideration of "Whole APSE" issues. Unresolved problems or action items are: sending the glossary from the Requirements Document to IDA, constructing a decision matrix for the Tools and Aids Document, obtaining CAIS input to Tools and Aids Document, sending a draft copy of the Tools and Aids Document to team members, development of rationales for requirements, and developing a strategy for acquisition of tools and aids. Projected work for the next quarter includes distribution of a revised draft of the Availability Assessment Document to team members, first try at developing an availability matrix, distribution for comment of a list of "Whole APSE" issues to team members, review of relevant material to expand and elaborate whole APSE issues, allocation of whole APSE issue items to working group members, and preparation of a short description for allocated whole APSE items.

3.3.2 APSE Working Group (APSEWG) Status Report

Personnel changes included the return of Guy Taylor and the replacement by Christine Stacey of Stowe Boyd. No deliverables were due this quarter. Accomplishments this quarter included the finalization of taxonomy mappings, the beginning of writeups, and the evaluation of "Boiler-Plate" portions of the document. Action Items presented include completion of the first draft of the ALS Decomposition and the forwarding of review comments to Guy Taylor, documentation to be done in the areas of evaluation, evaluation of ALS/N, and AIE. Projected work for next quarter includes completion of documentation in the area of environment and completion of SEE taxonomy mappings. The APSEWG taxonomy will include comments on ALS, ALS/N and AIE. No deliverables are due next quarter; however, a draft of the Analysis document is planned to be available for initial team review.

3.3.3 Standards Evaluation and Validation Working Group (SEVWG) Status Report

Jim Kirkpatrick, SEVWG chairperson presented the report of the SEVWG. Announcement was made that the name of the group has been changed from CAISWG to SEVWG, since the group will be dealing with evaluation and validation issues of other standards in addition to CAIS. New members of the group include Tim Lindquist, Kathleen Gilroy and Manda Sury. Darleen Sobota has been transferred to AFIT. SEVWG concerns included the evaluation of the task before the group and the planning of a single evolvable document dealing with issues relating to standards development. Work planned for next quarter includes the concentration on evaluation criteria while deemphasizing dependencies, planning of services to be performed by the E and V support contractor, and meeting with REQWG to discuss CAIS concerns which are applicable to both group . Deliverables included the APSE Components Validation Procedures Document (ACVPD). Accomplishments included the completion of version 2.0 of the ACVPD incorporating team comments and working on CAIS dependencies tests. Unresolved items include completion of dependency tests on MIL/STD CAIS sections. Projected work includes planning a new deliverable document concerning CAIS analysis and including sections on dependencies and evaluation criteria. A strawman of this document has been built. Deliverables due next quarter include an updated version of the ACVPD and a draft of the proposed document.

3.3.4 Coordination Working Group (COORDWG) Status Report

Don Jennings, COORDWG chairperson presented the status report. Accomplishments this quarter included the writing of the March minutes, the E&V Team Status Report, completion of an updated Project Reference List, and completion of a draft E&V Public Coordination Strategy Document, Version 2.0. Deliverables due included the draft E&V Public Coordination Strategy Document, which is ready to go out for comments. The Public Coordination Strategy Document, Version 2.0 has been completed ahead of schedule. Unresolved items include a need for team members to provide updates to project reference lists and a need for inputs to the Technical Coordination Strategy Document. Projected work for next quarter includes producing a status report, producing minutes of the June meeting, finalization of the Public Coordination Strategy Document, and updating the draft TCSD to include comments received over the net. Deliverables due next quarter include Version 2.0 of the Technical Coordination Strategy Document

3.4 Action Items

AI-6-7-85-1: J. Castor. Send the E&V Status Report to the LCF Newsletter

AI-6-7-85-2: J. Castor. Put the March minutes on MILNET <DHARTO>MIN.TXT

AI-6-7-85-3: J. Castor. Send Project Reference List item on the E&V Public Report to B. Wills

AI-6-7-85-4: J. Castor. Put Project Reference List on MILNET

AI-6-7-85-5: J. Castor. Put E&V meeting schedule on MILNET

AI-6-7-85-6: J. Castor. Send list of CAIS Validation Issues to E&V Team

AI-6-7-85-7: J. Castor. Send E&V Status Report to D. Jennings for publication

AI-6-7-85-8: J. Castor. Obtain MILNET accounts for Karyl Adams, Ray Szymanski, Amos Rohrer, Greg Gicca

AI-6-7-85-9: J. Castor. Update <EV-INFO>EV-TEAM-INFO.HLP file (note new name of SEVWG)

AI-6-7-85-10: J. Castor. Update <EV-INFO>EV-TEAM-MAIL.HLP file

AI-6-7-85-11: J. Castor. Call Lt. Pat Sheridan concerning copy of WIS SDME CPOP Annex

AI-6-7-85-12: J. Castor. Call Lt. Persons (AV 478-5980, ex 2694) for WIS compiler guidelines

AI-6-7-85-13: J. Castor. Send message clarifying CBD article on ACEC

AI-6-7-85-14: J. Castor. Send message notifying all that the E and V Public Report is available from DTIC.

AI-6-7-85-15: J. Castor. Send message telling how to set protection codes on files

AI-6-7-85-16: J. Castor. Send message to team limiting distribution of documents.

AI-6-7-85-17: J. Williamson. Contact K. Gilroy concerning input to TCSD about the ARTEWG

AI-6-7-85-18: T. Lindquist. Send a copy of CAIS requirements package to team members

AI-6-7-85-19: J. Brookshire. Call J. Castor for the guest MILNET access information (after 17 June: (202)694-0209 or AV 224-0209)

AI-6-7-85-20: COORDWG and WG Chairmen. Update Document boilerplate with statement concerning limited distribution to team only.

3.5 Discussion Items

1. It was announced that VPI papers are available for distribution to team members, thanks to the efforts of Tim Lindquist.

2. Liason status reports are valuable to team members; however, the time element at meetings is critical. Consensus seemed to be that a time limit should be set and enforced.

3. The Public Coordination Strategy Document indicates that all briefings on E&V by team members need to be reported. Reports were requested from Guy Taylor, who had briefed the Navy and Marlow Henne, who had briefed NATO.

4. It was noted that mapping of contractual efforts as related to E and V activicies should be done.

5. It was noted that a briefing should be provided at each E and V meeting concerning status of work being done by the support contractor. Inputs by working groups concerning how the contractor can be of support should be drafted.

6. It was announced that funding for E and V related travel is available to government personnel only.

7. It was announced that NASA has shown interest in E and V activities, as have several European countries, notably the English government and the German Navy.

8. It was announced that a presentation on IDA-Ada Prototype Compiler Benchmarks has tentatively been scheduled for the next meeting.

9. Liasons were asked to check the *Technical Coordination Strategy Document* to be sure that information concerning their areas is correct.

10. New chairman, Raymond Szymanski, was introduced to the group. His first executive decision was to decree, after considering input from team members, that the next meeting will be held 4 - 6 September as scheduled.

ATTENDANCE LIST
E and V Team Meeting, 5 - 7 June 1985

Adams, Karyl A.
AFWAL/FIGD
WPAFB, OH 45433

Bridges, Michael
General Dynamics
Data Systems Division
P. O. Box 85808, MZ VP 5300
San Diego, CA 92138

Brookshire, Jerry
Texas Instruments
Dallas, TX

Burlakoff, Mike
Southwest Missouri State Univ.

Castor, Virginia
AFWAL/AAAF-2
WPAFB, OH 45433

Crawford, Bard
TASC
One Jacob Way
Reading, MA 01867

Deese, Capt. Al
ASD/SIOL
WPAFB, OH 45433

De Millo, Richard
Georgia Institute of Technology
Atlanta, GA 30332

Fleming, Richard
The Aerospace Corp.
M1/112
P. O. Box 92957
Los Angeles, CA 90009

Fritz, Robert
CSC
4045 Hancock St.
San Diego, CA 92110

Gicca, Greg
GTE Government Systems
1 Federal St.
Billerica, MA 01821

Gilroy, Kathleen
SPS, Inc.
P. O. Box 361697
Melbourne, FL 32936

Hammons, Charles
Texas Instruments
P. O. Box 801, M/S 8007
McKinney, TX

Harto, Debra L.
AFATL/DLCM
Elgin AFB, FL 32542-5000

Hazel, Marlene
Mitre Corp.
Burlington Rd.
Bedford, MA 01730

Henne, Marlow
Harris Corp. GISD
505 John Rhodes Blvd.
Bldg. I
Melbourne, FL 32901

Jennings, Don
OC-ALC/MMECE
Tinker AFB, OK 73145-5990

Johnson, J.
AFWAL/AAAF
WPAFB, OH 45433

Kean, Elizabeth
RADC/COEE
Griffiss AFB, NY 13441

Kirkpatrick, James
AFALC/PTEC
WPAFB, OH 45433

Kopp, Maj. Allan
AJPO
Rm. 3D139 (Fern St/C107)
The Pentagon
Washington, D.C. 20301-3081

Lawlis, Patricia K.
AFIT/ENC
WPAFB, OH 45433

Leavitt, Thomas
Boeing Military Airplane Co.
Wichita, KS

Lindquist, Tim
VPI and State University
562 McBryde Hall
Blacksburg, VA 24061

Maher, Patrick
Magnavox
Fort Wayne, IN

Martin, Ronnie
Georgia Institute of Technology
Atlanta, GA 30332

Mahew, David
VPI and State University
Blacksburg, VA 24061

McKee, Gary
Martin Marietta Aerospace
M/S 0423, P. O. Box 179
Denver, CO 80201

Miller, John
SM-ALC/MMEHD
McClellan AFB, CA 95652-5609

Reddan, John
SYSCON Corp.
3990 Sherman St.
San Diego, CA92110

Reilly, Paul
Data General
Westboro, MA

Rohrer, Amos
EG<u>G</u>
Manassas, VA

Romanowsky, Helen
Rockwell International
400 Collins Road NE
Cedar Rapids, IA 52498

Schoonover, Kenneth
HQ AFSC/PLRT
Andrews AFB, MD

Shirley, Jane
SYSTRAN Corp.
4126 Linden Ave.
Dayton, OH 45432

Stacey, Christine
GTE Gov't Systems
Billerica, MD

Sury, Manda
Lockheed Austin Div.
M/S T2-32, 30E
21[illegible] E. St. Elmo
Austin, TX 78744

Taylor, Guy
FCDSSA
Code 822
Dam Neck
Virginia Beach, VA

Williamson, James
AFWAL/AAAF-2
WPAFB, OH 45433

Wills, Betty
CCSO/SKXD
Tinker AFB, OK 73145

APPENDIX F

MINUTES

of the

EVALUATION & VALIDATION (E&V) MEETING

4-6 September 1985

TABLE OF CONTENTS

1.0 WEDNESDAY, 4 SEPTEMBER 1985

1.1 WELCOME, INTRODUCTIONS, AND GENERAL BUSINESS

The Evaluation and Validation (E&V) Team meeting opened with welcoming remarks by chairperson Raymond Szymanski. Speakers for the morning session, Dr. Timothy Lindquist of Arizona State University, Audrey Hook of the Institute for Defense Analysis, Dr. Greg Riccardi of Florida State University, and Dr. Bard Crawford and Peter Clark of The Analytic Sciences Corporation, were introduced.

It was announced that:

- Virginia Castor has become the Acting Director of the Ada Joint Program Office (AJPO).

- Lt. Commander Phillip Myers has joined the AJPO and is the liaison for the E&V Team. His MILNET address is <PMYERS@Ada20>.

- Marlow Henne has brochures on accommodations and maps of Harris Corp. available for those who plan to attend the December meeting. Brochures can also be obtained by contacting Debra Harto.

Action items listed in the June minutes were reviewed. All outstanding action items were resolved.

1.2 OPERATIONAL DEFINITION OF CAIS - TASK STATUS
Dr. Timothy Lindquist
Arizona State University

The Common APSE Interface Set (CAIS) is a set of kernel-level interfaces for building APSE tools. The CAIS is intended to be a single set of tool/system interfaces founded on the ALS/ACS, and is incrementally developed. The CAIS, which is to be DoD maintained, will provide a validation capability.

The CAIS scope, which is typical of operating systems, includes facilities for structures and files, processes, and devices.

The CAIS has evolved to a level of maturity that allows it to be considered a Draft Military Standard.

The need for an Operational Definition of CAIS (CAIS OD) was identified, which would allow execution of the interfaces.

The CAIS OD technical objectives were to operationally define the CAIS in as semantically complete a version as possible, written almost entirely in Ada, and based on the Draft Standard CAIS.

Results of the first phase of the project are preliminary definitions of List Utilities, Node Model, and Process Control and the production of associated documentation. In addition, an examination report of the input/output (I/O) section of the CAIS is being formulated. This report will define the components of the I/O section that can profitably be written in a high order language such as Ada, and which portions must be left to the underlying system.

Work remaining in the area of LIST_UTILITIES includes resolution of problems concerning the use of dynamic strings as tokens, and tweaking the float items and integer items that have been written.

Work remaining in NODE_MODEL includes completing some copy routines, completing ACCESS_MODEL, and addressing some management routines.

Work remaining in PROCESS_CONTROL includes completing the SUSPEND and RESUME routines, devising hooks to the underlying system, and finalizing status reporting routines for machine time, I/O units, and three other areas.

A total of approximately 9,000 lines of code are included in he sixteen major packages that make up these three sections of the CAIS OD.

Problems are being addressed in the areas of the CAIS specification itself, as well as the three areas of work addressed by the Operational Definition.

In the CAIS specification, Appendices B and C present package specifications and package bodies in compilable form. However, the use of private types and limited private types prevent compilation when real statements are added to the package bodies. A specific example is the use of NODE_TYPE within NODE_DEFINITION, which is limited private. Various packages, such as NODE_MANAGEMENT and PROCESS_CONTROL use NODE_TYPE, but because it is limited private, these packages cannot function adequately. The classification prohibits users of NODE_DEFINITION from altering node types. At the same time, routines in NODE_MANAGEMENT are unable to set or alter the value of a handle. A possible solution is for NODE_MANAGEMENT to call routines existing within NODE_DEFINITIONS to cause modifications to take place, adding a level of indirection. Reorganization is another solution that has not been tested for usability.

Problems in the area of LIST_UTILITIES center around constrained strings that are returned by routines. An example is the routine TO_TEXT, whose input is a list and whose output is a text string. The length of the resulting constrained string must be the length of the identifier into which it will go. This requires using an aggregate assignment "T:=" which allows the changing or setting of string length.

The CAIS specification requires that you have a declaration block within each tool in order to call the tool. Ideally, a CAIS implementation should define the string type; for example, one could define the type string to be a dynamic string, which would solve the problem. The current specification does not allow this, because in several instances strings are used to initialize operations. Ada needs to be extended to a more flexible orientation for strings which would allow more dynamic structures for applications such as CAIS.

Implementing the Operational Definition has identified several areas in the CAIS that need refinement. Examples include the procedures required to create node attributes, and the overloads (DELETE, SET EXTRACT, and EXTRACT) that one has to consider in order to remove elements from a list.

Node management problems within NODE_MODEL are being addressed. The implementation approach is that nodes are used as carriers for entities (device file and process entities) and structures.

There are primary and secondary node relationships within the CAIS. Primary relationships enforce a hierarchical structure within the CAIS, whereas secondary relationships allow for a directed graph structure that may be cyclic. Each node has one primary relationship but may have several secondary relationships with other nodes. CAIS node management is defined in such a way as to delete the primary relationship when you delete a given node. However, the secondary relationships still remain. CAIS implementation needs to remember the secondary relationships so that they can be referred to later, even though the node itself can't be. The least undesirable way of handling the situation is by use of a capability table, active node table, or binary tree which would allow the user to determine whether the object referred to by the relationship is still an active entity in the storage system. Each node would be assigned a unique 64-bit sequence number which would be used throughout the system. The table or tree would reference only currently active nodes. If a node is referenced by a dangling relationship, its sequence number will not occur in the structure, and the implementation will know that it is not an active node.

Problems in the area of access control focused on a subset of discretionary access mechanisms. Subjects (processes) are given access to objects by adopting roles. The type of access given to the subject is determined by a grant attribute pertaining to the relationship between the subject and the object. This relationship involves granting the subject necessary rights. There is currently a problem regarding the determination of rights provided to a subject granted by roles adopted. The discretionary access mechanism specified in the CAIS is felt to be more complex than is necessary. Work done to date includes implementation of discretionary access through to only one level of indirection without allowing necessary rights to be on the resulting rights list.

The implementation approach used in PROCESS_CONTROL has been to use Ada entirely. Each process node contains two tasks: the execution of the Ada program and the synchronization with the spawning process. Process synchronization is handled through these tasks. In order to be a real CAIS environment, it is necessary to link to an underlying system to provide for process initiation. Dynamic linking is not possible with Ada. The current Ada implementation requires that all programs that are spawned or invoked must be compiled with the CAIS, omitting dynamic linking. To date, it has not been possible to suspend and resume in Ada. These two operations must rely on underlying primitives.

The CAIS Operational Definition is perceived as being a vehicle for exercising transportability studies, and rehosting studies onto the CAIS. The Operational Definition can play an important part in generating a CAIS validation capability by excercising the validation suite and offering a rigorous approach to constructing validation tests. The CAIS OD will provide an excellent prototyping basis for any extensions to the CAIS. Candidates for CAIS extensions are a distributed environment and a modified access control mechanism.

The area most critical to the CAIS program is tool studies. Since many software engineering tools are perceived as undergoing minimal change in the next ten to fifteen years, the CAIS's success depends upon the ease with which tools can be transported to the CAIS. Automated testing and test generation tools, and configuration management are examples of tools that should be examined in terms of CAIS functionality. CAIS adoption would be greatly advanced by transporting well known tools, written in Ada, onto the CAIS.

The current goal is to have the CAIS OD on the NET by late October 1985, together with documentation and a disclaimer.

1.3 PROTOTYPE ADA COMPILER EVALUATION CAPABILITY (ACEC)

Audrey Hook, Institute for Defense Analyses (IDA)
Dr. Greg Riccardi, Florida State University (FSU)

The scope of this task was to create a test suite using existing tests in the public domain, develop a report writer similar to the Ada Compiler Validation Capability (ACVC), write a user's manual for executing the test(s), and prepare a final report to be used as input to future ACEC work.

The approach chosen for accomplishing this task was to use development teams made up of people with expertise in various areas, such as compiler builders, data base designers, and benchmark experts. This synergism, the coming together of individuals from differing disciplines to address a common problem, has proved effective and has resulted in the availability of a diversity of hardware and compilers for use in the evaluation.

Work was organized into two areas. The first area of effort was the development of a test suite. This involved gathering and analyzing tests, and organizing the tests into a test suite. The second task was to develop an architecture that would support the test suite, a report writer being a major component of this package.

Four teams addressed the effort. The teams and their areas of responsibility were:

- Team A (IDA) addressed project management, benchmark construction, validation, and Ada language usage.

- Team B (General Systems Group) addressed benchmark construction, architecture, database design, and Ada language usage.

- Team C (Florida State University) addressed compiler development, instrumentation, and Ada language usage.

- Team D (SIGAda) addressed Beta testing.

A diversity of hardware and software was used as the development base for instrumenting the tests. The development base at IDA consisted of a Data General mainframe running the ADE compiler, and a VAX running the DEC compiler. The development base at FSU consisted of a VAX with a DDC compiler, a Cyber 170/760 executing the AFATL Ada Compiler, and a SUN development station using the VERDIX compiler. Beta testing involved a variety of compilers including the Data General (DG) ADE compiler, VAX machines using TELESOFT, VERDIX, DEC, SD-ADA System Designers Limited, and SofTech's ALS compiler, and IBM with the Intermetrics ACS compiler.

Contributors to the test suite included IBM, SRI, Harris, Ada Fair '84, and SIGAda. Six hundred tests were collected; 136 were selected to be retained. Tests were selected that would measure Ada language feature performance rather than conformance or compiler architecture. The purpose of this task was to develop a very basic prototype for compiler evaluation capability, tests that were developed to investigate specific compilers were eliminated. All tests selected had to be unique and had to compile correctly. Tests were written in Ada only.

Generally speaking, benchmarks are programs that represent a specific workload and are used to demonstrate relative measures of capacity and efficiency for different computer systems or configurations. For this task, compiler benchmarks are programs that demonstrate the effect of specific language feature usage on the capacity of a computer system, and programs that demonstrate the limitations that would be imposed by the compiler on application development.

The test suite was organized into two categories. The normative part of the test suite dealt with language granularity, language constructs which must be implemented in a conforming compiler, and addressed performance and capacity. The optional part addressed compiler features and algorithms.

Tests were instrumented to determine where each would fit into the benchmark architecture. E&V Team criteria for evaluating benchmarks were used. The language feature tested by each test had to be determined, and the best version of each test had to be selected. Test objectives were described, and the type of statistics rendered (addressing compilation, execution, or both) was noted. It was decided that a single test instrumentation strategy would be implemented. In view of the fact that each test provides compilation and execution statistics, host/target dependent capabilities of the various tests had to be identified. All that can be done to make a given test portable is to define interfaces for using available statistics.

For each test, a control version of the program is run. The control version does not contain the language feature being tested. The difference in the execution time between the two versions shows the cost of the language feature. The goal of this procedure is to measure both CPU time and real time.

The test suite is designed so that the user can run a single test or use all the tests to obtain data. Test results can be weighted to provide data on the particular feature being scrutinized.

The purpose of this task was not to create large synthetic benchmarks or to rate language features. This effort can provide the user with an incomplete sampling of tests that allows him to select applicable tests and create his own synthetic benchmarks. Intended users of the test suite are programmers who are familiar with their Ada compiler, the host/target operating system, and resource accounting packages.

Work remaining includes evaluating Beta Test results and developing enhancements for Version 0 (Beta Test). Tapes of Version 1 of the ACEC test suite and the User's Manual are planned for delivery by the end of September 1985. A technical report will be written that documents the work done, lists tasks that could not be undertaken because of the time constraints imposed, and identifies areas that require research and data. The report is planned for release in late December. Point of distribution for the software developed will be the Language Control Facility (LCF) at WPAFB. The LCF Newsletter will announce the availability of this package in its October or November edition. Access by MILNET/ARPANET will not be provided.

1.4 E&V CLASSIFICATION SCHEMA

Dr. Bard Crawford, Mr. Peter Clark
The Analytic Sciences Corporation (TASC)

The two-fold purpose of this briefing was to present a progress report from the E&V Technical Support Contractor and to present the status of the development of the E&V Classification Schema.

The three major tasks of the support contractor are to develop documentation, to provide the E&V Team with technical support, and to provide configuration management. Documentation to be produced in performing the first task includes an E&V Classification Schema document, an E&V Reference Manual, and an E&V Guidebook. The Reference Manual and Guidebook are to be updated yearly, and the feasibility of automating the Reference Manual and Schema is to be studied. The second task addresses providing the E&V Team and its working groups and workshops with technical support. This does not include administrative support or report preparation. Task three includes the development of a Configuration Management Plan for tracking and controlling documentation, and the implementation of the Plan.

In general terms, the job of the E&V support contractor is to support the development of E&V technology, publicize it, and make it available to the rest of the Ada community. Tool builders form the underlying base of this technology, and users of tools and APSEs are the ones whose needs should be addressed in the development of tools and reports.

A summary of E&V technology is to appear in the form of the Reference Manual and Guidebook, preliminary versions of which are scheduled to be released in early 1986. The Reference Manual is envisioned as a functional index that would contain single page summaries of each E&V technology and refers to the Guidebook. The Guidebook is envisioned as containing detailed explanations of each E&V technology and providing guidance in application of these technologies.

The Classification Schema, a draft of which is due in October 1985, will provide a framework for the Reference Manual and will determine the design of the Guidebook.

Alternatives are being considered for the development of the Classification Schema. As a starting point, the E&V Plan was analyzed. This plan, which pictures a taxonomy showing components, four interface classes, and the five E&V categories, states that components should be identified and classified in terms of a well-defined classification schema. The original concept is good, in that although this is a fairly complex axis requiring a hierarchy of components, other elements are simple, direct and relevant to the task.

An alternative would be to use functions rather than components as an axis. Functions tend to be more stable than components. Also, users and builders of tools are more concerned with what the tool does than with what components comprise the tool. Another alternative would be to use attributes as an axis in place of interface classes. E&V categories are important, in that users need to be guided to consider these characteristics. However, they may be ultimately omitted from the Schema, depending on the terminology developed.

It is proposed that the Classification Schema be defined as a multiple dimension taxonomy or set of axes to classify items that are subject to E&V. Items can be individual tools, tool sets, or entire software engineering environments such as minimal APSEs (MAPSEs) and APSEs. The Schema is used to establish a framework of reference indices which provide a structure for E&V record keeping.

A major objective is to make the Schema easy to use by incorporating terminology that is familiar to the user, who is then directed to the appropriate technology via the Schema. The following are attributes of the Schema that the support contractor perceives as desirable:

- Stability. The Schema should be comprised of information that will not quickly become obsolete.

- Open-endedness. The Schema should be able to accommodate new combinations of functions, new attributes, and other new developments.

- Comprehensiveness. The Schema should be applicable to all phases of the software development life cycle.

- Conciseness. The Schema should provide a framework that is easily understood by a broad range of users.

- User-friendliness. The Schema should be oriented toward the concerns of potential users.

The terminology used in the Schema should coincide with any unique terminology used in MIL-STDs and DoD-STDs.

In defining the Schema, the evolution of the definition of APSEs is being considered. The 1980 "Stoneman" document defines an APSE as a data base, user interface, system interface, and tool sets. This evolved into the 1983 National Bureau of Standards (NBS) definition of an APSE as a collection of input functions and output, with the data base being absorbed into both input and output, and interfaces and tool sets being absorbed into various functions. Another definition of an APSE might be the collection of input and output into a set of objects and functions executing in a host environment. Although many tools are hardware independent, various host environments influence tool performance.

Based on this theory, and using the Software Engineering Environment (SEE) Taxonomy as a point of departure, a taxonomy using an object axis is considered. In this taxonomy, input and output are combined. It is felt that the SEE Taxonomy approach to handling input and output is unclear. Another departure from the SEE Taxonomy would be the use of a functional taxonomy as an axis with the removal of the life cycle, but retaining the top three levels of detail used in the SEE Taxonomy. The life cycle might then become a separate axis. (The third axis might be host environment.)

Attributes that shape requirements and their definitions need to conform to E&V objectives. Because of the obvious importance of attributes, a taxonomy for attributes would have perhaps six to ten categories, and lower levels of the taxonomy would contain more specific definitions. Users would have the capability of weighting attributes.

E&V categories determine methodologies used to assess elements. The category of an element derived from the object/function/host environment taxonomy is undefined; only the category for an element/attribute pair is defined. This is subject to change as the technology matures; thus, E&V categories as an axis seems improbable.

In selecting the axes to be incorporated in the Schema, the developer must consider concerns of potential users, and provide support the users' application areas. Possible combinations of axes include function vs. object, function vs. lifecycle, function vs. attribute, function vs. host environment, function vs. E&V category, and function vs. application area.

COMMENTS FROM THE E&V TEAM CONCERNING THE SCHEMA ARE REQUESTED.

1.5 The general session of the E&V Team Meeting was adjourned. Working groups met for the remainder of the day.

2.0 THURSDAY, 5 SEPTEMBER 1985

2.1 THE APSE INTERACTIVE MONITOR (AIM)

Tim Harrison
Texas Instruments

[This report is a summary of a presentation that was given to KIT/KITIA in July.]

The goal of the Naval Ocean Systems Center (NOSC) sponsored APSE Interactive Monitor (AIM) project, was to evaluate interfaces available in the ALS and AIE, the two government APSEs being developed. The AIM project was to design a tool that would be tested on both the ALS and AIE. The project's purpose was not to develop tool(s) that would run on both systems, but to gain experience in design and use of Ada, and to evaluate the interfaces provided by the ALS and AIE. Another objective was to produce reports documenting discrepancies between the two sets of interfaces and the problems inherent in those interfaces that would impact tool development.

AIM is a tool that acts as an interface between the APSE user and APSE processes. AIM coordinates the input and output from APSE processes, and provides a device-independent computer terminal interface. The AIM provides a multi-window image on the terminal screen which allows the user to monitor several processes simultaneously, and allows him to edit one process while observing another. The AIM command language interpreter is constantly available to receive commands, build images, start processes, arrange windows, and halt completed processes. The AIM requires the following interfaces: terminal control and communication; data base; and process control and communication. Process control and communication allows creation and deletion of processes, the suspension and resumption of process execution, and interprocess communication.

The AIM consists of a total of 22,000 lines of code which make up 240 compilation units. Reusable software components include a virtual terminal, a help package, system dependencies, LALR parser support which serves as a table reader and Ada driver, and general support packages such as queue, stack, etc.

Because the ALS and AIE did not become available for AIM implementation, alternatives had to be selected. The Data General ADE validated compiler was available and therefore was implemented. From this initial implementation, the AIM was transported to VAX/VMS. All modifications to the AIM required for this rehost were accomplished in one month.

The two environments differ in nature. On the DG, the APSE is entered from the AOS/VS command language interpreter, and it extends the command set from AOS/VS. On the VAX, the APSE is the VMS Ada Compilation System (ACS), which is entered from the VAX/VMS command language interpreter (DCL). The Ada command is available directly from DCL, and all other ACS commands are executed from within ACS. Both compilers were validated Ada compilers. The same operations were performed on both:

- Parse the entire Ada source file. If any syntax errors are encountered, compilation is terminated.

- Assuming no syntactic errors were detected, semantically check each compilation unit. If any semantic errors are detected, compilation terminates for that unit, but continues for the remaining units.

- Generate relocatable binary machine code for each correct unit, and update the program library accordingly.

The Ada compiler can be invoked from the command line or executed in a batch stream.

Functional capabilities that emerged for the two systems differ. Both systems have the capabilities of generating assembly language, generating debug information, specifying a different program library, suppressing all run-time checks, and compiling multiple files at one time. The differences between the two lists of capabilities did not cause major problems.

Differences between the linker functional capabilities of the two systems were more marked. The VAX linker allows the user to link in other languages, which allows a routine in any language VAX supports to call subprograms written in any other VAX supported language. Other sophisticated features are available on the VAX.

With the interactive source level debuggers on both systems, it was possible to separate points on exceptions and identify the statements containing errors. Both tools were easy to use, but both were relatively new, untried tools with some capability limitations. The VAX tool offers automatic recompilation.

The files generated by the compilers and linkers of both systems do not differ greatly in number, but do vary in file content. The DG system gives the user all files available, but the VAX provides the option of turning some files off. The VAX also allows the user to specify a maximum number of revisions retained.

Capabilities of the program librarian and of the configuration management tools are similar. The VAX is somewhat easier to use.

The text editors provided on the the two systems were not used because of requirements imposed by the Local Area Network (LAN) at Texas Instruments (TI), and because implementors preferred to use other editors. The editor used on the VAX provided more capabilities than did the one on the DG.

The electronic mail system on the VAX provides more capabilities than do the MACROs written on the DG.

Conclusions are that the two environments contain very similar tools. The VAX ACS is integrated into the VAX system more completely than is the ADE into the DG. The DEC compiler/linker generates more helpful error messages.

Problems encountered when implementing the AIM on the DG AOS included problems with system dependencies in the areas of terminal communication, process control, and process communication. A significant problem encountered with the DG compiler concerned storage allocation. This is particularly true when one is dealing with dynamic allocation and aggregates. Documentation was very inadequate in these areas. Other problems encountered related to syntax error detection, linker/library search list closure, and package ody dependency.

The rehost of AIM to the VAX required 2.4 man-months. Some of this time was used in dealing with system dependencies. On VMS, access is provided to all system services in Ada. All system services are contained in six packages. The dependency linking among them caused problems. For the sake of timeliness, system-dependent parts of the AIM were written in assembler for the rehost.

Details of the project are reported in a three volume Interface Report which will be available from DTIC. Information included in the report details the AIM project work, including lessons learned in Ada programming in such areas as constructs, tasking problems, problems with object-oriented design, and how this project maps to other life cycle projections.

AIM is available through the Ada Software Repository on MILNET and ARPANET. To receive information concerning tools available from this source, send NET mail to: ADA-SW-REQUEST@SIMTEL20, or contact Rick Conn at Texas Instruments, MS/8007, P. O. Box 801, McKinney, TX 75069, telephone 214/952-2139.

2.2 ANNOUNCEMENTS

1. Team members are asked to review "Definition of a Production Quality Compiler." This is a prototype requirement for the AJPO to levy on compilers and is complete with guidelines for applying the requirements. A questionnaire concerning capacity and performance accompanies the document. Five or six copies are available to those who wish to participate in the review.

2. Copies of the Technical Coordination Strategy Document, Version 2.0 are available to team members.

3. Review comments are requested from team members on the June minutes, copies of which are available.

4. Copies of viewgraphs used in the TI presentation on the AIM will be available.

2.3 The general session of the E&V Team meeting was adjourned. Working groups met for the remainder of the day.

3.0 FRIDAY, 6 SEPTEMBER 1985

3.1 CLASSIFICATION SCHEMA REVISITED
Dr. Bard Crawford, TASC

This presentation took the form of an open discussion. The first topic of discussion was a clarification of items to be classified by means of the framework or schema. These items were identified as individual tools, tool sets, and APSEs.

Concern was expressed regarding the best way to evaluate a tool. A question was raised as to whether it is better to classify b: tool or by function. It was decided that items should be classified by function.

The SEE Taxonomy was cited as a good example of a working taxonomy. It was stated that the soon-to-be-released Schema should use this taxonomy as it is, or clearly document those areas that differ from the SEE and explain the rationale for these differences. It is proposed that the Schema would remove the life cycle and use a strictly functional axis. The second and third axes have not been definitely selected yet. Candidates include object, phase, and attribute. The SEVWG had discussed the possibility of putting life-cycle phase into a data base and localizing it to a particular viewpoint if it is removed from the schema.

The taxonomy used is envisioned as being three dimensional but having additional indices for the benefit of users with interest in a specific phase. A user wanting to evaluate a single tool would be guided toward a single intersection of function vs. object. The document or automated system would then point toward an appropriate set of single function attributes, which would point toward the category and the E&V technology to be used in evaluating the tool. Other users may want to classify an entire APSE or a tool set. In these instances, the user would be led to cluster attributes or other attributes that would appear elsewhere in the document or automated system.

It was suggested that some of the functions in this taxonomy could include the three categories mentioned in the NBS taxonomy and the SEE taxonomy: transformation, management, and analysis. The comment was made that the categories mentioned in the NBS and SEE taxonomies should not clutter the list of functions, but are actually points of view, and can be separated into an index, as with life cycle phases.

To assist the user, the Reference Manual might include an index of common component names and another index of commercial names. The purpose of such indices would be to reference a set of functional code numbers to assist in finding one's way into the maze

Clusters and cluster attributes and how they could be addressed via the schema were discussed. The need for top-down analysis was identified. It was noted that schemas identify characteristics, but don't always tell how they fit together.

3.2 WORKING GROUP STATUS REPORTS

3.2.1 Coordination Working Group (COORDWG) Status Report

COORDWG Chairperson, Don Jennings, stated that there were no personnel changes. Accomplishments this quarter included the review of June minutes, the writing of the E&V Team Status Report, delivery of the Public Coordination Strategy Document, Version 2.0, and delivery of the Technical Coordination Strategy Document, Version 2.0. Deliverables due this quarter included the Technical Coordination Strategy Document, Version 2.0. The Public Coordination Strategy Document, which was due last quarter, was delivered in July. Key issues addressed included the E&V Status Report and the Technical Coordination Strategy Document, Version 2.0. There were no unresolved problems or action items. Projected work for next quarter includes producing the Status Report and the minutes. No deliverables are due next quarter. No presentations are planned next quarter. Inputs are needed for the Project Reference List. Those who have given briefings on E&V are urged to provide information for the Public Exchange Record.

3.2.2 Standards Evaluation and Validation Working Group (SEVWG) Status Report

One personnel change occurred on the SEVWG; Jeff Facemire replaces Bud Hammons. Deliverables due this quarter were the APSE Components Validation Procedures Document, Version 1.3 (ACVPD) and the CAIS Analysis Document Version 1.0, which will address all aspects of CAIS including evaluation, validation, scope, description, use, and evolution. Accomplishments this quarter included the collection of comments on the ACVPD from Virginia Castor and Patricia Oberndorf. Work is being done to integrate their comments into the next version of the document. Other work accomplished included establishing a strawman of the CAIS Analysis Document Version 1.0. Unresolved action items are the integration of final comments to the ACVPD that will move it from Version 1.2 to Version 1.3, and the development of dependency tests for CAIS Sections 5.2 and 5.3. Projected work for next quarter includes the closing of past action items and further development of the CAIS Analysis Document. No deliverables are due next quarter, but plans are being made to include the CAIS Analysis Document in the E&V Plan, with revisions planned every six months. A presentation is planned for the next meeting on the CAIS Analysis Document. The possibility of repeating this presentation at the January KIT/KITIA meeting will be explored. Possible research topics for the technical support contractor to investigate include conducting a survey of existing CAIS implementation efforts, and reporting on the purpose, problems encountered, and issues discovered in implementations. Possible transport of the CAIS OD from the DG system to another system such as the VAX was explored.

3.2.3 APSE Working Group (APSEWG) Status Report

One personnel change occurred on the APSEWG; Greg Burns replaces C. Stacey. Accomplishments this quarter included resolving the issue of AIE analysis by removing the AIE analysis from the APSE Analysis Document. The form and a major part of the content of the APSE Analysis Document were finalized. Version 2 of this document nears completion with some work still needed on appendices. Appendix E will map the ALS and ALS/N onto the SEE Taxonomy. No deliverables were due this quarter. Key issues addressed were the removal of the AIE from the APSE Analysis Document and the consideration of conducting a survey of commercial environments. This is a possible project if it is found to be legal. Unresolved problems or action items concern investigating the legality of surveying commercial environments. Projected work for next quarter includes completing the APSE Analysis Document Version 2.0 and the planning of a survey format. The only deliverable due next quarter is the APSE Analysis Document.

3.2.4 Requirements Working Group (REQWG) Status Report

Three new people joined the REQWG: Peter Clark (TASC), Sandi Mulholland (General Dynamics), and Nelson Weiderman (Software Engineering Institute). No deliverables were due this quarter. Accomplishments this quarter included updating Section 4 of the Requirements Document, updating the To ls and Aids Requirements Document, completing the Availability Assessment Document, beginning work on whole APSE requirements issues, providing input to the STARS glossary, and coordinating with the SEVWG concerning requirements for standards. Key issues addressed during this quarter included the examining of the focus of the REQWG and formulating requirements and recommendations. Recommendations are:

- Establish a repository for APSE tool evaluator, and methodology information.
- Include in the Configuration Management activity the capability to manage working group drafts/documents that are not necessarily Team products.
- Prepare a short document outlining the concept of Team operations, organization of working groups, and team focus.
- Establish a liaison with Ada Europe.
- Archive all NET mail to the EV-TEAM in EV-INFO
- Request a presentation for the next meeting from the STARS Methodology Coordinating Team outlining their tasks, particularly the Methodology Classification, Evaluation, and Selection Tasks.

Unresolved problems or action items include putting the Draft Tools and Aids Requirements Document and the Draft Version 2.1 Requirements Document on the NET and providing follow up to STARS glossary input.

3.3 ANNOUNCEMENTS

1. A presentation by the Software Engineering Institute is planned for the December meeting.
2. Selected products will be presented to SIGAda at their February meeting in Los Angeles. Presentations and deliverables for next quarter need to be very high quality so they will reflect the excellence of the E&V Team.

3.4 OPEN DISCUSSION

Questions were asked concerning the proper procedure for maiking software available to the public. The standard procedure is to send a letter to the Federal Software Exchange and submit the software to them for distribution.

The Team was informed that the sole source of ACVC tests is through ASD/SIOL and their contractor, SofTech.

It was noted that only legal users of the NET can use tools available through SIMTEL20.

A question was raised concerning whether the evaluation of methodologies is within the scope of the E&V Team. The technology for evaluating APSEs will, to some extent, involve methodology, but won't evaluate methodologies. Input from STARS is needed to make a determination. Consensus was that it is probably not a part of the charter, but if this is not being done by STARS, perhaps the task should be considered as a possible addition.

3.5 ACTION ITEM LIST

AI-9-6-85-1	SYSTRAN. Compile a list of all documentation distributed at the September meeting and include it in the minutes.
AI-9-6-85-2	SYSTRAN. Implement CM on the documents distributed at the meeting.
AI-9-6-85-3	Szymanski. Archive the Team mail.
AI-9-6-85-4	Szymanski. Locate and make available E&V Team viewgraphs.
AI-9-6-85-5	Szymanski. Open NET accounts for Nelson Weiderman, SEI and Peter Clark, TASC. Change P. Dobbs account to S.L. Mulholland, and Bud Hammonds account to Jeff Facemire.
AI-9-6-85-6	Szymanski. Investigate meeting with Ada Europe.
AI-9-6-85-7	Szymanski. Investigate the legality of the survey proposed by REQWG on commercial environments.
AI-9-6-85-8	Szymanski. Arrange for the STARS Methodology Team to give a presentation at the December meeting.
AI-9-6-85-9	Szymanski. Consult with ITARS and J. Castor on the Public Review problem.
AI-9-6-85-10	Szymanski. Consult with STARS to see if methodology should be included in the E&V charter.
AI-9-6-85-11	Harto. Send a message on the NET telling where to send visit requests for the December meeting.
AI-9-6-85-12	Jennings. Send the E&V Status Report to R. Szymanski at the KIT/KITIA meeting.

AI-9-6-85-13	Fritz. Put the Draft Tools and Aids Requirements Document on the NET.
AI-9-6-85-14	Fleming, Lawlis. Put the Draft Requirements Document, Version 2.0 on the NET.

LIST OF DOCUMENTS DISTRIBUTED AT THE SEPTEMBER E&V TEAM MEETING

1. The PAMELA Methodology - A Process-Oriented Software Development Method for Ada - DRAFT
2. DRAFT Minutes of the June E&V Team Meeting
3. Presentation Materials used in "A Technical Briefing: CAIS Operational Definition"
4. Presentation Materials used in "Prototype Ada Compiler Evaluation Capability (ACEC)"
5. Presentation Materials used in "The APSE Interactive Monitor"
6. Attendance List
7. Technical Coordination Strategy Document, Version 2.0
8. Definition of a Production Quality Compiler (5 - 6 copies only)

LIST OF ATTENDEES

Adams, Karyl A.
AFWAL/FIGD
WPAFB, OH 45433

Brookshire, Jerry
Texas Instruments
M/S 3114, P. O. Box 660246
Dallas, TX 75266

Burns, Greg
GTE/WIS
1 Federal Street
Billerica, MA 01821

Clark, Peter
TASC
1 Jacob Way
Reading, MA 01867

Crawford, Bard
TASC
One Jacob Way
Reading, MA 01867

Deese, Capt. Al
ASD/SIOL
WPAFB, OH 454433

Estes, Nelson
ASD/AXT
WPAFB, OH 45433

Facemire, Jeff
Texas Instruments
M/S 8007, P. O. Box 801
McKinney, TX 75069

Fleming, Richard
The Aerospace Corp.
M1/112
P. O. Box 92957
Los Angeles, CA 90009

Gargaro, Anthony
Computer Science Corp.
4045 SLB/810
304 West Route 38
Moorestown, NJ 08057

Gicca, Greg
GTE Government Systems
1 Federal St.
Billerica, MA 01821

Harto, Debra L.
AFATL/DLCM
Eglin, AFB, FL 32542-5000

Hazel, Marlene
Mitre Corp.
Burlington Rd.
Bedford, MA 01730

Henne, Marlow
Harris Corp. GISD
505 John Rhodes Blvd.
Bldg. 1
Melbourne, FL 32901

Jennings, Don
OC-ALC/MMECE
Tinker AFB, OK 73145-5990

Kean, Elizabeth
RADC/COEE
Griffis AFB., NY 13441

Kirkpatrick, James
AFALC/PTEC
WPAFB, OH 45433

Lawlis, Patricia K.
AFIT/ENC
WPAFB, OH 45433

Lindquist, Tim
Computer Science Department
Arizona State University
Tempe, AZ 85287

Maher, Pat
Magnavox
TC-10-C3, Dept. 542
1010 Production Road
Fort Wayne, IN 46808

McKee, Gary
Martin Marietta Aerospace
M/S 0423, P. O. Box 179
Denver, CO 80201

Meirink, Mike
Sperry/DPG
3333 Pilot Knob Road
St. Paul, MN 55164

Miller, John
SM-ALC/MMEHD
McClellan AFB, CA 95652-5609

Mulholland, S. L.
General Dynamics
Suite 735
6100 Western Place
Ft. Worth, TX 76107

Reddan, John
SYSCON Corp.
3990 Sherman St.
San Diego, CA92110

Riccardi, Gregory A.
Florida State University
Dept. of Computer Science
Tallahassee, FL 32306

Romanowsky, Helen
Rockwell International
400 Collins Road NE
Cedar Rapids, IA 52498

Sandborg, Ray
Sperry Knowledge System Center
Suite 223
3001 Metro Parkway
Bloomington, MN 55420

Shirley, Jane
SYSTRAN Corp.
4126 Linden Ave.
Dayton, OH 45432

Szymanski, Ray
AFWAL/AAAF-2
WPAFB, OH 45433

Taylor, Guy
FCDSSA
Code 822
Dam Neck
Virginia Beach, VA 23461

Weiderman, Nelson
SEI
Carnegie Mellon University
Pittsburg, PA 15238

Williamson, James
AFWAL/AAAF-2
WPAFB, OH 45433

Wills, Betty
CCSO/SKXD
Tinker AFB, OK 73145

Witt, Donald J.
AFIT/EN
WPAFB, OH 45433

APPENDIX G

CAIS OPERATIONAL DEFINITION PROJECT STATUS

I. RESEARCH DESCRIPTION

a. Description. The Ada program has made the evolution of a single set of kernel facilities to support Ada Programming Support Environment (APSE) tools a clear objective. As one avenue toward the objective the KIT/KITIA (Kernel APSE Interface Team/Industry and Academic) has developed an initial set of facilities to support APSE tools, which is called CAIS (Common APSE Interface Set, pronounced as case). A preliminary study has developed a specification technique for CAIS that enables a more complete validation capability to be constructed. Using an Ada-based abstract machine, a specification of CAIS Node Model and Process Control has been generated. This research contract has addressed converting the abstract machine definition of CAIS Process Control, Node Model and List Utilities into operational Ada.

b. Significant Results. The project has developed an operational version of CAIS Process Control, Node Model, and List Utilities. In this definition, Process Control has been defined using the tasking facilities of the language. Thus, independent CAIS processes have been defined using Ada tasks. CAIS operations on processes and interactions with the operating environment are constructed through both a concurrent and a sequential form of program invocation. The Operational Definition implements the process hierarchy and imposes a hierarchical task structure on each process. A CAIS process and its interactions with the operating environment are defined as a tree of Ada tasks.

Of further significance in the definition is our treatment of discretionary access mechanisms. CAIS proposed a generalized set of dynamic access controls which define how a subject may view an object. This set of facilities has become controversial from the perspectives of utility to tools, ability to be demonstrated secure and efficiency of implementation. Our definition is the basis for analysis of these issues.

Another result of the project is continued refinement of a technique for converting the operational definition of CAIS into a set of validation tests. Our goal is to provide a rigorous technique to generate a nearly complete set of test programs that may be used to exercise a CAIS implementation to assure its conformity to the specification. Two masters students have been funded through this project to develop the technique, and one of the papers presented during the reporting period describes the technique.

c. Plans for Next Year's Research. A continuation proposal is currently being reviewed by the Ada Joint Program Office and the Evaluation and Validation Team. The proposal would provide resources to complete the operational definition. Our plans are to finish the definitions of Node Model and Process Control, and to construct an operational definition of the Input/Output section of CAIS. Supporting this work is continued work on the technique for developing validation tests from the operational definition.

Currently, a graduate student working with Dr. Tim Lindquist at Arizona State University is developing a masters thesis whose focus is implementing the technique for generating input/output pairs automatically from the operational definition. The input/output pairs form the basis for constructing validation tests.

II. ACCESS CONTROL FOR THE COMMON APSE INTERFACE SET

This section contains a portion of a Master's Thesis by Douglas J. Bower. It describes the design of the Operational Definition of CAIS discretionary access mechanisms.

1.0 INTRODUCTION

One of the major objectives of the Ada program is to reduce software costs by increasing the transportability of Ada software. To meet this objective for Ada Programming Support Environment (APSE) tools, a common set of kernel facilities have been developed called CAIS (Common APSE Interface Set). Transportability of APSE tools will be greatly enhanced if tools are developed to rely only on the Ada language and CAIS interface. By providing implementations of CAIS as the basis of an APSE, tools will be more transportable.

In this paper a design for an operational definition of the access control mechanisms of the CAIS is presented. Before doing that a general overview of the CAIS, as proposed for Department of Defense Standardization [Ada JPO.MIL-STD_CAIS] is presented.

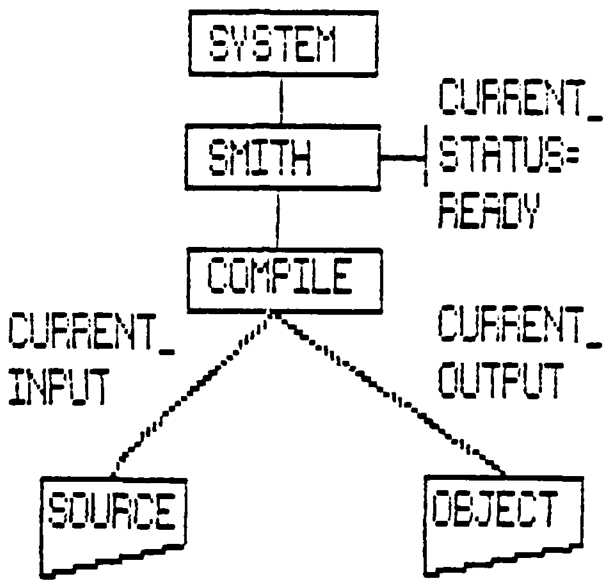
The CAIS provides interfaces to administer entities relevant to an APSE such as files, directories, processes, and devices. Each entity has various properties and may be interrelated with other entities. Within the CAIS a node serves as a carrier of information about an entity. A relationship represents an interconnection between two entities. An attribute represents a property of an entity or of an interconnection. The structure represented by CAIS nodes and relationships is that of a directed graph. The nodes form the vertices of the graph and relationships form the directed edges of the graph.

1.1 CAIS NODE MODEL

Three different kinds of nodes are identified within the CAIS: structural nodes, file nodes, and process nodes. A node may have contents and attributes and may be the source or target of relationships. The contents of a node depends on the kind of the node. A file node contains an Ada external file. A process node contains a representation of the execution of an Ada program. A structural node has no contents and serves as a holder of relationships and attributes. Nodes may be created, deleted, renamed, and accessed through CAIS operations.

A node may be the source or target of many relationships representing many different classes of connections. In order to distinguish among these different classes, the concept of a relation is introduced. A relation is a catagory which identifies the nature of a relationship. For example a relation called PARENT connects a newly created node back to its parent. The CAIS predefines certain relations and allows the user to define other relations. Some basic predefined relations are USER, DEVICE, JOB, CURRENT_JOB, CURRENT_USER, and CURRENT_NODE. According to this concept, each relationship is identified by a relation name and a relationship key. The relation name identifies the class of the relationship, and the relationship key distinguishes the relationship from other relationships of the same class emanating from the same node. Nodes can be obtained by traversing relationships, that is, following a relationship from its source to its target node. An example of a typical set of CAIS nodes and attributes is shown in Figure 1.

There are two kinds of relationship: primary and secondary. A primary relationship is established with a newly created node. Primary relationships are restricted to maintain a hierarchical structure of nodes. An example of a primary relationship is the predefined relation USER which emanates from the system-level node to a user top-level node. Primary relationships form a tree, in which there exists only one sequence of primary relationships from the root to each other node in the tree. A secondary relationship may be established between any two existing nodes. An example of a secondary relationship would be the connection between a source file and the corresponding object file which is created by compiling. The set of secondary relationships may form an arbitrary directed graph.



REPRESENTATION OF A COMPILATION

Figure G-1

A path is a sequence of relation name, relationship key pairs. A path starts at a known node (not necessarily top-level) and terminates at a desired node. Every accessible node may be reached by following some path from a system-level node to the given node which involves only the traversal of primary relationships. A node may be identified by a pathname. An example of a pathname is 'USER(JONES)TEXT_EDITOR(APPEND)TFILE which represents a path from user JONES to the node representing the file being edited (see Figure 2).

The properties describing nodes and relationships are maintained in attributes. Each attribute is represented by a named list which consists of an attribute name and a list of values. The CAIS predefines certain attributes which usually may not be manipulated by users. A user, however, may create and manipulate user defined attributes. Examples of attributes are found in Table 1.

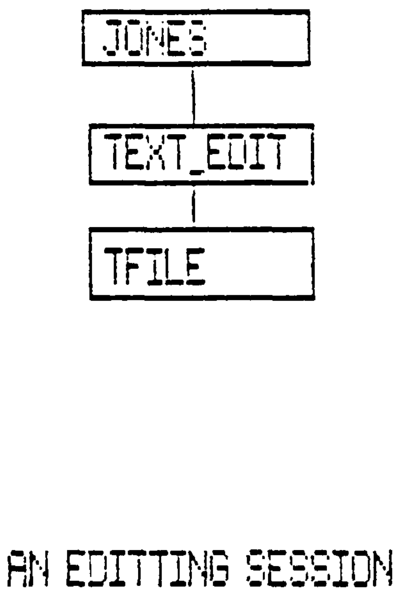
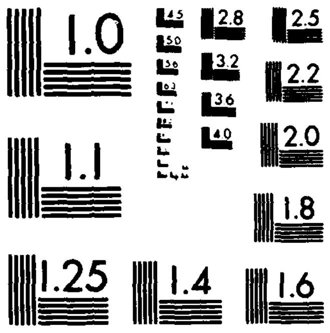


Figure G-2

Table G-1. Account and Quota Attributes

account_attribute_value ::= account_number => user_list
user_list ::= user_item | (user_item ,user_item)
user_item ::= user_name, password
user_name ::= identifier
password ::= string_literal
quota_attribute_value ::= account_number => user_list
quota_list ::= quota_item | (quota_item ,quota_item
user_name ::= identifier
quota ::= integer_number
Notation: 1. Words - syntactic catagories 2. [] - optional items 3. EL - an item repeated zero or more times 4. | - seperates alternatives

AD-A172 343 EVALUATION AND VALIDATION (E&V) TEAM PUBLIC REPORT 3/5
VOLUME 2(U) AIR FORCE WRIGHT AERONAUTICAL LABS
WRIGHT-PATTERSON AFB OH R SZYMANSKI 30 NOV 85
UNCLASSIFIED AFWAL-TR-85-1016-VOL-2 F/G 14/2 NL


1.0
4.5
5.0
5.6
6.3
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

1.2 CAIS PACKAGE STRUCTURE

The CAIS consists of several seperate, but related, Ada packages. The packages are grouped into four major areas (node management, list utilities, process management, and input/output). Each contain several supporting operations.

1.2.1 General Node Management

Node management consists of interfaces for the manipulation of structural nodes, relationships, and attributes. These interfaces are separated into five packages: NODE_DEFINITIONS, NODE_MANAGEMENT, ATTRIBUTES, ACCESS_CONTROL, and STRUCTURAL_NODES. An Ada type NODE_TYPE is defined for values that represent a node handle.

The package NODE_DEFINITIONS defines the Ada type NODE_TYPE. Handles are created from a pathname by the open operation. Open is performed to indicate that a node will be used in future operations. NODE_DEFINITIONS defines certain string and enumeration types and exceptions that are used for node manipulation. Tools would use this package to acquire visibility of types defined within the package so that objects of these types may be created for use with node management interfaces.

The package NODE_MANAGEMENT defines the general primitives for manipulating, copying, renaming, and deleting nodes and their relationships. These operations are generally applicable to all nodes, relationships, and attributes. Interfaces are provided for the manipulation of node handles and the performance of access synchronization. A tool could use routines from this package to open nodes, to obtain information about nodes and their relationships, and to modify the relationships among the nodes.

The package ATTRIBUTES supports the definition and manipulation of node attributes and relationships. The operations defined in this package are generally applicable to user defined attributes and not to attributes predefined by the CAIS. A tool could use routines from this package to define and modify information describing certain nodes and relationships.

The CAIS specifies two types of access control mechanisms: discretionary and mandatory. Discretionary access control involves dynamically granting to processes the right to perform certain operations on given objects. An example of discretionary access control would be to allow the current user to execute a given program file. Mandatory control is a static form of access control which involves restricting access to objects based on the sensitivity of the object and the clearance or authorization of the requesting process. For example an object with a secret clearance designated as a mail user could read any mail classified at the secret level or below. Within certain constraints, the user may manipulate discretionary access control information, however the user may not manipulate mandatory access control information.

The package ACCESS_CONTROL provides primatives for the manipulation of discretionary access control information for CAIS nodes. Routines are provided to establish, delete, and modify the necessary relationships. A routine is provided to determine if access rights have been granted. A tool could use routines from this package to grant or restrict access to objects under its control.

The package STRUCTURAL_NODES provides the ability to create nodes that do not have contents. A structural node only carries common information about other nodes related to it. A tool could use the interfaces of this package to create directories or configuration objects.

1.2.2 CAIS Process Nodes

Process management consists of interfaces for initiating and controlling the execution of Ada programs as represented by CAIS processes. The execution of a program including all of its tasks is represented by a process node. Predefined attributes of the process nodes maintain information about the process, such as CURRENT_STATUS, PARAMETERS, and RESULTS. Predefined relationships link a process node to its input, output, and error files. Process management is seperated into two packages: PROCESS_DEFINITIONS and PROCESS_CONTROL.

The package PROCESS_DEFINITIONS defines the types and exceptions associated with process nodes. Tools would use this package to acquire visibility of types defined within the package so that objects of these types may be created for use with process management interfaces.

The package PROCESS_CONTROL consists of interfaces for the creation and termination of processes and the creation and termination of process node attributes. Newly created process nodes have several secondary relationships established for them such as the predefined relation CURRENT_INPUT. The predefined relationships are initialized from input parameters to the invoking operation. Newly created nodes inherit several secondary relationships from the creating process such as all secondary relationships of the predefined relation CURRENT_USER. A tool could use the interfaces of this package to spawn several parallel processes.

1.2.3 CAIS Input and Output

Input and output management consists of interfaces for the control of the flow of data to and from four kinds of files: secondary storage, queue files, terminal files and magnetic tape drive files. Ada external files are represented by CAIS file nodes. Predefined attributes of a file node, such as ACCESS_METHOD, FILE_KIND, QUEUE_KIND, AND TERMINAL_KIND, maintain information about its contents and access method. There are ten input and output management packages: IO_DEFINITIONS, DIRECT_IO, SEQUENTIAL_IO, TEXT_IO, IO_CONTROL, SCROLL_TERMINAL, PAGE_TERMINAL, FORM_TERMINAL, MAGNETIC_TAPE, and FILE_IMPORT_EXPORT.

Package IO_DEFINITIONS defines the types and exceptions associated with file nodes. Tools would use this package to acquire visibility of types defined within the package so that objects of these types may be created for use with input and output interfaces.

Packages DIRECT_IO, SEQUENTIAL_IO, and TEXT_IO provide interfaces for input and output to their respective types of files comparable with those specified in the Ada Language Reference Manual. Files created using package DIRECT_IO are readable using package SEQUENTIAL_IO, if the two packages are instantiated with the same generic data type. A tool could use these packages to create, open, or delete Ada external files of the above types.

Package IO_CONTROL provides facilities to modify or query the functionality of CAIS files. Specialized facilities provide for: associating text files with output logging files, forcing data from an internal file to its associated external file, manipulating function keys and prompt strings, and creating mimic and copy queues. Tools could use the facilities of this package to set up a desirable input and output environment.

Three packages provide interfaces for terminal input and output: SCROLL_TERMINAL, PAGE_TERMINAL, and FORM_TERMINAL. Each package provides the functionality of the respective type of terminal listed here in order of increasing complexity. Tools can be constructed which handle input and output from a terminal at a higher level using the primitive routines of these packages.

Package MAGNETIC_TAPE provides interfaces for the support of input and output operations on both labeled and unlabeled magnetic tapes. The routines within this package provide specialized tape related functions such as: mounting, initializing, loading, unloading, and dismounting. After a tape is loaded the information from the tape is extracted using package TEXT_IO routines. A tool could use routines from package MAGNETIC_TAPE in conjunction with routines from package TEXT_IO to read and write information using magnetic tape.

The CAIS allows an implementation to maintain files separately from the files maintained by the host file system. Package FILE_IMPORT_EXPORT provides facilities to transfer files between these two systems.

1.2.4 CAIS Utilities

Within list utilities, the abstract data type LIST_TYPE is defined for use by other CAIS interfaces. A list (an entity of type LIST_TYPE) is a linearly ordered set of data elements called list items. A list may be named or unnamed. List items may be one of the following types: LIST_ITEM, STRING_ITEM, INTEGER_ITEM, FLOAT_ITEM, or IDENTIFIER_ITEM. Package LIST_UTILITIES defines the types, subtypes, constants, exceptions, and general list manipulation interfaces. Items of a list can be manipulated by: extracting items from a list, replacing or changing values of items in a list, and inserting new items into a list. Tools could use list utilities to create lists from external strings and insert and extract items or sets of items from such lists.

General Node Management, CAIS Process Nodes, CAIS Input and Output, and CAIS Utilities provide interfaces to file, directory, process, and device management services. These services are traditionally provided by an operating system and affect tool transportability. If all APSE tools are implemented using only the Ada language and the CAIS, maximal transportability will be acheived.

2.0 DESCRIPTION OF ACCESS MECHANISMS

The CAIS requires that mechanisms for discretionary and mandatory access control be established. According to the CAIS specifications, discretionary access control is "a means of restricting access to objects based on the identity of subjects and/or groups to which they [the subjects] belong. The controls are discretionary in the sense that a subject with certain access permission is capable of passing that permission (perhaps indirectly) on to any other subject." Mandatory access control "provides access controls based directly on a comparison of an individual's clearance or authorization for the information and the classification or sensitivity designation of the information being sought."

In the CAIS, access control consists of three basic components (access control rights, access control rules, and access checking). Access control rights describe the kinds of operations that may be performed. Access control rules are the rules which determine which access control rights are required for an intended operation. Access checking involves the determination of whether the granted access rights are sufficient for permitting the intended operation.

Access to a node is defined in terms of the operations which may be performed on the node. Four classes of operations constitute access to a node:

1. reading or writing the contents of the node

2. reading or writing of attributes of the node

3. reading, writing, or traversing relationships emenating from the node

4. reading or writing attributes of relationships emenating from the node

5. traversing the node.

The following operations do not constitute access to a node:

1. closing node handles to a node

2. opening a node with intent EXISTENCE (see Table 2)

3. querying the kind or status of node handles to a node

4. reading or writing of relationships pointing to a node.

A node is inaccessible if access to the node is not permitted after either discretionary or mandatory access checking. The property of inaccessibility is relative to the process currently requesting access to a node and not a property of the node itself.

2.1 DISCRETIONARY ACCESS CONTROL

Discretionary access control provides a means for specifying the operations a subject may perform on objects. An object is the node to which access is requested. A subject is a process (which is acting for a user) intending to perform an operation which requires access to the object. A role node is a structural node which serves as an intermediary between subjects and objects. Object nodes have established for them secondary relationships of the relation ACCESS (which is a predefined relation). These relationships define the operations which are allowed to be performed on the node and by virtue of their destination who may perform these operations. These relationships emanate from the object node and have targets which are role nodes. A subject may have established for it relationships of the relation ADOPTED_ROLE (which is a predefined relation). These relationships emanate from the subject node and have targets which are role nodes. The combination of these two kinds of relationships determines the access rights to the object which have been approved for the subject. The approved access rights are then compared to the intentio: of the subject to determine if an operation may be performed.

2.1.1 Access Rights

An object node may be the source of any number of access relationships to one or more role nodes. Each access relationship has an attribute called GRANT. The predefined GRANT attribute specififes the access rights of the object which can be granted to subjects. Access relationships and grant attributes are established in either of two ways: at node creation or explicitly using the interfaces of package ACCESS_CONTROL. The SET_ACCESS_CONTROL procedure (of package ACCESS_CONTROL) has two possible uses. It may be used to establish an access relationship between two nodes and set the value of the grant attribute of that relationship or, it may be used to change the value of a grant attribute of an existing ACCESS relationship. If the ACCESS relationships are to be established at node creation, a check is made to determine if creating such a relationship is permitted. If the relationship is permitted, then SET_ACCESS_CONTROL is called to establish the access relationship. The ACCESS relationship is defined using the key and GRANT value from the ACCESS_CONTROL parameter of the calling procedure. Regardless of how access relationships are established or modified the process carrying out such operations is required to have sufficient access rights to the object.

The value of the GRANT attribute is a list whose syntax is consistent with that of a CAIS list as supported by package LIST_UTILITIES. The BNF for the GRANT value is given in Table 2.

Table G-2. GRANT attribute BNF

grant_attribute_value::= ([grant_item{,grant_item}])
grant_item::= ([necessary_right=>]resulting_rights_list)
necessary_right::= identifier
resulting_rights_list::= identifier | (identifier{,identifier})
Notation: 1. Words - syntactic catagories 2. [] - optional items 3. {} - an item repeated zero or more times 4. | - seperates alternatives

Although access rights may be user defined, the CAIS has defined a set of predefined access rights. These are found in Table 3.

Table G-3. (Part 1 of 4). Predefined Access Rights

EXISTENCE	The minimum access rights without which the object is inaccessible to the subject. Without additional access rights the subject may neither read nor write attributes, relationships or contents of the object.
READ_RELATIONSHIPS	The subject may read attributes of relationships emanating from the object or use it for traversal to another node; the access right EXISTENCE is implicitly granted. This access right is necessary to open the object with intent READ_RELATIONSHIPS.
APPEND_RELATIONSHIPS	The subject may create relationships emanating from the object and attributes of these relationships; the access right EXISTENCE is implicitly granted. This access right is necessary to open the object with intent APPEND_RELATIONSHIPS.
READ_ATTRIBUTES	The subject may read attributes of the object; the access right EXISTENCE is implicitly granted. This access right is necessary to open the object with intent READ_ATTRIBUTES.

Table G-3. **(Part 2 of 4).** Predefined Access Rights

WRITE_ATTRIBUTES	The subject may create, write, or delete attributes of the object; the access right EXISTENCE is implicitly granted. This access right is necessary to open the object with intent WRITE_ATTRIBUTES.
APPEND_ATTRIBUTES	The subject may create attributes of the object; the access right EXISTENCE is implicitly granted. This access right is necessary to open the object with intent APPEND_ATTRIBUTES.
READ_CONTENTS	The subject may read contents of the object; the access right EXISTENCE is implicitly granted. This access right is necessary to open the object with intent READ_CONTENTS.
WRITE_CONTENTS	The subject may write the contents of the object; the access right EXISTENCE is implicitly granted. This access right is necessary to open the object with intent WRITE_CONTENTS.
APPEND_CONTENTS	The subject may append contents of the object; the access right EXISTENCE is implicitly granted. This access right is necessary to open the object with intent APPEND_CONTENTS.

Table G-3 (Part 3 of 4). Predefined Access Rights

READ	This is the union of READ_RELATIONSHIPS, READ_ATTRIBUTES, READ_CONTENTS, and EXISTENCE access rights. This access right is necessary to open the object with intent READ. It is sufficient to open the object with intent READ_RELATIONSHIPS, READ_ATTRIBUTES, or READ_CONTENTS.
WRITE	This is the union of WRITE_REALTIONSHIPS, WRITE_ATTRIBUTES, WRITE_CONTENTS, and EXISTENCE access rights. This access right is necessary to open the object with intent WRITE. It is sufficient to open the object with intent WRITE_RELATIONSHIPS, WRITE_ATTRIBUTES, or WRITE_CONTENTS.
APPEND	This is the union of APPEND_RELATIONSHIPS, APPEND_ATTRIBUTES, APPEND_CONTENTS and EXISTENCE access rights. This access right is necessary to open the object with intent APPEND. It is sufficient to open the object with intent APPEND_RELATIONSHIPS, APPEND_ATTRIBUTES, or APPEND_CONTENTS.

Table G-3 (Part 4 of 4). Predefined Access Rights

EXECUTE	The subject may create a process that takes the contents of the object as its executable image; the access right EXISTENCE is implicitly granted. This access right is necessary to open the object with intent EXECUTE.
CONTROL	The subject may modify access control information of the object; the access right EXISTENCE is implicitly granted. This access right is necessary to open the object with intent CONTROL.

2.1.2 Adopting Roles

A role, gives a subject a set of access rights that it may use when acting under the authority of that role. A role may be associated with a user, a program in execution, or a group of users, programs, or subprograms. Roles are acquired dynamically. A user may act under the authority of any number of different roles simultaneously. A role is represented by a node. This node may be a top level process node representing the file containing the executable image of a program or a structural node representing a group.

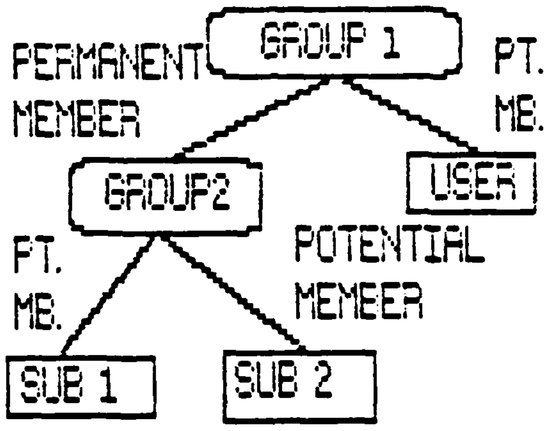
A structural node representing a group has relationships emanating from it whose target nodes represent members of the group. There are two kinds of relationships that can be used to identify group members. The first is the primary relationship of the predefined relation PERMANENT_MEMBER. The permanent member relationship may be used to create a hierarchy of users and/or groups of users. However, a user top level node may not be a permanent member of a group since it may only have a primary relationship from the system level node. The second kind of group relationship is the secondary relationship of the predefined relation POTENTIAL_MEMBER. This relationship identifies those subjects which may dynamically acquire membership in the group. Potential members of a group include all members identified by the relationship

POTENTIAL_MEMBER along with all permanent members. In order to adopt the role representing a group, a subject must be must be a potential member of the group. An example of a group is found in Figure 3.

When a process adopts a role, a secondary relationship of the relation ADOPTED_ROLE is established from the subject node (representing the process) to the role node. A process node may be the source of several ADOPTED_ROLE relationships. Roles may be either adopted at node creation or adopted explicitly. At node creation, a process node automatically adopts the role represented by the file node representing the executable image of the program that is currently executing. At creation, a root process node adopts the role of the current user node. Processes explicitly adopt roles using the procedure ADOPT of package ACCESS_CONTROL.

2.1.3 Evaluating Access Rights

The evaluation of discretionary access control rights involves the derivation of relevant grant items and approved access rights from the values of GRANT attributes. Relevant grant items are grant items that are in values of GRANT attributes of the access relationships emanating from the object and pointing to a role node representing an adopted role of the subject or a group node representing a group of which



GROUP RELATIONSHIPS

Figure G-3

an adopted role node is a permanent member. An approved access right is an access right which meets one of two criteria. First, the access right may be a necessary right for which the resulting rights list is null. Second, the access right may be within a resulting rights list of an approved necessary right. Examples of evaluating access rights follow in Figures 4 and 5.

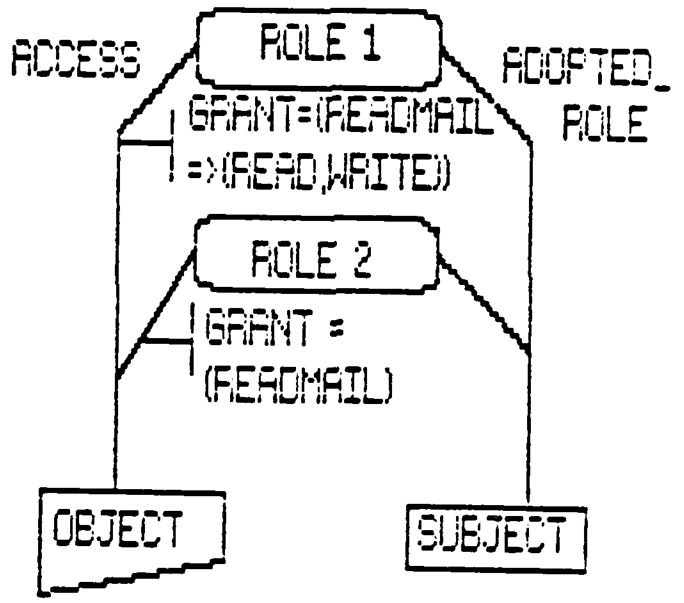
In Figure 4, a subject has adopted two roles, ROLE 1 and ROLE 2. From ROLE 2 the subject acquires the approved access right READMAIL. From ROLE 1 along with the approved necessary right of READMAIL the subject acquires the approved access rights of READ and WRITE. This combination of access rights is required for the subject to be permitted to perform the operations of read and write on the object.

Figure 5 represents part of a set of relationships that could be found within a project. SUBJECT 1 has adopted ROLE 1 and ROLE 3 and assumed the role of a project leader. SUBJECT 2 has adopted ROLE 2 and ROLE 3 and assumed the role of a project programmer. Through ROLE 3 both subjects acquire the approved access right EDIT. Using ROLE 1 along with the approved necessary right EDIT, SUBJECT 1 has approved access rights of READ and WRITE to both objects. Using ROLE 2 along with the approved necessary right EDIT, SUBJECT 2 has the approved access right of READ to OBJECT 1 and approved access rights of READ and WRITE to OBJECT 2.

Within the CAIS discretionary access checking is done at the time a node is opened. This is accomplished by comparing the INTENT parameter of the procedure OPEN with the approved access rights of the node to be opened. If the INTENT is not an approved access right then an exception is raised. If approved, the INTENT value can be used for comparison when other operations are attempted.

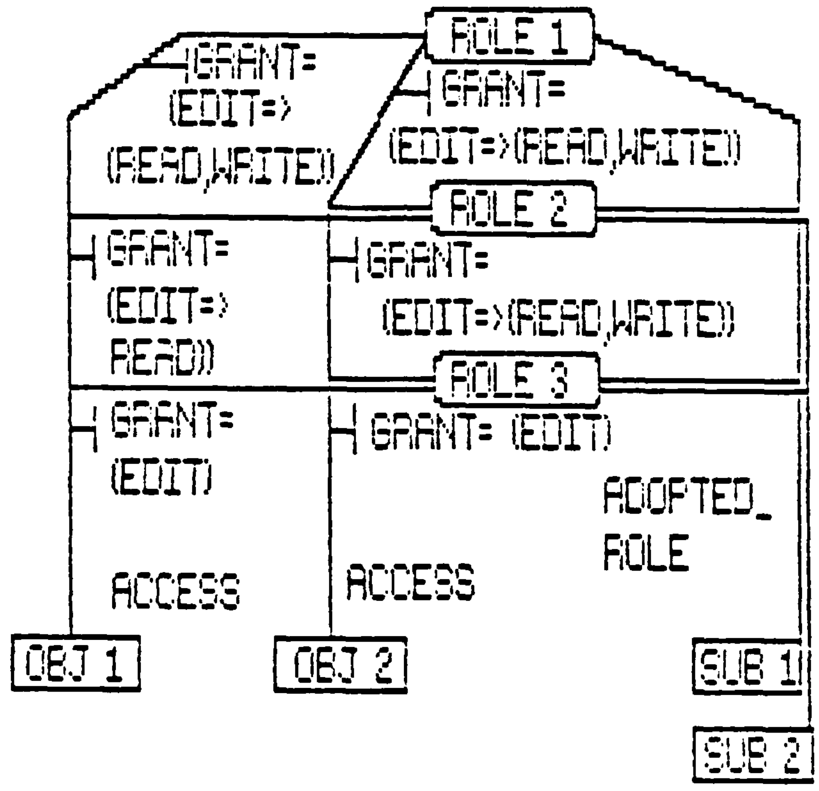
2.2 MANDATORY ACCESS CONTROL

Mandatory access control provides a mechanism for information security which may not be altered by CAIS users. There are two types of mandatory access control classification (hierarchical and non-hierarchical). A hierarchical classification level is one of an ordered set of classification levels. This classification represents either the trustworthiness of a subject or the sensitivity of an object. In order for a reading process to obtain access to an object it must have a hierarchical classification greater than or equal to that of the object being read. On the other hand, a writing process must have a hierarchical classification less than or equal to that of the information being written.



ACCESS RELATIONSHIPS

Figure G-4



ACCESS RELATIONSHIPS

Figure G-5.

Each subject and object is assigned any number of non-hierarchical catagories to represent coexisting classifications. A reader must be assigned each of the non-hierarchical catagories assigned to the object read. An object written to must be assigned each of the catagories of the writer. Both hierarchical and non-hierarchical access rules must be satisfied before a subject can obtain access to an object. Mandatory access checking is carried out at the time that an operation is requested by the subject. The classification of the subject and that of the object are compared before the operation is permitted.

In order to accomplish mandatory access checking nodes need to be labeled with their mandatory access classification. Process nodes have both an object and a subject classification, since process nodes can be both the subject and object of operations. Non-process nodes only have object classifications. Nodes for devices have two object classifications: LOWEST_CLASSIFICATION and HIGHEST_CLASSIFICATION to represent the range of information that may be handled by the particular device.

Mandatory access control rules are checked when access control is enforced for a given operation. If an operation violating mandatory access control rules is attempted then the operation is not permitted and a SECURITY_VIOLATION exception is raised. Figure 6 illustrates mandatory access labeling for a reading process and an object being read.

2.3 SUMMARY

The combination of the discretionary and mandatory access control mechanisms provides the basis for the complete package of CAIS ACCESS_CONTROL. The discretionary access control mechanisms provide the user a means of explicitly restricting access to information. The mandatory access control mechanism provides a means for maintaining information security.

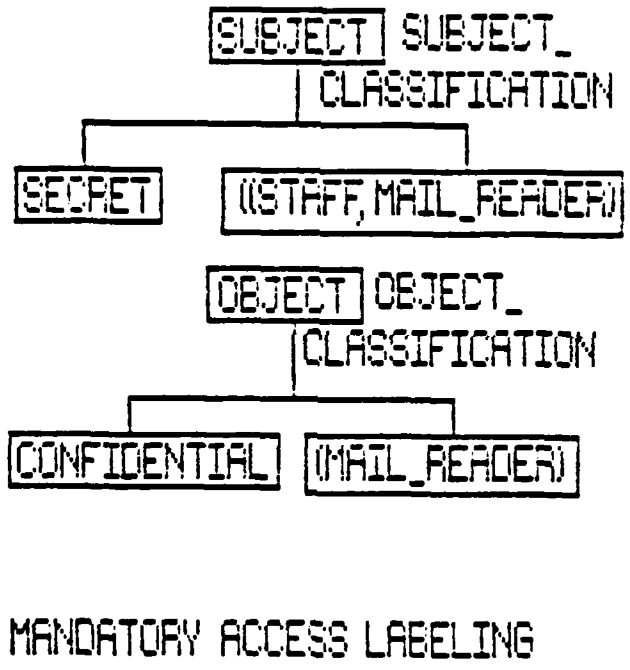

SUBJECT
SUBJECT_CLASSIFICATION
SECRET
((STAFF, MAIL_READER))
OBJECT
OBJECT_CLASSIFICATION
CONFIDENTIAL
(MAIL_READER)
MANDATORY ACCESS LABELING

Figure G-6.

3.0 DESIGNER IMPLEMENTATION

The implementation of the discretionary and mandatory access control mechanisms of the CAIS posed interesting information problems. Seperate strategies were chosen for the implementation of each mechanism. The discretionary access control mechanism closely adheres to the suggested implementation outlined within the CAIS specifications. The mandatory access control mechanism is implemented in a slightly different manner than the suggested implementation scheme. Any variations from the suggested implementaion strategy do not alter the interfaces specified by the CAIS.

3.1 DATA STRUCTURES

The discretionary access control mechanism uses many data structures that are previously defined within the CAIS. Mainly, the discretionary access control mechanism is highly reliant on the CAIS relationships and CAIS LIST_UTILITIES. As described previously, a discretionary access right is a cross of two relationships, the ACCESS relationship and the ADOPTED_ROLE relationship. It was necessary to develop mechanisms for creating and evaluating such a complicated cross of relationships. Such mechanisms are described in detail in the sections of this paper which describe the individual ACCESS_CONTROL procedures.

The mandatory access control mechanism still uses many previously defined data structures, but also introduces a structure of its own. It was suggested that the labeling of nodes for mandatory access control be accomplished by implementing such labels as predefined node attributes. In the interest of simplicity, a different approach is used. The mandatory access control labels have been made into fields of the node itself. A variant record approach is used since the labels are different for different kinds of nodes. By using this implementation approach the process of mandatory access checking is simplified. Mandatory access rights can be checked directly as opposed to searching through an entire attribute list. It is important that mandatory access checking be as simple as possible, since mandatory access checking should be done with almost every operation. The mandatory access control mechanisms do use CAIS LIST_UTILITIES to a limited extent in accordance with the suggested specifications. The precise details of mandatory access checking are described in the section on the underlying routine MANDATORY_CHECK.

3.2 SET ACCESS CONTROL

The procedure SET_ACCESS_CONTROL is a CAIS interface that sets access control information for a given node. The procedure has three input parameters: NODE of type NODE_TYPE, the object node handle; ROLE_NODE of type NODE_TYPE, the role node handle; GRANT of type GRANT_VALUE, the value to be assigned the grant attribute. If an access relationship does not exist from the object node to the role node, one is created. Also, a grant attribute of the access relationship is created. The effect of this procedure is to grant the access rights specified by the GRANT parameter to processes who have adopted the role represented by ROLE_NODE.

The SET_ACCESS_CONTROL_PROCEDURE is designed as follows. A check is made to determine if the object node and role node have been opened. If either is not open a STATUS_ERROR exception is raised. A check is made to determine if the object node is open with the intention of CONTROL. If this is not true an INTENT_VIOLATION exception is raised. The procedure MANDATORY_CHECK is called to make sure all mandatory access control constraints are met. Access to the object node is achieved using the procedure GET_ACCESS_NODE. The relationships emanating from the object node are searched to see if an access relationship exists to the role node. If no such relationship is found, one is created. If such a relationship is found, access to it is established using the procedure GET_ACCESS_REL. If the access relationship already has a grant attribute, its value is replaced by that of GRANT, otherwise, a new grant attribute with the value GRANT is attached to the relationship.

A second interface is provided by an overload version of SET_ACCESS_CONTROL. In this version of the procedure the object node and role node are referred to by their name strings. This version opens the object and role nodes using their name strings and gets node handles. It then calls the first version of SET_ACCESS_CONTROL using the acquired node handles. Finally, it closes the role and object nodes regardless of the termination status of the procedure call to the first version.

3.3 IS GRANTED

The function IS_GRANTED returns true if the current process, as a subject has an approved access right to the object node. Otherwise, it returns false. The function has two incoming parameters: OBJECT_NODE of type NODE_TYPE, the object node handle and ACCESS_RIGHT of type NAME_STRING, which is an access right.

The function IS_GRANTED is designed as follows. A check is made to determine if the object node has been opened. If it has not been opened then a STATUS_ERROR exception is raised. A check is made to determine if the object node is open with intent CONTROL or with intent READ_RELATIONSHIPS. If this is not true then an INTENT_VIOLATION exception is raised. The outgoing relationships of the object node are searched until one or more access relationships are found. Each grant item is extracted from the grant attribute of the access relationship. If the resulting rights list of the grant item is null then the grant item name and the access right are compared. If the two values are equal, then the access relationship is considered complete to the role node. If the resulting rights list is not null, a call is made to IS_IN to determine if the access right is contained in the resulting rights list. If the access right is so contained then IS_GRANTED is called recursively to determine if the necessary right has been approved. Once both checks are successful the access relationship is considered complete to the role node. The procedure IS_ROLE is called to see if the role node is an adopted role of the SYSTEM_CURRENT_PROCESS_NODE (current user). If the search is successful, then the value of true is returned. Otherwise, the value of false is returned. Using the access relationships of Figure 4, IS_GRANTED could be invoked as IS_GRANTED(OBJECT,"READ"). Once the access relationship is found to ROLE 1, the procedure IS_GRANTED is invoked recursively as IS_GRANTED(OBJECT,"READMAIL"). This simple example illustrates the need for at least one level of recursion to evaluate access rights.

A second interface is provided by an overload version of IS_GRANTED. In this version the object node is referred to by its name string. This version opens the object node using its name string to get a node handle. It then calls the first version of IS_GRANTED using the acquired node handle. If the function terminates normally, the object node is closed and the acquired value is returned. If the function call terminates with an exception no value is returned, the object node is closed and the exception is raised again.

3.4 ADOPT

The procedure ADOPT causes the current process to adopt the group role specified by ROLE_NODE. The procedure has two parameters: ROLE_NODE of type NODE_TYPE, which is the group role node handle; ROLE_KEY of type RELATIONSHIP_KEY, which is the key of the ADOPTED_ROLE relationship to be created. An adopted role relationship is created from the calling process to the role node with the relationship key ROLE_KEY. In order for the calling process to adopt the group role, some other adopted role of the calling process must be a potential member of the group.

The ADOPT procedure is designed as follows. A call is made to procedure IS_ROLE to see if the current user is a potential member of the group represented by the role node. If it is not a potential member then a USE_ERROR exception is raised. A check is made to determine if the role node is open. If it is not open a STATUS_ERROR exception is raised. A check is made to determine if the role node is open with intention APPEND_RELATIONSHIPS. If this is not true then a LOCK_ERROR exception is raised. A call is made to the procedure mandatory check to make sure all mandatory access control constraints are met. A new relationship cell with the name "ADOPTED_ROLE" is created with key ROLE_KEY. Finally, the newly created relationship cell is attached to the SYSTEM_CURRENT_PROCESS_NODE, which represents the calling process.

3.5 UNADOPT

The procedure UNADOPT deletes the adopted role relationship with key ROLE_KEY (the only parameter to the procedure). If such a relationship does not exist, the procedure has no effect. The procedure UNADOPT is designed as follows. A check is made to determine if the SYSTEM_CURRENT_PROCESS_NODE is opened with intent WRITE_RELATIONSHIPS. If this is not true then a LOCK_ERROR exception is raised. The ADOPTED_ROLE relationship is accessed using the procedure GET_ACCESS_REL.

The target node of the adopted role relationship is checked to see if it is the top level node. If it is the top level node then a USE_ERROR exception is raised. Finally, the adopted role relationship is detached from the SYSTEM_CURRENT_PROCESS_NODE.

3.6 UNDERLYING ROUTINES

Three underlying routines have been introduced to aid in the implementation of package ACCESS_CONTROL. These routines carry out operations that are implicitly necessary for the functioning of CAIS access control mechanisms. The names of the routines are IS_IN, IS_ROLE, and MANDATORY_CHECK.

3.6.1 IS IN

The routine IS_IN is a simple augmentation to the list management routines. Its function is to check if a given element is found within a given list. It calls the list management routine POSITION_BY_NAME. If POSITION_BY_NAME finds the item in the list it returns a position number. Since the position number is of no consequence IS_IN simply returns true. If the item is not in the list POSITION_BY_NAME raises a SEARCH_ERROR exception. IS_IN handles the exception and returns false.

An overload version of IS_IN is also present. Its function is to check if all the elements of a given list are contained within a second list. It calls the list management routine EXTRACT to extract items from the first list. It calls the first version of IS_IN to determine if each extracted item is in the second list. If all the items of the first list are contained in the second list the function returns true. Otherwise, it rerurns false.

3.6.2 IS ROLE

The routine IS_ROLE is useful in access checking. Its function is to establish if a specified kind of relationship emanates from the SYSTEM_CURRENT_PROCESS_NODE (the calling user) and has a target which is the given node. It searches through all of the relationships emanating from the SYSTEM_CURRENT_PROCESS_NODE. It returns true only if a relationship meeting the above criteria is found. Otherwise, the routine returns false.

3.6.3 MANDATORY CHECK

The routine MANDATORY_CHECK performs the mandatory access control checking. It first checks to make sure that the NODE1 parameter is a process node (since it represents the subject). The procedure derives the subject classification.

The procedure checks the type of the NODE2 parameter and derives its object classification or range of classifications. It checks the intention of the object node (NODE2) to determine which kind of comparisons need to be made between the subject's and object's hierarchical and non-hierarchical classifications. For a read operation, the subject's hierarchical classification must be greater than or equal to that of the object. The subject's set of non-hierarchical catagories must be contained within that of the object. For a write operation the subject's hierarchical classification must be less than or equal to that of the object. The object's set of non-hierarchical catagories must be contained within that of the subject. If any of the mandatory checks are failed a SECURITY_VIOLATION exception is raised.

4.0 IMPLEMENTABILITY ANALYSIS

The access control mechanisms of the CAIS are designed to provide a great deal of flexibility of use while keeping storage demands reasonable. The major expense of this design is the amount of time required to evaluate (check) access rights. Each discretionary access check requires a great deal of searching. Each relationship emanating from the object node must be examined to determine if it is an access relationship. For each access relationship found, each attribute must be examined in order to find the grant attribute. A grant attribute may have several grant items to be examined. Once a desired access right is found within some resulting rights list, a recursive discretionary access check is performed to determine if the necessary right is approved. Finally, when an access relationship has been found to a role node, a search is performed to determine if the role has been adopted by the current process. This search is reasonably straight forward with simple access rights lists. However, if the necessary right is located within a resulting rights list the complexity of the search could be enormous.

The access control mechanisms are not consistently specified throughout the CAIS. The interfaces specified in the package ACCESS_CONTROL have access rights represented by items of type NAME_STRING or lists of these items. The interfaces specified in package NODE_MANAGEMENT have INTENTs specified as arrays of type INTENT_SPECIFICATION. The type NAME_STRING is a subtype of the type STRING and the type INTENT_SPECIFICATION is an enumeration type. As such the two types are not compatable for direct comparison as required by the CAIS specifications. Therefore, it was necessary to develop a mechanism to convert items of type INTENT_SPECIFICATION to type NAME_STRING to enable discretionary access checking to be performed within other packages.

The organization of the CAIS has made it difficult to implement access control mechanisms. According to the specifications the interfaces which provide the primatives for discretionary access checking are to be contained in package ACCESS_CONTROL. As specified package ACCESS_CONTROL must use routines from package NODE_MANAGEMENT. At the same time routines within package NODE_MANAGEMENT are required to perform discretionary acccess checking. It would seem logical to use the interfaces provided in package ACCESS_CONTROL to perform such checking. This is not possible since package ACCESS_CONTROL is dependent on package NODE_MANAGEMENT. As a result of this situation it was necessary to duplicate some of the routines from package ACCESS_CONTROL within package NODE_MANAGEMENT.

An alternate access control mechanism is the use of access control lists. Each object node would have attached to it a list of subjects and the access rights that each subject is granted. While access checking is clearly easier to implement with access lists than with CAIS access relationships, they are deficient in two respects. Access lists require more storage than CAIS access relationships, and access lists are not as flexible to use as CAIS access relationships.

The design of an operational definition of the CAIS has revealed that the specified access control mechanisms are implementable. CAIS access mechanisms provide for a large amount of flexibility at a moderate cost.

5.0 CONCLUSION

The operational definition of access control mechanisms described herein has been designed to meet all specified requirements. The completion of the design has demonstrated the implementability of CAIS access mechanisms. The CAIS access mechanisms sacrifice execution efficiency in favor of flexibility of use and storage economy. Since the CAIS is intended for use within a program development environment, such a sacriface seems quite reasonable. The completion of this operational definition is an important step leading to the development of a useful Common Ada Programming Support Environment Interface Set.

III. AN ABSTRACT MACHINE SPECIFICATION OF THE NODE MODEL

In this section, using text from a Master's Thesis by Margaret D. Little, the operational definition's approach to specifying CAIS Node Model is presented. The implementation approach in this Ada-only representation of the Node Model is a set of highly linked structures for nodes, relationships, and attributes.

Chapter I

INTRODUCTION

In 1975 the Department of Defense (DoD) established the High Order Language Working Group to "produce a minimal number of common, modern High Order Computer Languages for the DoD embedded computer systems applications and to assure a unified, well supported, widely available, and powerful programming support environment for these languages." From this project came the STEELMAN language requirements document and the Ada[1] language. Also from this project came the STONEMAN document outlining requirements for an Ada Programming Support Environment (APSE).

An APSE is a complete programming environment whose purpose is "to support the development and maintenance of Ada applications software throughout its life cycle" [6]. To fulfill this objective, an APSE provides a coordinated set of software tools intended to facilitate project management and long term maintenance as well as program development.

A major goal in the design of an APSE is to facilitate the transfer of APSE tools and tool sets between machines. The layered structure of an APSE outlined in STONEMAN is

[1] Ada is a registered trademark of the U.S. Department of Defense.

designed to allow source-level transportability of software tools written in Ada. The structure has four levels, as follows:

level 0: Hardware and host software as appropriate

level 1: Kernel Ada Program Support Environment (KAPSE), which provides database, communication and run-time support functions to enable the execution of an Ada program (including a MAPSE tool) and which presents a machine-independent portability interface.

level 2: Minimal Ada Program Support Environment (MAPSE) which provides a minimal set of tools, written in Ada and supported by the KAPSE, which are both necessary and sufficient for the development and continuing support of Ada programs.

level 3: Ada Program Support Environments (APSEs) which are constructed by extensions of the MAPSE to provide fuller support of particular applications or methodologies. [6]

Software tools written in Ada that use only the machine-independent interface provided by the Kernel APSE (KAPSE) to communicate with the underlying machine may be transported to other machines that support the same KAPSE interface.

The potential introduction of several different APSEs, with correspondingly different KAPSEs, lead to the formation of the KAPSE Interface Team (KIT) and the Kernel APSE Interface Team from Industry and Academia (KITIA) to define standard KAPSE interface conventions. The KIT and KITIA have developed a set of kernel interfaces called the Common APSE Interface Set (CAIS), described in the _Draft Specification of the Common APSE Interface Set (CAIS)_, hereafter referred to as the CAIS Specification.[2] The CAIS is a collection of Ada packages; when implemented, the subprograms in these packages can be used by APSE tools to provide system services.

To ensure uniform implementations of the CAIS, a suite of validation tests, the CAIS Validation Capability, is being developed through the APSE Evaluation and Validation Team [2]. To develop validation tests a complete, precise specification of the CAIS is needed. An important part of such a specification is the definition of the function, or semantics, of each subprogram of the CAIS.

The interface semantics are not well-defined in the current CAIS specification. The specification describes the function of each subprogram using commentary; these descriptions are incomplete and contain inconsistencies. Consequently, implementations based on this specification are not likely to be uniform. A specification that overcomes these problems is needed.

---

[2] A subsequent version of this specification, [4], has been proposed as a Government standard and is currently in the review process.

The objective of this paper is to identify a semantic specification technique that is easily understandable, yet lends itself to a complete and consistent description of the Common APSE Interface Set, and to use this technique to develop a new specification for the CAIS node model.

This paper develops specifications for the Ada packages outlined in [1] that constitute the CAIS node model. These packages are CAIS_NODE_DEFS, CAIS_NODE_MANAGEMENT, and CAIS_ATTRIBUTES. In addition, a specification for the CAIS_STRUCTURAL_NODES package, which is not a general node package, is given. Since the CAIS specification available for this project [1] did not thoroughly define access control for the node model, the CAIS_NODE_CONTROL package described therein is omitted from this study.

The following terms are used in this paper:

specification: a document that defines requirements, details a design, or describes a product (IEEE);

transportability: "the degree to which a [APSE] tool can be installed on a different APSE without reprogramming; the tool must perform with the same functionality in both APSEs. Synonyms commonly used are portability and transferability." [3];

facility: a function implemented in the kernel APSE interface: the means of interaction between a software tool and a facility [15];

syntax: the collection of rules which indicate whether a string of characters is a valid use of a kernel interface;

semantics: the rules that give meaning to a kernel interface, used synonymously with function and functionality.

Chapter 2 of this paper presents a review of proposed techniques for the semantic specification of kernel interfaces and applications of these techniques; Chapter 3 details the specification technique used in this study. Chapter 4 provides a preliminary description of the CAIS node model, and Chapter 5 reflects the details of the application of the specification technique to the node model. Chapters 6, 7, and 8 present the specifications; and Chapter 9 reports summary conclusions and suggests areas for further study.

Chapter II

SEMANTIC SPECIFICATION TECHNIQUES

This chapter presents a review of the literature pertaining to the semantic specification of kernel APSE interfaces.

A report[3] by Kafura and others [15] states that a complete specification of kernel APSE facilities must include four elements:

1. syntax,
2. semantics,
3. limits: details of any structural or usage constraints placed on a facility, and
4. hidden protocols: interactions between facilities, by means of direct communication or shared data, that may not be visible.

The report points out that, of the four parts of the specification, the semantics are the most difficult to define and, at the same time, the most important.

The problems with semantic specifications are not new. The specification of the semantics of programming languages, the rules that give meaning to programs, is a topic that has been extensively researched.

---

[3] This material is also covered in a published article [16].

Aho and Ullman [7] discuss some of the approaches to specifying programming language semantics. One such approach is _axiomatic definition_ [13] [7]. In this approach axioms and rules of inference are developed to define the meaning of the language. These axioms and rules are used in proving properties of programs written in the language. Another approach to semantic description of languages is _denotational_ (or _mathematical_) _semantics_ [7]. In a denotational semantic specification, rules are defined that translate programs into abstract mathematical objects.

Another approach described by Aho and Ullman is _operational_ (or _interpretive_) _semantics_. The premise behind this method is that the semantics of a machine language can be defined by the machine itself; "a machine language program 'means' exactly what the computer does when the program is run." The semantics of a high level language cannot be defined in this way, however, since a high level language program cannot be run on a computer without first being compiled into some machine language, which itself requires a semantic definition of the language. Aho and Ullman give the following description:

> The interpretive [or operational] approach to defining the semantics of programming languages is to postulate an abstract machine and provide rules for executing programs on this abstract machine. These rules then define the meaning of programs the way the real machine did for assembly language programs. Usually, the abstract machine is characterized by a _state_, consisting

of all data objects, their values, and the program with its program counter (indicator of the 'current' step in the program). The semantic rules specify how the state is transformed by the various programming language constructs [7].

Each of these approaches is the basis for a semantic specification method for kernel APSE interfaces.

The report by Kafura and others [15] outlines four general methods for specifying the semantics of kernel APSE facilities:

1. natural language specification,
2. formal methods,
3. by example, and
4. abstract machine specification.

A natural language specification is an informal description, by means of commentary, of the intended function. Specifications of this type are relatively easy to construct and very readable; they are, however, quite difficult to check for completeness and absence of ambiguity.

The report [15] suggests several formal methods that might be used to specify the function of kernel facilities. These methods include axiomatic specification and denotation semantics. Formal methods produce precise specifications that can be mathematically analyzed. However, they are difficult to construct and not easy to comprehend.

Another method of semantic specification suggested in the report [15] is by example. This method requires the construction of a set of validation tests. By definition, then, any implementation that produces the output expected by the validation tests is functionally correct. Specifications of this type would have to be extremely well-documented to be useful to implementors, which would again lead to a natural language specification, albeit a more thorough one. The quality of a specification of this type depends entirely on the techniques used to develop the validation tests.

A final method for specifying the function of kernel facilities, described in the report [15], is abstract machine specification. This approach is based on the operational semantics described above. An abstract machine is detailed that contains primitive objects representing the elements in the kernel environment. Programs written for the machine that perform the intended function of a facility then provide a semantic specification of the facility. A specification of this type is more precise than a natural language specification, yet easier to understand than a formal specification. The main difficulty with this method pointed out in the report is that the semantics of the language of the abstract machine must be defined. The report suggests that this difficulty can be eased by using Ada as the programming language.

Freedman brings up some issues involved in developing a formal specification for a standard kernel APSE interface [11]. He discusses the relative merits of an operational approach (using an abstract machine) versus a denotational approach, and points out that an operational specification is built using a bottom-up design technique, a relative disadvantage; the machine must be defined before programs can be written and their semantics defined. Another disadvantage pointed out to this approach is that it is implementation-dependent; the specification is based entirely on the definition of the abstract machine.

The abstract machine approach to specifying kernel APSE interface components is examined further by Lindquist and others in [16]. This article points out that, while an abstract machine specification may seem to require a specific implementation, the abstract machine programs are intended to describe semantics solely by their (perceived) execution effects. Thus, any implementation that generates identical output defines the same semantics.

The article [16] goes on to specify an abstract machine and illustrate its use through an abstract machine program for one procedure of the CAIS_PROCESS_CONTROL package. This abstract machine is defined in terms of three components: a processor, a storage, and an instruction set. The basis of the instruction set is the Ada language, chosen because of its well-defined semantics and its familiarity to the intended reader. The example program for the abstract machine clearly identifies both valid inputs and error conditions, and the error handling techniques are explicit. These are points that are incomplete in the natural language description of the CAIS given in [1]. The Ada based abstract machine described in this report provides a readable and well-defined semantic description for the example given.

The abstract machine approach to specifying the CAIS has been further analyzed in a paper by Facemire [9], in which a validation mechanism based on an abstract machine specification is developed. Another paper, by Srivastava [17], has successfully applied an abstract machine approach in developing semantic specifications for the process node packages of the CAIS.

As indicated by this review, the abstract machine approach to specifying the semantics of the CAIS looks promising. It leads to a more complete and precise semantic specification than the natural language approach used in the current specifications, and the specifications are easier to construct and more readable than formal mathematical methods. In addition, it forms a specification on which validation mechanisms can be based. This paper focuses on applying this approach to the node model packages of the CAIS to develop a semantic specification that is complete, consistent and unambiguous.

Chapter III

THE ABSTRACT MACHINE METHOD OF SPECIFYING CAIS FUNCTION

The method used in this paper to specify the function of CAIS operations is an abstract machine approach based on operational semantics. As proposed in [15] and [16], and illustrated in [9] and [17], this method can be used to specify the function of the kernel APSE interfaces. To apply this approach, an abstract machine must be defined. The meaning of any program written for the abstract machine is then defined by the execution rules of the machine. In particular, the meaning of the CAIS operations can be defined by giving programs that perform the intended function of the operations on the abstract machine.

3.1 THE ABSTRACT MACHINE

As seen in the previous chapter, to write a program for an abstract machine that describes the function of a CAIS operation, a complete description of the abstract machine must be given. The design of a complete abstract machine with its states and execution rules could be a major job. An ideal solution to this problem is to use the Ada language to describe the function of the CAIS operations. This language has a thorough semantic specification on which to base the abstract machine. In addition, since it should be familiar to those readers interested in kernel APSE interfaces, the resulting specifications, programs written in Ada, will be easily comprehensible.

The abstract machine used in this report is a high level language machine based on the Ada language. This machine accepts Ada statements as its instruction set; programs written for the machine are written in Ada.

The specification of the Ada language, given in the language reference manual [5], provides the execution rules for the abstract machine. These rules clearly define the results of the execution of any instruction on the machine.

The data objects for the machine can be defined by each program in the Ada language using Ada type and object declarations. Thus, each Ada program can define the initial state of the storage of the machine and can modify this storage during execution.

State characterizations of the abstract machine during execution of a program are not given here. It is assumed that the reader is familiar with the Ada language or some similar high level language. The language reference manual [5] should be used as a reference where needed by the reader.

3.2 PROGRAMS FOR THE ABSTRACT MACHINE

To describe the CAIS node model, this paper develops the bodies of several packages of the CAIS. The CAIS Specification gives the Ada package specifications of the packages that form the node model. These package specifications contain type and object declarations and subprogram specifications for the node model operations. This paper develops the bodies of the subprograms to describe the node model operations. The subprogram bodies form part of the package bodies presented in this paper. Together with the Ada package specifications, the package bodies provide a complete description of the semantics, as well as the syntax, of the CAIS node model.

3.3 SUMMARY

This chapter details the abstract machine approach used in this paper to specify the function of CAIS operations. The abstract machine itself is a high-level language machine based on the Ada language; therefore, programs written for the machine are written in the Ada language. The operations of the CAIS node model are described by developing an Ada package body corresponding to the Ada package specification given for each package comprising the node model in the CAIS Specification [1].

Chapter IV

THE CAIS NODE MODEL

This chapter provides an overview of the elements and structure of the CAIS node model, as described in the CAIS Specification, and provides a basis for the discussion of implementation techniques in the following chapter. The node model provides the basic foundation upon which the remainder of the CAIS is built. The intent of the CAIS is to provide standard interfaces to traditional operating system services. Thus, the model must provide some representation for typical operating system entities, such as files, processes, and devices.

The node model has three basic elements, nodes, relationship, and attributes. A _node_ holds information about an entity. A _relationship_ indicates a logical connection between two entities. An _attribute_ represents a property of an entity or a connection between entities.

The structure of the model is a directed graph. The nodes of the model are the vertices of the graph, and the relationships are the directed edges.

4.1 NODES

A node has properties that depend on the entity it represents. version 1.1 of the CAIS defines four kinds of nodes: structural nodes, file nodes, process nodes, and device nodes. This node kind is a property of the node.

Nodes also have content that varies according to the kind of node. A structural node has no content, it holds relationships and attributes. The content of a file node is a data file on an external storage device. The content of a process node is some representation of the execution of a program. The content of a device node is a representation of a logical or a physical device.

All nodes can hold relationships and attributes. Nodes are not named; they are accessed through the named relationships connecting them.

4.2 RELATIONSHIPS

Relationships can be viewed as the directed edges that connect the nodes in the directed graph structure of the model. A relationship emanates from one node called the source node, and terminates at another node, the target node. A relationship may only be accessed from its source node.

Because entities may have many different types of logical connections with one another, relationships are categorized. A relationship type is called a relation. Relations group relationships according to their characteristics. There are several predefined relations that serve special purposes.

A relationship is an instance of a relation, and is identified by a relation name and a relationship key. The relation name identifies the relation. The relationship key identifies one of many relationships the source node may hold of the relation.

4.3 ATTRIBUTES

Nodes and relationships can have one or more attributes associated with them. Attributes represent a property of the node or relationship.

An attribute of a node or a relationship is identified by an attribute name. Each attribute has a list of values; the values are represented by a LIST type as described in the CAIS_LIST_UTILS utility package of the CAIS Specification [1].

4.4 STRUCTURE

Relationships are further classified as primary or secondary. The primary relationships form strict tree structures within the generalized directed graph structure of the model.

The nodes of the model make up a forest of trees. The root of a tree is called a top-level node. Each particular tree structure can be thought of as belonging to a user of the APSE. The top-level node of the tree represents the user.

Each node has one primary relationship to it from another node, its parent node. Each node, except a top-level node, also has a secondary relationship emanating from it to its parent, acting as a back pointer. Thus, a tree can be traversed in either direction, up or down.

4.5 PATHS

A path is a sequence of relationships. Since nodes are not named, they are accessed by traversing a sequence of relationships, which are named. An absolute path is any path beginning at a top-level node. A relative path is a path that begins at any convenient node (not necessarily a top-level node). A primary path is a path consisting of only primary relationships. Since these primary relationships follow strict tree structures, each node has one unique absolute primary path (that is, a primary path beginning at a top-level node).

A pathname is a sequence of relationship names indicating a path to a node. A pathname can be broken up into path elements, each of which represents one relationship in the sequence. The syntax of a path element is an apostrophe ('), called a tick, followed by a relation name, followed by a relationship key within parentheses, for example: 'DOT(CONTROLLER). A pathname consists of a sequence of these path elements.

The path element of the previous example shows an occurrence of a predefined relation, DOT. Instances of the DOT relation in a path element can be abbreviated by a dot (.), followed by the relationship key, for example: .CONTROLLER.

A process uses a pathname to identify nodes when making a call to a CAIS interface. In a CAIS implementation, this process will be associated with a process node. This process node, called the current process node, is used as the starting node in interpreting and traversing a path.

4.6 PREDEFINED RELATIONS

The CAIS Specification gives some predefined relations that have special meaning. The relation DOT discussed in the previous section is a predefined relation. It is used as a default relation. Instances of the DOT relation can be abbreviated in pathnames producing simple and easily readable pathnames.

The PARENT relation is another predefined relation. Each node has a secondary relationship to its parent node. This relationship is an instance of the PARENT relation. PARENT relationships are managed automatically by the CAIS in maintaining the primary tree structures.

A few predefined relations are provided to give process nodes some basic access channels for connections with other nodes. The relation USER allows access to top-level nodes, the roots of the tree structures. A process node can access any top-level node using the predefined relation USER and a relationship key that identifies a user of the APSE.

A user can have one or more process node trees, called a job, at any time. A relationship of the predefined JOB relation connects a root process node to its user's top-level node. Each process node is a node of one process tree, which is a subtree of one user's tree.

Three special relations are provided to allow easy access to certain nodes from a process node. The CURRENT_JOB relationship of a process node indicates the root node of its process tree. The CURRENT_USER relationship indicates the user's top-level node. The CURRENT_NODE relationship points to some known node (often a structural node) used in specifying relative paths.

4.7 NODE HANDLES

Node handles are objects associated with a process that allow easy identification of nodes. They provide direct access to a node, bypassing the relationship traversals necessary when a pathname is used to identify a node.

4.8 OPERATIONS

The details of the operations of the node model are presented in package descriptions of the CAIS Specification. The CAIS_NODE_DEFS package provides definitions of data types and objects. No operations are given in this package. The package CAIS_NODE_MANAGEMENT defines the basic operations on nodes and relationships. It gives operations to copy and delete nodes, create and delete secondary relationships, change primary relationships, use node handles, and obtain information about existing nodes. The

CAIS_ATTRIBUTES package defines some data types and operations for node and relationship attributes.

The package CAIS_NODE_CONTROL presented in version 1.1 of the CAIS Specification [1] defines some node access synchronization primitives. This package is incomplete, and its semantics are not clearly enough described to appear useful. It is anticipated that this package will either be completely rewritten, or deleted altogether. CAIS_NODE_CONTROL will not be further described in this report.

In addition to these basic packages comprising the node model of the CAIS, the package CAIS_STRUCTURAL_NODES is included in this paper. The operations to create nodes are included in the packages dealing with a particular kind of node. The CAIS_STRUCTURAL_NODES package defines the operations to create structural nodes.

4.9 SUMMARY

This chapter provides a basic overview of the CAIS node model. Nodes, relationships, and attributes are introduced, and the basic structure of the model is illustrated. The use of pathnames, which represent sequences of relationships, to identify nodes is described, and node handles, which allow short-cut access to nodes, are introduced. The packages of the CAIS containing the node model operations are briefly discussed.

Chapter V

THE IMPLEMENTATION TECHNIQUES USED IN THE ABSTRACT MACHINE PROGRAMS

This chapter outlines the implementation techniques used in developing the abstract machine programs that describe the CAIS Node Model. To write programs that manipulate nodes and relationships, some storage representations are needed. An implementation package, CAIS_PRIVATE_DEFS, defines data types and data objects used in the abstract machine programs. In addition, to allow the development of more readable programs, CAIS_PRIVATE_DEFS defines some simple operations to manipulate these data types. This chapter presents an overview of the concepts used in defining the storage structures. The details of the CAIS_PRIVATE_DEFS package appear in Appendix A.

The package CAIS_NODE_DEFS, defined in the CAIS Specification [1], also defines some data types and data objects used in the abstract machine programs. Parts of this package are discussed here; the details are in Appendix B.

5.1 AN ADJACENCY LIST REPRESENTATION OF THE NODE MODEL

As seen in Chapter 2, the structure of the node model is a directed graph. The node model, however, does not enforce two restrictions placed on formal graph structures [14]. One of these restrictions requires that a vertex not have an edge connecting it to itself; the other restriction requires that two vertices not have multiple edges connecting themselves. Consequently, a representation technique for the node model must be chosen that can be modified to include these characteristics not generally held by directed graphs.

Adjacency list representation, detailed in Horowitz and Sahni's book [14], is a particularly good representation technique for a directed graph. Using this representation method, each vertex of a graph has a linked list associated with it. The cells of this linked list can be considered to represent the directed edges (arcs) out of the vertex. Each cell of the linked list must contain two fields; VERTEX indicates which vertex the arc points to, and LINK connects the linked list by pointing to the next list cell. An adjacency list representation of the graph in figure 1A is shown in figure 2A.

The nodes and relationships of the CAIS node model can be represented in the same way using adjacency lists, see

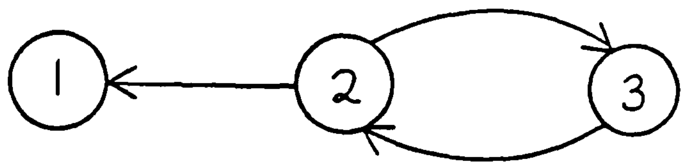


Figure G-1A: **A Directed Graph**

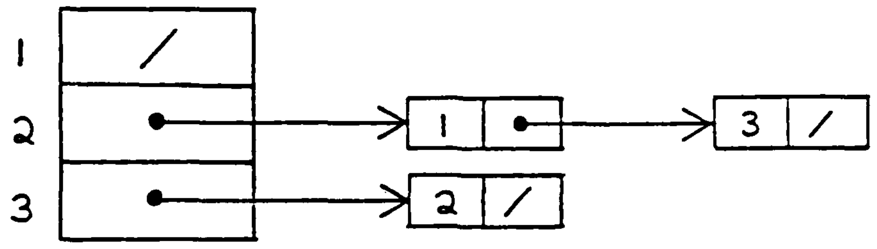


Figure G-2A: **An Adjacency List Representation of a Directed Graph** figures 3A and 4A. Each node has a linked list of relationships associated with it allowing access to other nodes. The node field ARC_OUT is the head of the linked list of relationships the node holds with other nodes. The relationship field TO_NODE indicates the target node of the relationship, and the field LINK connects the list of relationships of the source node.

5.2 PRIMARY RELATIONSHIPS

As seen in Chapter 2, the primary relationships of the node model form tree structures; the user top-level nodes are the roots of the trees. In this implementation, these tree structures are maintained within the directed graph representation. Both primary and secondary relationship records contain a field PRIMARY, which is true, for a primary relationship, or false, for a secondary relationship. Primary and secondary relationship records are linked together in the arc out list of a node.

The primary relationships in this implementation form a single tree structure. A system node called SYSTEM_ROOT, defined in CAIS_PRIVATE_DEFS, is the root of the tree. All relationships held by this node are primary USER relationships that access the top-level nodes. These USER relationships are the only primary relationships not having

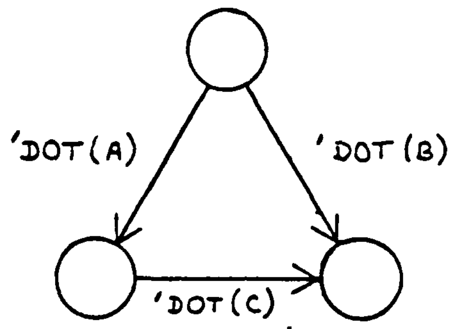
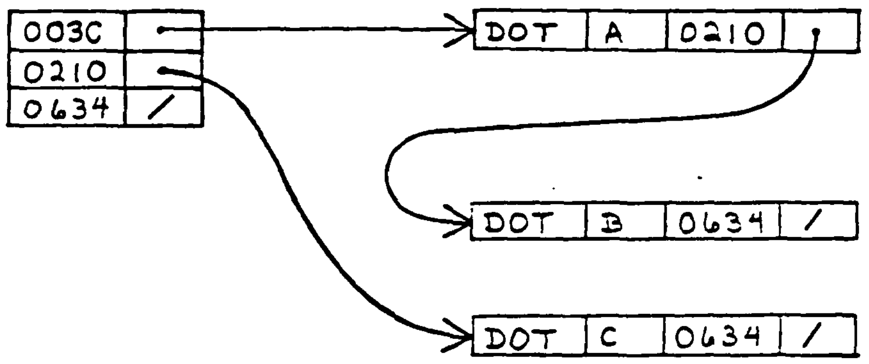


Figure G-3A: **A Node Model Example**



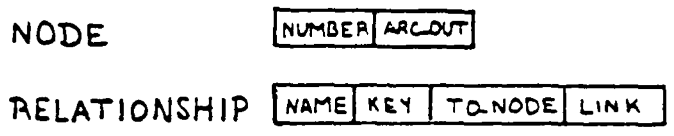


Figure G-4A: **A Representation of the Node Model Example** a corresponding secondary predefined PARENT relationship from the target node to the source node of the primary relationship that simulates bi-directional access between parent and child nodes of the tree.

5.3 THE ACCESS CONTROL TABLE

The possible existence of dangling references to deleted nodes makes a detection method necessary. The CAIS Specification states, "If a node is deleted, (i.e., its primary relationship is broken), outstanding secondary relationships for which it is the target may remain, but attempts to access the node via these relationships will raise an exception" [1]. An efficient method for detecting both dangling secondary relationships and open node handles to deleted nodes is needed so the appropriate exception can be raised.

The method used here adds a layer of indirection using node sequence numbers and an access control table to detect deleted nodes. A unique sequence number, supplied by the package SEQNUM_MANAGER (see Appendix A), is assigned to each node when it is created. The number, along with a pointer to the node record, is stored in the access control table. The entry in the table provides the only access to the node; relationships and node handles contain only the sequence number of the target node and must go through the table to obtain access to the node record itself. When the node is deleted, its entry in the access control table is deleted, and since these sequence numbers are unique and never reused, any attempted access using the old sequence number fails. A representation using an access control table of the node model example in figure 3 is shown in figure 5A.

Assigning each node a unique sequence number is not an impracticable implementation technique. Assume that, on the average, a new node is created and a new sequence number assigned every millisecond. Using a sixty-four bit sequence number, it would take 2^{64} milliseconds, or approximately 585 million years, to use up all possible unique numbers.[4]

5.4 TYPE SPECIFICATIONS

The type specifications that reflect the implementation methods are now presented. The Ada definitions of nodes, relationships, attributes, and the access control table are the basic objects manipulated by the abstract machine programs.

[4] 2^{64} ms = $1.84 * 10^{19}$ ms = $1.84 * 10^{16}$ s = $5.12 * 10^{12}$ hr
= $2.14 * 10^{11}$ day = $5.85 * 10^{8}$ yr

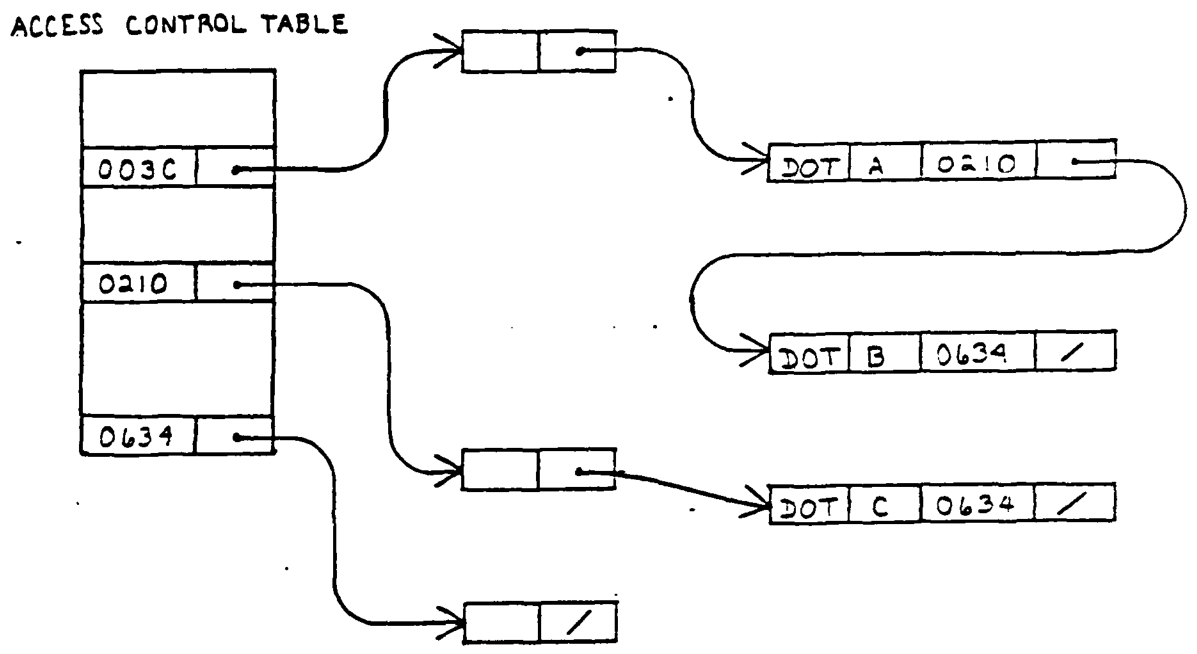


Figure G-5A: A Representation Using an Access Control Table

5.4.1 Access Types

The implementation defines nodes and relationships as dynamically created storage structures. Thus, when a node record or relationship record is created, it is accessed through a pointer to the record, an Ada access type. The following incomplete record specifications and access type specifications are defined in CAIS_PRIVATE_DEFS.

```
    type RELATIONSHIP (PRIMARY : boolean  := false);
    type ACCESS_REL is access RELATIONSHIP;

    type NODE (KIND : CAIS_NODE_DEFS.NODE_KIND);
    type ACCESS_NODE is access NODE;
```

5.4.2 A Node Record

The type specification of a node appears in CAIS_PRIVATE_DEFS.

```
    type NODE (KIND : CAIS_NODE_DEFS.NODE_KIND) is
        record
            NUMBER          :  NODE_SEQUENCE_NUMBER;
            ARC_OUT         :  ACCESS_REL;
            ATTRIBUTE       :  ACCESS_ATTRIBUTE;
            case KIND is
                when STRUCTURAL =>
                     null;
            end case;
        end record;
```

The KIND field of this variant record indicates whether this is a file, structural, process, or device node; the content of the node depends on this field. A structural node has no content, as shown in the type specification. The specification of the contents of other types of nodes is beyond the scope of this paper.

The NUMBER field of the node record holds a NODE_SEQUENCE_NUMBER that serves as a unique identifier of the node. The ARC_OUT field holds a pointer to the first relationship in the linked list of relationships held by the node. The ATTRIBUTE field allows access to the attributes of the node; the implementation of attributes is discussed later in this chapter.

5.4.3 A Relationship Record

The type specification of a relationship record, below, is taken from the CAIS_PRIVATE_DEFS package.

```
type RELATIONSHIP (PRIMARY : boolean  := false) is
    record
        RELATION      :  CAIS_NODE_DEFS.RELATION_NAME;
        KEY           :  CAIS_NODE_DEFS.RELATIONSHIP_KEY;
        TO_NODE       :  NODE_SEQUENCE_NUMBER;
        ATTRIBUTE     :  ACCESS_ATTRIBUTE;
        LINK          :  ACCESS_REL;
        case PRIMARY is
            when true  =>
                null;
            when false =>
                PREDEFINED  :  boolean := false;
        end case;
    end record;
```

The PRIMARY field of this variant record distinguishes a primary relationship from a secondary relationship. The PREDEFINED field of a secondary relationship identifies those predefined secondary relationships, for example, PARENT, that the user cannot modify. The TO_NODE field identifies the target node of the relationship. The LINK field connects the linked list of relationships of the source node and contains a pointer to the next relationship in the linked list.

5.4.4 A Node Handle

A node handle is a NODE_TYPE variable held by a user process that allows efficient access to a node. Many of the CAIS operations require a node handle to identify a particular node.

The CAIS Specification defines NODE_TYPE as a limited private type in the CAIS_NODE_DEFS package. Consequently, outside the CAIS_NODE_DEFS package, the details of the definition of the type are not visible, and the assignment operation and tests for equality and inequality are not available. Because the free use of the NODE_TYPE data type is required to detail the abstract machine programs for routines in other packages, NODE_TYPE has not been defined as limited private in this paper. Some other protection mechanism is assumed to exist to prevent users of the CAIS from directly examining or changing the contents of a NODE_TYPE variable.

The abstract machine programs in this paper define a node handle as a pointer to a relationship record. The detailed type specification given here appears in the CAIS_NODE_DEFS package.

```
type NODE_TYPE is new ACCESS_REL;
```

Since this is a derived type, the abstract machine programs can make explicit type conversions to assign a pointer to a relationship record into a NODE_TYPE variable. The relationship record contains the node number of its target node, and thus the node record is easily accessed using the node handle. Additionally, all information that must be carried by a NODE_TYPE variable detailing how the node was originally accessed is directly available.

5.5 PRIMITIVE OPERATIONS ON NODES AND RELATIONSHIPS

This section provides an overview of some basic operations on node and relationship structures used in the abstract machine programs. The purpose of using such operations is to avoid complicated code in the abstract machine programs that makes the programs difficult to read. The operations are described in more detail in the CAIS_PRIVATE_DEFS package in Appendix A.

The functions GET_ACCESS_NODE and EXISTS perform similar duties. GET_ACCESS_NODE takes a node number and searches the access control table for the pointer to the node record. If the number is not found, GET_ACCESS_NODE raises an exception, NONEXISTENT_NODE. A similar function, EXISTS, takes a node number and searches the access control table. If a control record is found, the node exists, and the function returns true, otherwise, it returns false.

CREATE_NODE_CELL creates a node record and makes an appropriate entry in the access control table. DELETE_NODE_CELL deletes a node's entry from the access control table without actually deleting the node record, making the node inaccessible.

The functions GET_ACCESS_REL and IS_RELATIONSHIP search the list of relationship records of a specified node for a relationship with a specified relation name and relationship key. GET_ACCESS_REL returns a pointer to the relationship, or, if the relationship is not found, raises an exception called RELATIONSHIP_NOT_FOUND. IS_RELATIONSHIP returns true if the relationship is found, false if it is not found.

GET_ACCESS_PRIMARY_REL requires two parameters, a pointer to a parent node and a pointer to a child node. The function searches the relationship list of the parent node for the primary relationship to the child node, and returns a pointer to this relationship.

CREATE_RELATIONSHIP_CELL creates and initializes a new relationship record, and ATTACH links the record into the relationship list of the specified source node. All relationships are linked in ASCII lexicographical order by relation name and then by relationship key. DETACH unlinks a relationship record from the relationship list of a specified source node, effectively deleting the relationship.

5.6 PATHNAMES

When a pathname is supplied as an argument to a CAIS operation in the form of a NAME_STRING variable, the pathname must be analyzed and the nodes and relationships in the path traversed to access the indicated node. The package CAIS_PRIVATE_DEFS defines operations to perform these functions.

The procedure PARSE lexically analyzes the valid NAME_STRING argument passed in and builds a list of (relation name, relationship key) tuples. TRAVERSE uses this list of tuples to navigate through the specified node and relationship records, and returns a pointer to the last relationship record encountered. The target node is not accessed, thus it is possible to access a dangling secondary relationship using this operation. PARSE raises an exception, SYNTACTIC_ERROR, if the NAME_STRING is syntactically invalid, and TRAVERSE raises TRAVERSE_EXCEPTION if it encounters a deleted node, or if a specified relationship is not found.

CAIS_PRIVATE_DEFS defines a data object called SYSTEM_CURRENT_NODE. This ACCESS_NODE variable always points to the process node that represents the process currently executing. The procedure TRAVERSE uses this node as a starting point in traversing the sequence of node and relationship records in a path.

Several CAIS operations defined in [1] use a NAME_STRING argument to specify a relationship that is to be created. In this case, the first part of the path must be traversed to find a node, but the last path element of the path name refers to the relationship that must be created. The procedure DISSECT, defined in the CAIS_PRIVATE_DEFS package, picks off the last relation name and relationship key pair and returns these, together with the path name to the base node.

5.7 ATTRIBUTES

Nodes and relationships, as seen in Chapter 2, can have one or more associated attributes. Each attribute is identified by a name and has a list of values.

The abstract machine programs developed in this paper use a singly linked list representation for attributes. Each node and relationship is the head of a linked list of attributes, and each attribute is a cell of a linked list. Each node and relationship record contains an ATTRIBUTE field that points to the first attribute in the list, and each attribute record contains a LINK field that points to the next attribute in the list.

The type specification of an attribute record, given in CAIS_PRIVATE_DEFS, reflects this linked list implementation method.

```
type ATTRIBUTE is
    record
        NAME           :  ATTRIB_NAME;
        VALUE          :  LIST;
        READ_ONLY      :  boolean  := false;
        INHERIT        :  boolean  := false;
        NEXT           :  ACCESS_ATTRIBUTE;
    end record;
```

The NAME field holds the name by which the attribute is identified. READ_ONLY and INHERIT are flags associated with each attribute as defined in section 3.6.2 of [1].

The VALUE field of an attribute is a variable of type LIST_TYPE. The CAIS Specification defines this type in a package called CAIS_LIST_UTILS. A LIST_TYPE is a limited private type and can only be operated on by the routines in the CAIS_LIST_UTILS package, described in [1].

The operations that manipulate the attribute structures used in this implementation are outlined here and described in more detail in Appendix A. The functions IS_ATTRIBUTE and GET_ACCESS_ATTRIBUTE search a linked list of attribute records for the attribute with the given name. GET_ACCESS_ATTRIBUTE returns a pointer to the attribute record. IS_ATTRIBUTE returns true if the attribute is found, false if it is not found. CREATE_ATTRIBUTE creates a new attribute record with a specified attribute name, and ATTACH_ATTRIBUTE attaches it into the linked list of a node record or a relationship record in ASCII lexicographical order by attribute name. DETACH_ATTRIBUTE deletes an attribute by unlinking the attribute record from the node or relationship attribute list.

5.8 SUMMARY

This chapter describes the storage structures used in developing the abstract machine programs that define the semantics of the operations of the CAIS node model. It outlines the representation chosen for the node model, which is based on adjacency list representation, and develops a method for detecting deleted nodes using an access control table. The chapter details the objects of the abstract machine, nodes, relationships, and attributes, and outlines basic operations that manipulate these objects.

AFIT/GCS/MA/85D-6

USING ADA IN THE REAL-TIME AVIONICS ENVIRONMENT: ISSUES AND CONCLUSIONS

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

Donald J. Witt, M.A., B.S.

Captain, USAF

December 1985

Contents

Acknowledgements

The project described in this thesis was a natural offspring of the Air Force Wright Aeronautical Laboratories' (AFWAL) Evaluation & Validation (E & V) team's efforts. They had tasked the Institute for Defense Analyses to develop a Prototype Ada Compiler Evaluation Capability (ACEC) and Virginia Castor, who was then chief of the E & V team, suggested that someone needed to determine if the concerns of the real-time avionics environment were addressed by the ACEC. Further discussions with Ms. Castor and with my thesis advisor and AFIT instructor, Capt Patricia Lawlis, helped shape the thesis proposal.

The project changed as time went on, and the task of searching other sources for applicable tests was added. In addition, one of the original goals, compiling the applicable tests on an Ada to MIL-STD-1750A compiler and running them on a MIL-STD-1750A machine, had to be modified because the compiler couldn't be obtained in time.

I think the results of the project are useful and will become even more useful in the near future as more MIL-STD-1750A compilers are developed. The tests identified in the project should help prospective users of Ada evaluate different compilers and run-time systems. The builders of the ACEC would also benefit from looking at the issues identified in the project.

I would like to thank my thesis advisor, Capt Lawlis, for her guidance and advice. Mr. Ray Szymanski, who replaced

Ms. Castor when she transfered to the Ada Joint Program Office, deserves special thanks for providing invaluable assistance whenever I requested it. Lts Long and Wood of AFWAL were also very helpful as was SSgt Swem of SeaFac. I am extremely grateful to Colonel Walter Figel, Jr., for his assistance, encouragement, and advice.

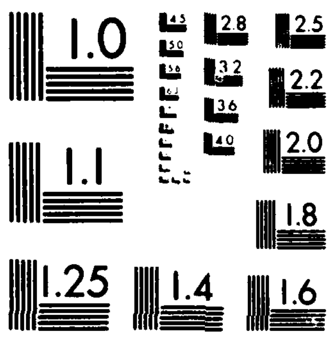
I reserve my deepest gratitude for my family. My wife, Carolyn, and my thirteen year old daughter, Kimberly, have truly sustained me throughout this trying period. They always exhibited patience when I couldn't be with them or had to cancel our plans because the best time to access the computer was on the weekends. They were extremely tolerant of my moods, sleepless nights, and frustrations while they continually voiced encouragement and support. I couldn't have done it without them.

List of Figures

List of Acronyms

ACVC	Ada Compiler Validation Capability
ACEC	Ada Compiler Evaluation Capability
ADE	Ada Development Environment
AFWAL	Air Force Wright Aeronautical Laboratories
AJPO	Ada Joint Programming Office
ANSI	American National Standards Institute
ASPE	Ada Programming Support Environment
AVO	Ada Validation Organization
BMAC	Boeing Military Aircraft Company
CAIS	Common ASPE Interface Set
CPU	Central Processor Unit
DoD	Department of Defense
E & V	Evaluation and Validation
HOL	High Order Language
HOLWG	High Order Language Working Group
IDA	Institute for Defense Analyses
ISA	Instruction Set Architecture
KB	KiloByte
LRM	Language Reference Manual
MCCS	Mission Critical Computer Systems
RTS	Run-time Support
STARS	Software Technology for Adaptable Reliable Systems
VDU	Visual Display Unit

AD-A172 343 EVALUATION AND VALIDATION (E&V) TEAM PUBLIC REPORT 4/5
VOLUME 2(U) AIR FORCE WRIGHT AERONAUTICAL LABS
WRIGHT-PATTERSON AFB OH R SZYMANSKI 30 NOV 85
UNCLASSIFIED AFWAL-TR-85-1016-VOL-2 F/G 14/2 NL


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0

Abstract

This project involved studying the real-time avionics environment in which Ada will become the primary programming language in the near future. A set of issues of concern regarding the use of Ada within this environment was identified and described. Test programs, including the new Prototype Ada Compiler Evaluation Capability, were evaluated as to their applicability to these issues. The applicable test programs were compiled and executed using two validated Ada compilers. Compile time and run time statistics were gathered to form a baseline against which other Ada compilers (preferably MIL-STD-1750A Ada compilers) may be compared.

I. Introduction

The Ada programming language was developed primarily for use in embedded computers within the United States Department of Defense. An embedded computer is one that is part of a larger system. For example, a guided missile system contains, among others, an embedded guidance computer. These embedded computers may physically consist of anything from a stand-alone microprocessor to a network of computers. "In general, embedded systems are large and have similar requirements for parallel processing, real-time control, and high reliability" (Booch, 1983:3).

The United States Air Force uses embedded computers in several areas. The particular area of interest to this effort is referred to as the real-time avionics environment. Avionics is a phrase used to mean "aviation electronics". The use of Ada to program applications in this environment is of particular interest to the Air Force since the vast majority of these applications are defined as defense mission-critical applications. On 10 June 1983, Dr. Richard D. DeLauer, Under Secretary of Defense for Research and Engineering, issued a memorandum stating that "the Ada programming language shall become the single, common computer programming language for defense mission-critical applications" (Kramer and McDonald, 1984:42). In addition, the Air Force is the lead service, and the Air Force Wright Aeronautical Laboratories (AFWAL) is the lead organization for the DoD Evaluation and Validation (E & V) Task. In this capacity, AFWAL contracted the Institute for Defense Analyses (IDA) to develop a single, coordinated test suite of programs, taken from several existing test suites, to assist in the effort to evaluate Ada compilers.

This paper discusses a project which establishes a set of issues of particular importance to real-time avionics, evaluates the effectiveness of the IDA test suite towards accomplishing satisfactory evaluation of those issues, searches other sources for tests applicable to the issues, compiles applicable test programs from the IDA test suite and other sources using two Ada compilers, and evaluates the results of running the tests on the compilers' target machines.

Background

In 1975, the Department of Defense (DoD) began a process to select one standard high order language to be used for writing software for embedded computer systems. Most of the languages used at this time were not high order - they were assembly languages. A twofold reason has been given for this action:

1. There were many problems associated with high order language (HOL) design such as:

   - incompatible dialects among existing HOLs.

   - creating entirely new languages very limited in their application.

2. The DoD used over 400 languages in a wide variety of applications. There are high costs

associated with this language proliferation such as:

- the direct cost of language design and compiler implementation efforts.

- language maintenance costs.

- cost of slipped schedules and testing problems.

(Kramer and McDonald, 1984:v)

The approach taken by the DoD to develop a standard HOL for embedded computer systems was unique. The first step was to form a High Order Language Working Group (HOLWG) in 1975. The HOLWG was tasked to identify DoD's requirements for computer programming languages, evaluate the existing languages, and recommend the implementation and control of a "minimal set". Later that year, HOLWG published a requirements analysis, called STRAWMAN, and sent it out for public review. This process was repeated twice more, and in 1976, DoD issued Directive 5000.29 (Management of Computer Resources in Major Defense Systems). This directive limited languages used in Defense System Projects to those approved by the DoD. In November 1976, DoD Instruction 5000.31 was issued which implemented DoDD 5000.29 and established an "Interim List of DoD Approved High Order Languages" that contained seven approved HOLs (Kramer and McDonald, 1984:v).

These languages were not considered to be the long-term solution DoD was looking for at the beginning of the project. In June of 1978, the finalized DoD HOL requirements were released in a report called STEELMAN. The requirements specified in this document were used to competitively select the language developed by Honeywell/Cii-Honeywell Bull as the new standardized HOL for the DoD. In 1979, the new language was officially named Ada to honor Augusta Ada Byron, Countess of Lovelace. The Countess was an associate of Charles Babbage and is presumed to be the world's first programmer (Barnes, 1982:2).

Many other activities took place in the attempt to systematically design and implement Ada as the sole HOL for DoD's many critical systems. Another set of requirements accompanied the language design specifications. These requirements were for the development of a robust Ada Programming Support Environment (APSE) to improve productivity both in software system development and in continued system evolution. Again, world-wide review and many revisions were performed before the final report, called STONEMAN, was released in February 1980.

A proposed language reference manual (LRM) for the new language was released in July 1980 and was designated MIL-STD-1815 in December 1980. In July 1983, when the American National Standards Institute (ANSI) Ada was adopted, the standard was redesignated MIL-STD-1815A.

A compiler validation guide was published in October of 1980 to assist Ada language implementors prepare their compilers to meet the Ada language definition before submitting them to the DoD designated testing facility for validation.

In December 1980, the Ada Joint Program Office (AJPO) was established by the DoD "to manage the effort to implement, introduce, and provide life-cycle support for Ada" (Kramer and McDonald, 1984:vi). One objective of the AJPO is to ensure conformance of Ada language translators to the Ada standard. The Ada Validation Organization (AVO) was established by the AJPO to make sure this objective is reached. The Ada Compiler Validation Capability (ACVC) is a test suite of Ada programs used by the AVO to conduct validation testing of proposed compilers (Kramer and McDonald, 1984:8-10).

Later, in early 1983, the DoD started a project that concentrates on solving mission critical computer software problems. This project is called the Software Technology for Adaptable Reliable Systems (STARS). An objective of the STARS project is "controlling the cost and improving the quality of the software by facilitating the application of modern software engineering practices to mission critical computer system (MCCS) developments" (Kramer and McDonald, 1984:15).

An Evaluation and Validation (E & V) Team was formed in 1983 for the purpose of developing "the techniques and tools which will provide a capability to perform assessment of APSEs and to determine the conformance of APSEs to the CAIS (Common APSE Interface Set)" (Kean, 1984:1).

In September 1983, AFWAL commissioned the Boeing Military Airplane Company (BMAC) to conduct research into the use of Ada in embedded avionic systems. Two of the principal areas of study during the first phase of this research were:

1. Issues relating to the compilation of Ada code for a distributed MIL-STD-1750A architecture.
2. The Run-time Support (RTS) required to support execution of Ada programs.

Much careful work has been accomplished, striving to ensure the Ada language is standardized. Many aids are available to developers and implementors of Ada compilers to assist them in producing compilers that adhere to the rigorous standards set forth by the DoD. The evaluation of these validated compilers by the Air Force is the next step. This process has already begun. The E & V Team has initiated procedures for the development of an Ada Compiler Evaluation Capability (ACEC), a second software test suite which will complement the ACVC. A contribution to this effort is the major concern of this thesis.

The Problem

Ada is a unique computer language in that it is the first practical language to combine important features such as data abstraction, multitasking, exception handling, encapsulation, and generics (Barnes, 1982:vii). An important part of real-time applications is the multitasking feature which will be referred to in this thesis simply as tasking. Tasking is provided for within the constructs of the language itself and is the ability to accomplish a series of activities in parallel instead of in sequential order. Concepts of tasking such as instantiation, scheduling, memory management, rendezvous, and termination present complex problems to developers and implementors of Ada compilers. Many decisions involving timing and scheduling of single processor and multiprocessor systems must be made, then implemented and tested.

Eight Ada compilers were validated and two were re-validated in 1984. These numbers represent a significant increase over the three compilers validated in 1983. As more compilers are validated, DoD organizations are already asking questions about their suitability for real-time applications. The requirement for a means to evaluate Ada compilers becomes more pressing all the time. The test suite of programs assembled by IDA for use by AFWAL as part of the E & V Team activities is an important start toward a complete Ada compiler evaluation capability, the ACEC.

The main goal of the project described in this thesis is to initially identify issues of importance to the real-time avionics environment, and then to find test programs that address those issues. This will be accomplished by first analyzing the Prototype ACEC and tests from other sources. Next, those test programs that are applicable to the identified issues will be compiled and executed. Finally, the results will be reported and discussed.

The Prototype ACEC is included in the analysis because it is essential that any test suite of programs presented for use as a standard for evaluating Ada compilers include tests that address the concerns of real-time programming applications.

Scope

Identification of every compilation and run-time issue related to Ada and subsequently evaluating the test suite of programs in relation to those issues would be too large a task for one master's thesis project. Hence, the project described in this paper will focus on those issues most crucial to a real-time avionics environment with an embedded computer system as the target machine. It involves:

1. Researching issues of the real-time avionics environment;
2. Evaluating the test suite of programs presented to AFWAL by IDA;
3. Compiling those tests selected as applicable to the identified issues using two validated Ada compilers; and
4. Running the compiled tests on the target machines and reporting the results.

General Approach

Issues related to the real-time avionics environment will be identified following a thorough literature search. Then the Prototype ACEC test suite furnished by IDA will be analyzed to determine those tests, if any, that pertain to the identified issues. If a determination is made that those issues suitable for empirical testing are not addressed by the Prototype ACEC, tests from other sources will be presented as candidates for the test suite. Next, the applicable test programs of the IDA test suite and the candidate test programs will be compiled using the previously identified Ada compilers. Finally, the compiled tests will be linked and executed on the target machine and the results will be discussed.

Sequence of Presentation

The real-time avionics environment is described in Chapter II. Also, the Ada constructs of tasking and exception handling are briefly described. Then an outline of some applicable data structures is presented.

An analysis of compilation issues relative to real-time avionics applications is presented in Chapter III. In addition to identification of the issue at hand, the relevance of the issue and possible testing criteria are discussed.

Chapter IV is an analysis of the test suite of programs furnished to AFWAL by IDA. Each program that is designed to test any of the identified real-time compilation issues is evaluated.

Chapter V reports on the results of compiling and running all selected test programs on the Ada compilers.

Finally, conclusions and recommendations concerning this project are discussed in Chapter VI. The objective of the project focused on a specific subset of issues relevant to the real-time avionics environment and the degree to which available compilers and tests address those issues. This chapter also considers expansion of this effort in future projects.

II. The Environment

The real-time avionics environment differs from other environments that are supported by computer programs because of the timeliness required in responding to requests and because of the physical constraints of the hardware. This chapter discusses that real-time environment as well as some of the Ada language issues related to programming for this environment.

Real-time Avionics

A real-time system differs from a conventional interactive or batch oriented system in the timeliness of responses. Ben-Ari states that "the term real-time system is usually restricted to systems that are required to respond to specific predefined requests from a user or an external sensor" (Ben-Ari, 1982:12). Mellichamp defines real-time processing as follows:

> "Real-time processing involves the interconnection of a process with a computer utilizing analog/digital and digital/analog interfaces and/or generalized digital (binary) data interfaces. Data acquisition by the computer must be keyed to the time scale of the process. If the computer is to influence the process as well, then its own response must be timely, resulting in an appropriate process response."
>
> (Mellichamp, 1983:10)

While these definitions hold for most real-time applications, and these applications are well suited for Ada's tasking capability, real-time embedded avionics applications have additional characteristics that must be considered. D. A. Fisher, who was then a DoD Staff Specialist for Computers, Communications, and Command and Control, describes this environment in the following manner:

> "Embedded computer software often exhibits characteristics that are strikingly different from those of other computer applications.... Many embedded computer applications require software that will continue to operate in the presence of faults.... The applications may require the monitoring of sensors, control of equipment displays, or operator input processing. They must interface to special peripheral equipment.... Software must sometimes be able to respond at periodic (real time) intervals, to service interrupts within limited times, and to predict computation times.... In many applications..it is necessary to access, manipulate and display large quantities of data. Much of this data is symbolic or textual rather than numeric and must be organized in an orderly and accessible fashion."
>
> (Department of Defense, 1980:4)

The processor hardware used in avionics applications is another critical factor that must be considered in this environment. "Typical avionics applications take place in an environment where weight, power, and volume of the processor are at a premium" (Phillips and Stevenson, 1984:100). Traditionally, processors used in embedded systems have been restricted due to cost and/or size constraints. The majority of these processors have 64 KB of memory or less. This means that reducing overhead generated by using Ada constructs such as tasking and exceptions is more critical in embedded computer applications than when using a machine in which the addressable memory can be easily expanded. This viewpoint is stated very succinctly in the BMAC report as follows:

"Compilers for embedded avionics are very concerned about the performance of the executed code. Tradeoffs between efforts to improve code and increasing the time (or other cost factors like memory usage) of the compiler are strongly biased towards improved code as long as the resulting compilation costs stay tolerable."

(Avionics Laboratory, 1984:3-44)

In general, avionics applications use processors to provide interface and control within an integrated environment. The processor must sense and control its environment so that it can make and execute decisions. In order to accomplish these objectives, the processor must respond to its environment and dynamically resolve errors.

The processor's response must be fast enough to accommodate asynchronous events, which are usually signaled by interrupts and need to be handled very quickly. The response must also be orderly to ensure the available machine resources are used effectively. This requires an accurately maintained timing relationship between the data sampling rate of the sensors and the frequency at which the data being sampled is available. Failure to accurately maintain this relationship could lead to erroneous extrapolations or an unstable control loop (Phillips and Stevenson, 1984:100).

Software and hardware errors must be resolved in such a way that the processor maintains control of its environment. Software errors can be handled, for example, by restarting the program at some point or using an exception handler routine. Hardware errors can be planned for by using methods such as fault tolerant design or reconfiguration.

Another consideration critical to the real-time avionics environment is that an existing operating system is usually not present in an embedded computer such as the MIL-STD-1750A. The significance of using Ada as the programming language for real-time avionics applications targeted for an embedded computer is that an Ada compilation system must provide various run-time support (RTS) features. In contrast, languages previously used for these applications assumed their programs ran under a user supplied system executive. Usually, various mathematical routines were the only compiler RTS provided by compilation systems of other languages. An Ada program must have the ability to run on a machine without an operating system since "each main program acts as if called by some environment task" (Department of Defense, 1983:10-2). Thus, the Ada compilation system has to act as the executive system and provide for such features as program initiation, construct (i.e., tasking or exception handling) support, and even program termination. It also must provide a program supervisor.

One major area of concern is that the Ada standard (MIL-STD-1815A) cannot be as yet efficiently implemented on a MIL-STD-1750A architecture that is used in the environment by the Air Force. It is expected that the capabilities of the 1750A will be exceeded due to the limits in logical address space, speed requirements, and need for symbolic processing. Major problems faced by implementors using a segmented Instruction Set Architecture (ISA) include:

1. Implementing the tasking model with efficient interrupt handling;

2. Implementing a heap space management method with efficient garbage collection;

3. Implementing data types for efficient range checking;

4. Implementing context switching for efficient task frame management; and

5. Implementing epilog and prolog models with efficient argument binding and large logical address spaces.

(Estes, 1985a:2).

The report on a survey of a major portion of the defense industry associated with the Air Force and Ada examines this area in greater detail (Estes, 1985a).

Architectures and Levels-of-Distribution

Using Ada to write programs for real-time avionics applications has implications that are not fully understood at this time. At present, most applications are suited for single processor shared memory architectures. The single physical processor is divided into separate logical processors that support the parallel processing constructs. However, it is possible that future requirements will dictate using either an architecture featuring a multiprocessor target system (having separate physical processors) with shared memory or one having a multiprocessor target system with separate memories instead of the single processor architecture. The use of either of the multiprocessor architectures increases the complexity of an Ada compilation system and the resultant RTS when compared to the complexity involved in using a single processor shared memory architecture.

There are problems that must be solved before implementing multiprocessor compilers. Ada features that present the most design and implementation difficulties are the conditional entry call and access types. In addition, Ada does not support the assignment of compilation units to specific processors. This could possibly require placing duplicate code on each processor which in turn could lead to problems with Ada scoping rules. The selection and placement of this duplicate code is an area of concern. Other issues such as the use of interrupts, exception handling, and other machine dependent features must be considered. Because of these problems, any difficulties encountered trying to correctly compile and link Ada programs for a single processor architecture are likely to be compounded considerably if transferred to a multiprocessor architecture. Additional problems not applicable to single processor architectures are also likely to surface as research in this area continues. (Armitage and Chelini, 1985:36).

Another factor affecting the complexity of an Ada compiler and its associated RTS is the level-of-distribution of the proposed system. The three possible levels-of-distribution are:

1. No distribution - a single Ada main program together with all its tasks run on a single processor;

2. Fixed assignment of program parts to processors - for example, on program initiation all tasks are assigned a processor and that assignment remains throughout execution; and

3. Dynamic assignment of program parts to processors - in this case the assignment of a program part to a processor may change.

(Lindquist and Joyce, 1985:9).

Matching the three previously mentioned architectures with these levels-of-distribution produces nine possible combinations of systems. The easiest combination for an Ada compilation system to implement correct RTS procedures on is a single-processor architecture with no distribution. The complexity and resultant difficulty of achieving correct and efficient RTS only increases for the remaining combinations. The most difficult is an environment using a multiprocessor without shared memory and dynamically assigning program parts to processors. Figure 1 illustrates these two extremes.

Since many of the issues identified in the next chapter are related either directly or indirectly to tasks, exception handling, or data structures, a brief explanation of these subjects is required. For a more detailed explanation, the reader may refer to Software Engineering With Ada by Booch and to the Ada LRM (Department of Defense, 1983).

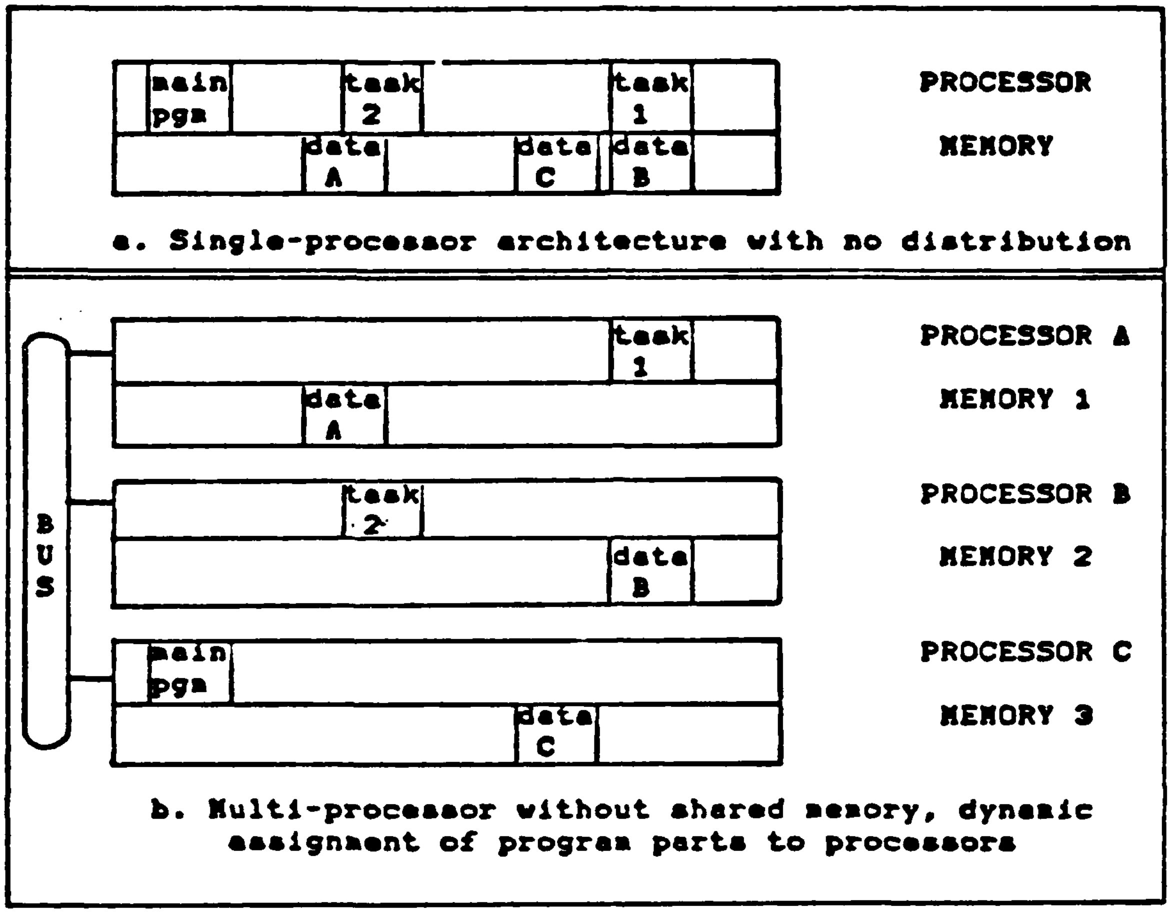


Fig H-1. Extreme Architecture/Distribution Combinations

Tasks

The following definition of tasks is taken from the Ada LRM:

"Tasks are entities whose executions proceed in parallel in the following sense. Each task can be considered to be executed by a logical processor of its own. Different tasks (different logical processors) proceed independently, except at points where they synchronize."

(Department of Defense, 1983:9-1)

This definition makes no distinction between machine architectures. The physical operation of tasks can take place on multicomputer systems, multiprocessor systems, or on logically separate processors in a single-processor system. Whatever the choice of architecture, the effect must be the same. In other words, tasks are executed on their own logical processor regardless of the physical processor configuration.

Tasks are program components with the unique characteristic of operating concurrently with other program components. Tasks depend on the unit which declares them. This unit is often referred to as the "parent". A task declared by a unit is called the "child", while multiple tasks of the same parent are called "siblings". Tasks have a specification part which establishes the interface used by other program components to interact with the task. Tasks also have a body that contains statements defining its actions. When the task is executed, the declarative part of the body is elaborated. This means that the RTS can now associate a name with the task and can initialize newly declared variables of the task. Elaboration is followed by the action associated with the body's statement sequence. The statement sequence often takes the form of an infinite loop in order to continue processing indefinitely. Also, the delay statement is often used to establish a pattern within the task, typically in monitoring applications.

Tasks can usually be broadly assigned to one of two groups according to the specific function they perform in the system to which they belong. These groups have been labeled "producers and consumers" (Booch, 1983:233) or "servers and requestors" (Olsen and Whitehill, 1983:160). Producers or servers exist to provide services to consumers or requestors. A server task is able to provide any one of several services if the select statement is used by the programmer.

Communication between tasks is vital, especially since tasks are executed asynchronously. Explicit, programmer-specified synchronization is provided by what is known as a rendezvous. This "rendezvous" is the message-passing concept in Ada. This concept works through the use of task entries. A task is not required to contain any entries. However, in order to become synchronized with other tasks, it must have an entry and be called by another task. In this case, the task may accept a call of an entry by executing an accept statement for the entry. Synchronization is then achieved between the calling and the accepting task, and a rendezvous has occurred. During the rendezvous, memory is protected thus accomplishing the "critical region" function of mutual exclusion necessary in real-time applications. After a rendezvous is complete, the two tasks continue independently. The program may choose to ignore a rendezvous by using flags known as "guards" attached to the select statement.

While an Ada program is running, a task can be in one of several states. The task can be activated, executed, suspended, completed, or terminated.

Task activation is the process by which tasks are put into execution. The first step of this process is the elaboration of the task body's declarations. It is during this state that the run-time system must initialize its tasking data structures. The task is then ready to execute the sequence of instructions in its body. A task is activated prior to execution of the first statement in the program component that declared the task.

When activation is through, the task body is executed. This can occur in parallel with other tasks or program components and is not dependent on them.

A task can be suspended as necessary in order to rendezvous with other tasks. This takes place at the entry points. It is also suspended if it has to wait on a processor.

A task is completed upon reaching the end of the sequence of statements in the task body.

A task can be terminated normally or abnormally. A task is considered to have terminated normally if it reaches the end of its statement sequence and all of its dependent tasks are completed. Abnormal termination is explicitly controlled by using either the terminate alternative in a select statement or by using the abort statement. The terminate alternative is applicable only if the block of which the task is a part is waiting for termination of its dependent tasks. The more powerful abort statement may be used to terminate any task, even itself, and all dependents of the task. However, if the task is in the terminate state already, the abort statement has no effect.

Errors can occur in any Ada unit, including tasks. Ada has provided the exception mechanism to allow the programmer to respond to situations beyond the scope of normal program operations. The exception and exception handling are described in the next section.

Exceptions

Any of several different factors may cause an error during execution of a program on a computer. Most errors cannot be ignored. Good programming practices dictate that the program react to errors. This reaction can take the form of some specific action to correct or "get around" the error, or the program can abort and allow the operating system to decide subsequent operations.

In real-time avionics applications, software and hardware systems must be able to continue performing their functions in spite of serious errors. This is especially critical since these systems are often used to support human life as well as investments of extremely large amounts of money.

The designers of Ada provided a mechanism intended "to be used to report and handle unusual errors that are not expected to occur when a program is executing properly" (Olsen and Whitehill, 1983:183). This mechanism is called an exception and, together with an associated exception handler, provides the programmer the ability to respond to the exception or to continue processing with reduced capability. An exception is raised as it is brought to the program's attention. Exception handling refers to the response to the raised exception. The LRM lists several pre-defined exceptions which are always available to the program unless explicitly suppressed by the use of the pragma SUPPRESS (Department of Defense, 1983:11-1). This pragma is discussed in the next chapter.

An exception handler is associated with one or more specific exceptions and contains the action to be taken when those exceptions are raised. When an exception occurs during execution of any unit, including an exception handler, none of the remaining statements of the unit are executed. Control goes to an exception handler if there is one associated with the particular exception. If not, the exception is propagated to either the caller of the subprogram, the enclosing unit of the block, or the parent unit of a task. Thus, the exception is propagated all the way to the main body if necessary. If there is still no exception handler for the exception, the program is then terminated.

Data Structures

Two classes of data structure are referenced in this thesis. The first is the shared variable and the second is an indivisible data structure that exceeds memory boundaries.

As previously mentioned, tasks communicate via entry calls and accept statements. If two tasks read (load) or update (store) a variable accessible to them both, that variable is called a shared variable. Neither of the two tasks may assume anything about the order in which the other task's operations are carried out except when the twc tasks are synchronized. Synchronization occurs at the start and end of two tasks' rendezvous. Also, at the start and end of a task's activation, it is synchronized with the task that caused the activation (its parent unit). Any task that has completed its execution is likewise synchronized with any other task (Department of Defense, 1983:9-19).

Ada provides a pragma called SHARED. This pragma can be applied to an appropriate variable to specify that every read or update of the variable is a synchronization point for the variable. It is the programmer's responsibility to use the shared variable correctly. The pragma SHARED is necessary to ensure that the shared data are properly referenced by two or more tasks. An implementation must restrict the objects for which the pragma SHARED is allowed such that direct reading and direct updating of the objects is implemented as an indivisible operation (Department of Defense, 1983:9-19).

An indivisible data structure that exceeds memory boundaries is a distinct possibility in real-time avionics applications. An example of the use of this structure might be for scene generation on an aircraft console's "Heads Up

Display." While it is possible and, in the avionics environment, highly desirable to restrict memory use to the applicable bounds (normally 64 KB of memory), future requirements could dictate using these large indivisible data structures. For such cases in which large data structures are unavoidable, the associated overhead required to track and access the data could be unacceptable. Currently, intimate involvement by the application programmer regarding the design and physical layout of the code is required in order to avoid problems with large data structures.

The above discussion of the environment of real-time avionics applications for use in an embedded computer reveals unique considerations regarding the use of Ada in that environment. Therefore, compiler and/or RTS solutions that might have applicability in the normal real-time environment could be unsatisfactory when applied to avionics applications because of the stringent requirements of the avionics environment. Programmers and designers of avionics systems are also concerned whether Ada compilers and their associated RTSs effectively and efficiently address issues of importance to the avionics environment. A set of these issues is identified and discussed in the next chapter.

III. Issues

There is continuing research regarding the best methods of implementing RTS for Ada with various architecture/level of distribution combinations. Recently, methods have been presented that show how tasking can be implemented efficiently for multiprocessors with shared memory (Lindquist and Joyce, 1985:9-19). Their paper also references the implementation methods that D. Cornhill and E.S. Roberts proposed for other architecture/level-of-distribution combinations. An efficient implementation of tasking for a single shared processor has also been demonstrated. This implementation uses "simple, straight-forward, and efficient algorithms" (Baker and Riccardi, 1985:34). Designs for other specialized RTS systems are presented in the literature (Leathrum, 1984:4-13) and (Riccardi and Baker, 1984:14-22) .

As noted previously, the real-time avionics environment presents special challenges and calls for special solutions within the constraints of the environment. For example, if an effective implementation of tasking were developed that required 36 KB of memory, many applications could and probably would use the implementation. However, that amount of memory in the current real-time avionics environment is not available to support the implementation of a single function.

Many issues of concern have been raised by programmers, designers, and implementors of real-time avionics systems concerning the use of Ada in this environment. These issues of concern provide the impetus for further research and are identified and described in this chapter.

The nineteen issues identified here were compiled after reviewing current literature regarding Ada and the real-time avionics environment. In addition, interviews were conducted with people who work in the environment. The issues were subjectively placed into two categories by the author. First are those issues that are not suitable for empirical testing. Issues were placed into this category if an Ada-based test could not reasonably disclose the issue or that the development of an Ada test program would be too difficult a task for this project. Next, those issues for which it was possible to identify criteria or tests that could be used to evaluate an Ada compiler's treatment of the issues are listed. Ada-based test programs that can be used to test the issue are provided or referenced.

Issues Not Suitable for Empirical Testing

For the following issues, it was determined that either empirical test resolution is uncertain or that tests do not exist and would be too difficult, if not impossible, to develop during this project. In some cases, test development is hampered by not having access to sophisticated enough compilers, RTS systems, or programming tools. In others, the issue does not lend itself to empirical testing.

Issue 1. Is the overhead associated with an effective rendezvous efficient to the point that the time sequenced operations are not disrupted?

The rendezvous is central to the concept of Ada tasking. Software is divided into tasks because it simplifies design and development. However, Ada will not be useful for real-time avionics applications unless the tasking operations (e.g., activation, execution, and termination) are implemented efficiently. The following opinion reflects a major concern of real-time avionics application programmers:

> "Embedded systems which spend half their time doing task manipulation are not going to be very effective in performing their primary function. If it turns out that tasking constructions involve large amounts of overhead, then the application systems will minimize the use of tasking."
>
> (Avionics Laboratory, 1984:3-47).

The responsibility of providing an efficient rendezvous lies with the RTS which is effectively the interface to the architecture. As previously discussed, Ada compilers will have to generate code for a variety of systems associated with any number of host operating systems. These operating systems range from the complex (e.g., VAX 11/780 VMS) to the nonexistant (e.g., MIL-STD-1750A). Accordingly, RTS will range from a collection of primatives that call operating system functions (such as memory management, page swapping, etc.) to RTS that contain routines to implement these functions internally.

However conceived, the RTS must in one way or another provide such functions as synchronization, mutual exclusion, queue processing, memory management (including heap memory allocation and perhaps some form of "garbage collection"), stack management, dispatching, and scheduling.

The RTS is required to provide mutual exclusion and synchronization through the rendezvous. Mutual exclusion is the concept of a single task having access to data (memory contents) at any point in time. The rendezvous may be used to enforce this. Synchronization is implicit in the entry call/accept protocol. All this must be accomplished without interfering with the ability of the computer to respond to stimuli from a process sufficiently fast to accommodate the needs of the process.

Issue 2. How does the RTS system deal with the interaction between tasks and lexical scopes?

Every task is created by a declaration. This ties the livelihood of the task to the existance of the declarative framework of the program. The operations which are required to support creation, activation, execution, and termination of tasks have been described as:

1. Create dependent tasks (children).
2. Activate all children. Begin execution when children have ended their activation processing.
3. End of activation.
4. Wait for termination of all children.

5. Terminate.

(Leathrum, 1984:6)

There are some critical coordination points between these operations. As an example, the declaration processing for a child cannot begin until the declaration processing for the parent is completed. At this time, the parent activates all its children. Also, the parent must wait to begin execution of its own statement sequence until all its children have completed their activation operation. Figure 2 illustrates typical Ada life cycles of one parent and two children. The critical coordination points are indicated by "***". These coordination points are critical because, for example, the declaration processing for a child cannot begin until the declaration processing for the parent is complete, at which point the parent does an "activate all children" operation. The parent may not begin executing its own code until all children have performed an "end of activation" operation. Also, the parent may not pass through the end of the scope of its declaration until all its children have terminated.

Issue 3. "If multiprocessing is supported by the implementation, are Ada tasks mapped to a single underlying processor, or is each task mapped to a separate processor?" (Kean, 1984:7)

Implementing Ada on various architecture/level-of-distribution combinations is an area of current research.

PARENT	FIRST CHILD	SECOND CHILD
declaration		
processing		
create first		
child		
create second		
child		
•		
•		
•		
activate all	declaration	declaration
children	processing	processing
	•	activate all
and	•	children
	•	and then
then	•	begin execution
	activate all	end of activation
	children	•
	and	•
	then	•
	begin execution	•
begin execution	end of activation	•
•••		
	•	•
end of activation	•	•
•	•	•
•	•	•
•	•	•
•	•	•
•	end execution	end execution
end execution	wait for children	wait for children
	to terminate	to terminate
	end of declaration	end of declaration
	scope	scope
wait for children	terminate •••	terminate •••
to term •••		
end of declaration		
scope		
terminate		

(Leathrum 1984:6)

Fig H-2. **Typical Ada Life Cycles: One Parent and Two Children**

Hardware designers are now proposing architectures that address some of the more difficult aspects of implementing Ada on conventional architecture machines such as supporting run-time constraint checking and representation of discriminant records and dynamic arrays (Biswas, 1984:23). An instruction set architecture that has special features to support Ada on a computer known as the High Level Language Machine has also been proposed (Avionics Laboratory, 1985). Meanwhile, other ideas are being presented to support distributed Ada tasking. Weatherly describes the fundamentals of a proposed network operating system (Weatherly, 1984:136-144). Cornhill discusses four approaches to partitioning application software for execution on a distributed target system. They are:

1. Write an Ada program for each processor in the distributed system;
2. Partition only tasks;
3. Allow partitioning on any source level construct; and
4. Extend Ada with constructions specific for programming distributed systems

(Cornhill, 1984:153-162).

With the myriad of methods available, the programmer needs to be aware of the implementation method used.

<u>Issue 4</u>. Are shared variables protected by the rendezvous?

Occasionally, there is a need for a variable to be shared by two tasks. On those occasions, the tasks must be assured that the shared variable is not available for either reading or updating by any other task during the time the two original tasks are synchronized. They are synchronized at the start and at the end of their rendezvous. It is important that the implementation restrict the objects for which the pragma SHARED is allowed to objects for which each of direct reading and direct updating is implemented as an indivisible operation (Department of Defense, 1983:9.11).

NOTE! Many identifiers used from this point forward in this paper contain an underscore (_), primarily to make the identifier more meaningful.

Issue 5. What impact on performance does run-time constraint checking have?

The LRM lists many situations which will cause the raising of the CONSTRAINT_ERROR exception. To check each possible situation during run-time would unsatisfactorily degrade the system. There are numerous compiler optimization techniques available which could reduce the costs of constraint checking considerably. However, situations exist that can only be sufficiently tested during run-time. Therefore, the run-time constraint checking that is implemented must not degrade the system's operation such that timing sequences are adversely affected.

Issue 6. How is dynamic type checking of parameters handled and what impact on performance does it have?

The exception CONSTRAINT_ERROR may be raised by a

DISCRIMINANT_CHECK. This checks that a discriminant of a composite value has the value imposed by a discrimination constraint. A discriminant is a special component of certain record and private types. The values of discriminants distinguish alternative forms of values of one of these types. Also, when accessing a record component, a check is made to ensure that it exists for the current discriminant values (Department of Defense, 1983:11.7).

Issue 7. What is the range of typical context switching times?

The context switching scheme is largely implementation dependent since there are different methods that can be used to accomplish this function. One method is for the compiler to generate relocatable code without worrying about the memory boundary. This requires the linker to find all unresolved external references and deal with them by identifying all procedure calls requiring a context switch. Another method is to create some shared data which can be accessed directly by all the applicable memory images. Whatever the method or combination of methods used to accomplish context switching, it must be fast enough not to interfere with the important timing constraints already established.

In order to enhance the following discussion, two assumptions about the environment are made. The first is that a contiguous instruction space is limited to the memory bounds of a single processor (normally 64 KB for avionics applications like the MIL-STD-1750A). The second is that a contiguous data structure is also limited to the available memory bounds. In general, two situations could arise needing context switching. They are instruction branches or calls, and instruction operand data references.

Given the assumptions, the first situation of context switching should be provided by the implementation with relatively little overhead. An optimizing compiler could even decide to duplicate code for small routines within each address space so that they can be accessed without a context switch.

Instruction operand data references are more difficult to implement efficiently. Problems surface when there are general references to data allowed between address spaces, especially with formal parameters and access types. One such problem is that more bits of address must be used to ensure identification of the data object in the extended data address space. Another problem is that a large penalty could be incurred while accessing what is actually local data to accommodate the possibility that the data might not be local (Avionics Laboratory, 1984:4-15 thru 4-19).

Issues Suitable for Empirical Testing

The following issues were placed in this section after determining that empirical testing should disclose the information desired. Also, a program either exists to test the issue, or could be developed in a relatively straightforward manner. Either the source code of existing tests or references to where it may be obtained is listed in Appendix B.

Issue 8. "Does the number of select choices affect performance?" (Kean, 1984:7)

The selective wait form of the select statement allows a combination of waiting for, and selecting from, one or more alternatives. It does not limit the numbers of select choices. One could theorize that an increase in the number of select choices would increase performance since the likelihood of a rendezvous is increased.

Issue 9. "How does using select alternatives affect the performance of the executable code?" (Kean, 1984:7)

Ada provides several variations of the entry call and accept statements. The simple entry call and simple accept statement may be extended by using the conditional entry call, the timed entry call, or the selective wait. The conditional entry call makes the calling task issue a call for another task's entry point with no waiting (zero time delay). It takes the form of:

```
conditional_entry_call ::=
  select
    entry_call_statement [sequence_of_statements]
  else
    sequence_of_statements
  end select;
```

The timed entry call ensures that an alternative sequence of statements is executed if a rendezvous cannot begin in a specified amount of time. It takes the form of:

```
     timed_entry_call ::=
       select
         entry_call_statement [sequence_of_statements]
       or
         delay_statement [sequence_of_statements]
       end select;
```

The selective wait allows a called task to wait for only one of a set of entry points for a rendezvous. It takes the form of:

```
     selective_wait ::=
       select
         select_alternative
      {or select_alternative}
      else
         sequence_of_statements
       end select;
```

where the select_alternative takes the form:

```
     select_alternative ::=
       [when condition =>] select_wait_alternative

     select_wait_alternative ::=
       accept_statement [sequence_of_statements]
     | delay_statement [sequence_of_statements]
     | terminate
```

A special implementation problem is introduced by the selective wait. A task may be ready to accept a call on a set of several entries at the same moment. This requires checking to determine if the called entry corresponds to one of the open alternatives. Also, since there may be several open accept alternatives, the set of pending entries must be checked against the set of open accept alternatives (Baker and Riccardi, 1985:40).

There are many possible ways to implement these alternatives in a compiler. For example, Leathrum has combined these Ada forms into a very general select statement (Leathrum, 1984:5). The reader is referred to Leathrum's paper for the form of this general select statement and his rules summarizing the Ada forms. The chosen method of implementing these Ada forms will determine the effect upon execution time when the forms are used.

<u>Issue 10</u>. "Is it better to have many small tasks with single entry choices or a few large tasks with many select choices?" (Kean, 1984:7)

It is possible in most cases to perform the same function with either of the above strategies. Depending on the implementation techniques used, one strategy could prove to be more efficient than the other. In the past, avionics application programmers have tended to code in large modules in order to reduce the overhead associated with calling the modules. Because the Ada task is assigned its own logical processor (could be physical as well), it might well be more efficient to code in small tasks. If more rendezvous candidates are available, more tasks could be completed in a shorter time. However, the overhead associated with tasking in the particular implementation will determine the better strategy to use (Fogel, 1985).

<u>Issue 11</u>. "Does the ordering of entry clauses in a SELECT impact execution speed?" (Kean, 1984:7)

The select statement is non-deterministic. Thus, if several alternatives can be selected, one is chosen arbitrarily according to the Ada language rules. Over the long run, ordering of entry calls should not impact execution speed.

<u>Issue 12</u>. "Can the Ada scheduler starve a task?" (Kean, 1984:7)

Each task is an independent activity with the capability of interacting with other units. Each task is assigned its own' logical (may be physical as well) processor. Therefore, each task should receive a share of available processing time for its logical processor to execute.

<u>Issue 13</u>. Are there any aids in the compiler or RTS to assist the programmer find deadness errors in tasking programs?

"A new class of errors, not found in sequential languages, can result when the tasking constructs of Ada are used. These errors are called deadness errors and arise when task communication fails" (Helmbold and Luckham, 1984:96).

A task can be considered "dead" if two conditions are true. First, the task becomes blocked. This happens when a task cannot proceed with its own computation because it is waiting for one or more tasks. Being blocked is a normal part of task execution and occurs in several task activities such as: issuing an entry call that has not been answered yet; waiting for an entry call at an accept statement; waiting at the end of a block for tasks dependent on the block to terminate; or finished executing its own code and waiting for its dependent tasks to terminate. The second condition necessary for a task to be considered "dead" is when there is no possible continuation of the program where the blocked task can continue with its own execution. When a program contains a dead task, a deadness error has occurred.

Deadness errors such as global blocking, circular deadlock, and local blocking depend only on the status of the tasks involved. Global blocking errors occur when at least one task is blocked, and every other task is either blocked or is terminated. Circular deadlock occurs when there is a closed circle of tasks and each task has issued an entry call to the next task in the circle. In this situation, each task in the circle waits forever for the next task to accept its entry call. Local blocking is similar to global blocking, but with the universe restricted to those tasks directly or indirectly dependent on one task that is either block-waiting, completed, or issuing an entry call to a task in the dependent universe.

Other forms of deadness errors depend more on the semantics of the particular program in which they occur. An example of this type of deadness error is the call-wait deadlock. In this error, task T_1 is accepting an entry that can be called only by task T_2, and T_2 is calling a different entry of task T_1. If any other task but T_2 could call T_1's entry, this would not result in a deadness error. Therefore, the error is a fault of the program rather than the language. Helmbold and Luckham present a detailed discussion of deadness errors in their paper (Helmbold and Luckham, 1984:96-105) .

Since the use of tasks is an integral part of the real-time avionics environment, any tools such as a run-time monitor, snapshot displays, or state history monitor would be helpful to the programmer.

Issue 14. "Do idle tasks impact the performance of the executable code?" (Kean, 1984:7)

A task should not add to execution time until it is activated. During execution, if one task is ready to rendezvous before the other task, it should not use processing resources. Instead, the waiting task should be suspended until the other task is ready for the rendezvous.

Issue 15. How much overhead in execution time does an exception take if it is never invoked?

Exception handling can and should be considered part of the termination operation of the program unit they reside in rather than the execution operation. Exceptions were designed to be used sparingly and only to protect a program from situations outside the range of normal program bugs. It is desirable that features which are not used, not slow down execution of the program. This means that exceptions should add to the execution time if and only if they are raised.

Issue 16. Does the compiler effectively deal with indivisible data structures exceeding memory boundaries?

It would be extremely inefficient to treat each data element as if it could exceed the memory boundaries. However, in those relatively rare instances in which an indivisible data structure does in fact exceed the boundary, the compiler or associated RTS should be capable of efficient access operations. The programmer needs to know if this type of data structure is allowed and if so, what cost in execution time is required to use it.

Issue 17. What is the effect of using each of the following three options: OPTIMIZE = none? OPTIMIZE = space? OPTIMIZE = time? (Kean, 1984:7-8)

The use of these options could affect the size of resulting object code, the CPU time required for execution of the compiled code, and the maximum/minimum size of the RTS system with or without tasking. As previously stated, the real-time avionics environment is concerned with the size, weight, and space of embedded computers. Also, compilers used in this environment are more concerned with the performance of executable code than other factors. A programmer should be aware of the performance characteristics of these three OPTIMIZE options.

Issue 18. How does use of a SUPPRESS pragma affect execution time?

The use of a SUPPRESS pragma gives permission to an implementation to omit certain run-time checks. It seems logical that the use of a SUPPRESS pragma would reduce run-time. However, implementations that incorporate Ada constructs such as range or length checking within the architecture could possibly require more overhead in execution time and code to suppress those constructs. In the real-time avionics environment, this pragma will probably not be used until the final version of the program is tested. At that point, it would logically be invoked for some areas of code in order to save run-time overhead. The applications programmer needs to know if there is a penalty incurred or no benefit derived from using the pragma SUPPRESS.

<u>Issue 19</u>. For which checks is the pragma SUPPRESS implemented?

"For certain implementations, it may be impossible or too costly to suppress certain checks. The corresponding SUPPRESS pragma can be ignored" (Department of Defense, 1983:11.7). A programmer needs to be aware of which checks the compiler allows to be suppressed and which are too costly to suppress.

<u>Summary</u>

The preceding discussion identifies nineteen issues of concern that have surfaced in current literature or during conversations with experienced real-time avionics application personnel. (For ease of reference, just the issues are listed in Appendix A.) Seven of the identified issues have to be considered "untestable" at this time because of the nature of the issues and the current state of the art regarding the evaluation of concurrent processes. However, the remaining twelve issues can be empirically tested. Applicable tests must now be applied to the issues.

Since the DoD recognizes the benefits of Ada compiler and RTS evaluation, they are gathering a common test suite of programs called the ACEC to assist in this effort. The next chapter presents the first version of this ACEC and analyzes its applicability to the twelve testable issues.

IV. The Prototype ACEC and Other Test Programs

In June 1983, the AJPO proposed that the E & V Task be initiated and a tri-service E & V Team be formed. The Air Force was designated as the lead service and AFWAL was designated as the lead organization of the E & V Task. The overall goal of the E & V Task is to develop and provide the Ada community the technology for the evaluation and validation of ASPEs. In order to accomplish this goal, eleven specific objectives were identified (Castor, 1984:A-8 thru A-12).

One of these objectives is to develop evaluation and validation tools and aids. These include test suites (sets), test scenarios, a data reduction capability, and other designated means of automated support. The ACVC is included in this set of tools. Another tool deemed necessary to support this objective is the ACEC. The Prototype ACEC test suite has been prepared for AFWAL by IDA.

Version 0 (Beta Test) of the Prototype ACEC test suite is the subject of the first section in this chapter. After a brief background discussion of the Prototype ACEC, an analysis of each of the groups of test programs is presented. This analysis is based on the test group's applicability to any of the issues identified in Chapter III. The analysis was conducted by examining the program code and associated documentation. Based on this analysis, each test was either rejected as not applicable or accepted for compilation and execution. A rejection in the context of this thesis in no manner implies that the test is invalid or useless. It simply means that it has been subjectively judged as not directly applicable to the issues identified for empirical testing in Chapter III (listed in Appendix A.) Appendix B is a summary of the results of this analysis.

Other test programs are available in the public domain. Some of these programs have been reviewed by IDA and not included in the Prototype ACEC because they failed to meet one or more of the three test selection criteria established by IDA and listed in the first section. However, if the author determined that a particular test program was applicable to any of the issues numbered 8-19 in Chapter III, the test was included for analysis and is discussed in the second section of this chapter. These tests are compiled and executed in the same manner as the applicable Prototype ACEC tests. Listings of those applicable programs readily available (for example, tests that are part of the ACVC) are referenced but not included as part of this work. Other program listings are contained in Appendix C. Chapter V presents and discusses the results of compiling and executing all of the selected tests.

Prototype ACEC

Background. The purpose of the Prototype ACEC is to provide users or potential users of Ada compilers with a single, organized suite of compiler performance tests.

Included in this prototype ACEC is the support software for executing these tests and for collecting performance statistics. The test programs are those taken from existing test suites in the public domain that meet three selection criteria. These criteria are:

1. The test measures Ada language feature performance versus conformance versus compiler architecture;

2. The test is a unique test; and

3. The test compiles correctly.

(Hook, 1985).

The tests have been organized, and in some cases, instrumented to provide statistics about the Ada language features that are being tested.

The Prototype ACEC is intended for use by programmers or analysts already familiar with a particular Ada compilation system. They must be able to invoke the Ada compiler and the host or target dependent portions of the Prototype ACEC. Familiarity with the operation of the ACVC would benefit the user as the ACEC is roughly equivalent in execution complexity.

The design goal of the Prototype ACEC was to collect objective, quantifiable attributes of an Ada compiler and its associated RTS system. An interested user, programmer, or analyst could use this collection of attributes to evaluate the usefulness of a particular compiler for a specific application. The actual language constructs most frequently used in a particular application will have the most effect on the perceived usefulness of the compiler. For example, if an application plans not to use generic instantiation of packages, procedures, or functions; then it matters little if that part of the compiler/RTS is inefficient. However, if tasking is a substantial part of the application, the user will be most demanding that the compiler/RTS produce efficient, effective tasking constructs.

It is important for users of the Prototype ACEC or any Ada test suite to understand that the measurements obtained from tests are only an indication of the effect produced by an Ada language feature under controlled conditions. The frequency that the features are used and the methodology under which they are programmed in a real application could cause significantly different results. With this in mind, this test suite provides two options for evaluating an Ada compiler. A set of tests that approximate the frequency distribution of the proposed application can be selected or all tests can be executed to evaluate a stress load for the compiler/RTS.

Test Suite Composition. A group of tests consists of basically the initial test (control) repeated one or more times with controlled changes made to produce a test version and perhaps a special version (for example, to test the effect of the pragma SUPPRESS). Normally, these changes are contained in all versions, but appear only as comments in the versions not using them.

Figure 3 shows the group of tests that measure the time required for an add instruction. The three versions are ADDSA1, ADDSA2, and ADDSA3 which are the control, test, and pragma SUPPRESS versions, respectively. Only one version of the test group is shown in Figure 3 with appropriate comments referencing the versions in which changes in the code appear. Examination of the source code reveals that it is identical for all three versions except that certain lines are comments rather than actual code in two of the versions. The version of the test dictates which lines become code vice remain comments. The code exactly as shown in Figure 3 is the control version, with explanatory comments added by the author. The test version is made by deleting the comment indicator (--) from in front of the code "Y := Y + X + X + X + X + X + X + X + X + X + X;." When this line becomes code, the effect of addition can be measured between the control version and the test version. Since everything else is exactly the same, the difference in time divided by the number of additions (10,000 in this case) provides the cost of an addition. Similarly, the pragma SUPPRESS version is made by removing some or all of the comment indicators from the lines of code that invoke the different pragma SUPPRESS statements. The effect of each individual SUPPRESS statement or any combination of SUPPRESS statements could be tested in this manner.

Two categories of tests were established by IDA. They are called Normative and Optional. Test groups placed in the

```
WITH INSTRUMENT; USE INSTRUMENT;
  PROCEDURE ADDSA# IS            -- # IS 1, 2, or 3
-- PRAGMA SUPPRESS (ACCESS_CHECK);
-- PRAGMA SUPPRESS (DISCRIMINANT_CHECK);
-- PRAGMA SUPPRESS (INDEX_CHECK);
-- PRAGMA SUPPRESS (LENGTH_CHECK);
-- PRAGMA SUPPRESS (RANGE_CHECK);
-- PRAGMA SUPPRESS (DIVISION_CHECK);
-- PRAGMA SUPPRESS (OVERFLOW_CHECK);
-- PRAGMA SUPPRESS (STORAGE_CHECK);

-- ********************************************************
-- The  8  lines above are not comments in ADDSA3  (pragma
-- SUPPRESS)
-- ********************************************************

     X : FLOAT := 0.000_001;
     Y : FLOAT := 0.0;

 BEGIN
 START("ADDSA#","ADD PROGRAM, @@@ VERSION - 10_000 ADDS");
   FOR I IN 1..100 LOOP
     FOR J IN 1..10 LOOP

    -- Y := Y + X + X + X + X + X + X + X + X + X + X;

-- ********************************************************
-- The  above line is not a comment in ADDSA2  (test)  and
-- ADDSA3 (pragma SUPPRESS).
-- ********************************************************

      Y := FLOAT(IDENT_INT(I+J));
    END LOOP;
  END LOOP;
STOP;
END ADDSA#;

-- ********************************************************
--The  package INSTRUMENT is part of the  test  suite.  It
--contains  procedure  START that must be invoked  at  the
--start    of  a  test,  before any of  the  other  report
--routines  are   invoked.  It  saves the test  name  and
--outputs  the name and   description.  Procedure STOP  is
--also in INSTRUMENT and must  be invoked at the end of a
--test. It outputs a message  indicating whether the test
--as a whole passed or failed,   or is not applicable.
-- ********************************************************
```

Fig H-3. **Source code for ADDSA#**

Normative category are intended to provide a means for determining the system costs for a particular language feature. They should provide a quantifiable indication of the compiler's usefulness. Within the Normative category, a test group is placed into either the Performance or Capacity sub-category. Performance tests collect speed and space attributes for various Ada features while Capacity tests will indicate the limits imposed by the compiler/RTS.

Test groups of the Optional category may be selected by the user to represent an applications profile consisting of the most frequently used language features. There are also two sub-categories within the Optional category. Features tests provide measurement of those features which are not a required part of the Ada compiler. They also provide measurements of the effects of certain compiling options referenced in Chapter 13 of the LRM. Tests of the Special Algorithms sub-category are combinations of language constructs that are characteristic of synthetic benchmark programs. Some examples of these are Whetstone, Dhrystone, and the Sieve of Eratosthenes.

While not actually part of the tests, the Prototype ACEC provides some "support software" which is designed to make it easier for the user to schedule compiles and execution runs of the tests and maintain records of the test activities. The "support software" architecture and operating instructions are contained in the User's Manual for the Prototype ACEC (Hook and others, 1985).

<u>Analysis</u>. The groups of programs in the Version 0 (Beta Test) of the Prototype ACEC were analyzed according to the following criteria:

1. Was the Ada language feature tested in the group of programs a key part of the discussion of an issue?
2. Would any information (for example, execution times) gained by using the group of test programs contribute to measurements related to an issue?

The methodology was to make a determination regarding the applicability of each group of tests to any of the issues numbered 8-19 discussed in Chapter III based on these criteria.

A description of the individual Prototype ACEC program groups and comments regarding their applicability to the issues of Chapter III follows. The test program groups that were determined not applicable to any relevant issue were not included for subsequent compile and execution. The results have been summarized in Appendix B for easy reference.

The tests labeled ADDSA1, ADDSA2, and ADDSA3 are in sub-category Performance. These tests perform 10,000 floating point additions. They are the control, test, and pragma SUPPRESS versions, respectively. These tests are applicable to Issues 18 and 19.

The tests labeled AKERA2 and AKERA3 are in the Special Algorithm sub-category. This test performs the Ackermann algorithm. The versions are test and pragma SUPPRESS respectively. The test is accepted even though it is only applicable to the same issues (18 and 19) as the ADDSA# tests.

LVRNA2, LVRNA1, LVRNB2, and LVRNB1 are from the Performance sub-category. LVRNA2 and LVRNA1 are the test and control versions respectively and measure reference to one local variable, non-access scalar type. LVRNB2 and LVRNB1 are also the test and control versions, respectively; but these tests measure reference to ten local variables, non-access scalar types. These tests evaluate the code efficiency of referencing local variables. A local variable is defined for purposes of these tests as a variable that is declared immediately within the given declarative region. These tests are not applicable to any relevant issue.

LVRAA2, LVRAA1, LVRAB2, and LVRAB1 are from the Performance sub-category. LVRAA2 and LVRAA1 are the test and control versions respectively and measure reference to one local variable, access type. LVRAB2 and LVRAB1 are also the test and control versions, respectively; but these tests measure reference to ten local variables, access types. These tests evaluate the code efficiency of referencing local variables. A local variable is defined as above. These tests are not applicable to any relevant issue.

BRUAA2, BRUAA1, BRUNA2, AND BRUNA1 are from the Performance sub-category. These tests evaluate the code efficiency of a reference from within a block to an uplevel variable. An uplevel variable is defined for purposes of these tests as a variable that is declared in a declarative region that statically encloses the block. The reference is to a variable declared in the immediately enclosing region (one level up). BRUAA2 and BRUAA1 reference an access type in this mode while BRUNA2 and BRUNA1 reference a non-access scalar type variable. These tests are not applicable to any relevant issue.

PRUAA2, PRUAA1, PRUNA2, AND PRUNA1 are from the Performance sub-category. These tests evaluate the code efficiency of a reference from within a procedure to an uplevel variable. An uplevel variable is defined for purposes of these tests as a variable that is declared in a declarative region that statically encloses the procedure. The reference is to a variable declared in the immediately enclosing region (one level up). PRUAA2 and PRUAA1 reference an access type in this mode while PRUNA2 and PRUNA1 reference a non-access scalar type variable. These tests are not applicable to any relevant issue.

GVRAA2, GVRAA1, GVRNA2, AND GVRNA1 are from the Performance sub-category. These tests evaluate the code efficiency of referencing a global variable. A global variable is defined for purposes of these tests as a variable that is declared in the outermost declarative region. GVRAA2 and GVRAA1 reference an access type in this mode while GVRNA2 and GVRNA1 reference a non-access scalar type variable. These tests are not applicable to any relevant issue.

FPRAA2, FPRAA1, FPRNA2, AND FPRNA1 are from the Performance sub-category. These tests evaluate the code efficiency of referencing a formal parameter of mode IN OUT. FPRAA2 and FPRAA1 reference an access type in this mode while FPRNA2 and FPRNA1 reference a non-access scalar type variable. These tests are not applicable to any relevant issue.

LRR1A2 and LRR1A1 are from the Performance sub-category. LRR1A2 is the test version and LRR1A1 is the control version. These tests evaluate the code efficiency of referencing a first level component of a local record variable. A local record variable is defined for purposes of these tests as a variable in a record that is declared immediately within the declarative region in which the component is referenced. These tests are not applicable to any relevant issue.

LRR2A2 and LRR2A1 are from the Performance sub-category. LRR2A2 is the test version and LRR2A1 is the control version. These tests evaluate the code efficiency of referencing a second level component of a local record variable. A local record variable is defined as above. These tests are not applicable to any relevant issue.

LRR3A2 and LRR3A1 are from the Performance sub-category. LRR3A2 is the test version and LRR3A1 is the control version. These tests evaluate the code efficiency of referencing a third level component of a local record variable. A local record variable is defined as above. These tests are not applicable to any relevant issue.

LAVRA2, LAVRA1, LAVRB2, and LAVRB1 are from the Performance sub-category. LAVRA2 and LAVRA1 are the test and control versions respectively and measure reference to one component of a local array. LAVRB2 and LAVRB1 are also the test and control versions, respectively; but these tests measure ten local references. These tests evaluate the code efficiency of referencing a component of a local variable array. A local variable array is defined for purposes of these tests as an array that is declared immediately within the given declarative region in which the component is referenced. These tests are not applicable to any relevant issue.

NL00A1, NL07A2, and NL65A2 are from the Capacity sub-category. These tests evaluate the overhead for nested loops. NL00A1 contains 0 loops, while NL07A2 contains 7 loops, and NL65A2 contains 65 loops. These tests are not applicable to any relevant issue.

The tests labeled MULTA1, MULTA2, and MULTA3 are in the Performance sub-category. These tests perform 10,000 multiplications. There are control, test, and pragma SUPPRESS versions, respectively. These tests are accepted because of their applicability to Issues 18 and 19.

CHSSA1, CHSSA2, and CHSSA3 are from the Special Algorithms sub-category. These tests perform a character string search. There are control, test, and pragma SUPPRESS versions respectively. These tests are accepted because of their applicability to Issues 18 and 19.

SIEVA1 and SIEVA2 are from the Special Algorithms sub-category. These tests perform the "Sieve of Eratosthenes" benchmark. They are not applicable to any relevant issue.

The following tests were delivered to AFWAL 18 October 1985 and are part of the Prototype ACEC. The documentation was changed and written information regarding the sub-category assignments of the tests was deleted. This information is available through the support system which was not used for this project.

AOCEA1, AOCEA2, AOIEA1, and AOIEA2 evaluate a code improvement potential that results from executing a less expensive arithmetic operation when the opportunity is offered to replace a multiplication that uses a loop parameter and a constant with an addition. These tests are applicable to Issue 17.

ASSIA2, ASSIA3, ASSIA4, ASSIA5, and ASSIB2 have groups of assignment statements and comments in various distributions. They are not applicable to any relevant issue.

BALPA1 and BALPA2 evaluate the code efficiency of a simple loop statement and are not applicable to any relevant issue.

BLEMA2 checks that blocks can be embedded up to an arbitrary level (65). It is not applicable to any relevant issue.

BSRCA2 and BSRCA3 are packages that implement a generic binary search function designed to allow use of an enumeration type for the table index. These tests were included for uniformity with other generic operations on unconstrained arrays and are not applicable to any relevant issue.

C31PA2 checks that 31 parameters may be passed and is not applicable to any relevant issue.

CAPAA1, CAPAA2, CAPAB1, and CAPAB2 evaluate the code efficiency of a call to a procedure that has a single formal IN OUT parameter of an unconstrained array subtype. They are not applicable to any relevant issue.

CASEA2 checks a case statement of size 256 and is not applicable to any relevant issue.

CENTA2 and CENTB2 check enumerated types and enumerated literals, respectively, and are not applicable to any relevant issue.

CSBTA1, CSBTA2, CSCTA1, CSCTA2, CSDTA1, CSDTA2, CSETA1, CSETA2, CSSTA1, CSSTA2, CSSTB1, CSSTB2, CSSTC1, CSSTC2, CSSTD1, CSSTD2, CSSTE1, and CSSTE2 evaluate the code efficiency of various case statements (i.e., binary, clustered, dense, exhaustive, and sparse). They are not applicable to any relevant issue.

DRPCA1 and DRPCA2 evaluate the efficiency of a recursive call to a procedure that has no formal parameters. They are not applicable to any relevant issue.

F1IUA1, F1IUA2, FL2RA1, FL2RA2, FLP1A1, FLP1A2, FLP2A1, and FLP2A2 evaluate the code efficiency of loop statements in various forms. They are not applicable to any relevant issue.

FACTA1 and FACTA2 determine a factorial value using a recursive function and are not applicable to any relevant issue.

FPAAA1, FPAAA2, FPAAB1, FPAAB2, FPANA1, FPANA2, FPANB1, FPANB2, FPANC1, FPANC2, FPAND1, and FPAND2 evaluate the code efficiency of a call to a procedure that has formal IN OUT parameters of the same generic formal type. They are not applicable to any relevant issue.

HSDRA2 is a heapsort benchmark program and is not applicable to any relevant issue.

IADDA1, IADDA2, IDIVA1, IDIVA2, IEXPA1, IEXPA2, IMIXA1, IMIXA2, IMIXB1, IMIXB2, IMIXC1, IMIXC2, IMIXD1, IMIXD2, IMIXE1, IMIXE2, IMODA1, IMODA2, IMULA1, IMULA2, IREMA1, IREMA2, ISUBA1, and ISUBA2 evaluate integer expressions and are not applicable to any relevant issue.

INTDA2, INTDB2, and INTDB3 evaluate Ada declaration statements and are not applicable to any relevant issue.

INTQA2 evaluates a full integer queue and is not applicable to any relevant issue.

ISEQA2, MTCQA2, PGQUA2, SQ1OA2, SQPGA2, and VPGSA2 evaluate PUT, ASSIGN, RESET, and GET sequences. They are not applicable to any relevant issue.

LFIRA1, LFIRA2, LFSRA1, and LFSRA2 evaluate loops and are not applicable to any relevant issue.

LOAEA1, LOAEA2, LOECA1, LOECA2, LOFCA1, LOFCA2, LONEA1, LONEA2, LOSCA1, LOSCA2, LOUIA1, LOUIA2, LOUSA1, and LOUSA2 evaluate code improvement potential in loops. They are relevant to Issue 17.

MINIA2 is a nominal minimum program with one declaration and one assignment. It yields three statements and five lines. It is not applicable to any relevant issue.

MTESA2 and MTISA2 evaluate an empty set of enumeration type and an empty set of integers, respectively. They are not applicable to any relevant issue.

NPPCA1 and NPPCA2 evaluate the efficiency of a call to a procedure that has no formal parameters and are not applicable to any relevant issue.

NRPCA1 and NRPCA2 evaluate the code efficiency of a nested recursive call to a procedure that has no formal parameters and are not applicable to any relevant issue.

NULLA1 and NULLA2 are null procedures that don't have declarations or executable statements and are not applicable to any relevant issue.

OPAEA1, OPAEA2, OPBFA1, OPBFA2, OPCEA1, OPCEA2, OPCFA1, OPCFA2, OPDSA1, OPDSA2, OPISA1, OPISA2, OPLEA1, OPLEA2, OPNFA1, OPNFA2, OPSCA1, OPSCA2, OPSEA1, and OPSEA2 evaluate a code improvement potential that results from the reduction of computational operations when the opportunity is offered. Areas presented are: arithmetic elimination, boolean folding, function call elimination, constant folding, distributed simplification, identity simplification, load elimination, numeric folding, subscript calculation elimination, and store elimination. These tests are applicable to Issue 17.

PIALA2 is a version of the PI benchmark test program and is not applicable to any relevant issue.

PKGEA1, PKGEA2, PKGSA1, and PKGSA2 evaluate packages and are not applicable to any relevant issue.

PRCOA2 evaluates tasking performance using the buffering task given as an example in Chapter 9 (9.12) of the LRM. It is not applicable to any relevant issue.

PRPCA1 and PRPCA2 evaluate the code efficiency of a parallel recursive call and are not applicable to any relevant issue.

PUZZA2 and PUZZA3 are puzzle programs and are applicable to Issues 18 and 19 because PUZZA3 uses pragma SUPPRESS.

RANDA2 returns a random integer result and is not applicable to any relevant issue.

RCDSA2 checks 400 field records and is not applicable to any relevant issue.

RENDA1 and RENDA2 measures the time required for a simple rendezvous. This information is critical for evaluating Issue 1. However, more data (such as the timing constraints of the time sequenced operations and the time required for complex rendezvous) are required to complete the evaluation.

SHARA2 illustrates the use of tasking to provide shared access to Global variables and is not applicable to any relevant issue.

SORTA2 is a sort shell and is not applicable to any relevant issue.

SRCRA1 and SRCRA2 evaluate the code efficiency of representing a component of a composite object that has been declared as a record type with a representation clause. They are not applicable to any relevant issue.

SRTEA1 and SRTEA2 evaluate the code efficiency of elaborating an object declaration of a simple record type that has an associated record representation clause. They are not applicable to any relevant issue.

TAIPA1, TAIPA2, TAIPB1, TAIPB2, TAIPC1, TAIPC2, TAIPD1, TAIPD2, TAIPE1, TAIPE2, TAIPF1, TAIPF2, TAIPG1, and TAIPG2 evaluate the effect of an IN parameter size on task performance. They are not applicable to any relevant issue.

TAOPA1, TAOPA2, TAOPB1, TAOPB2, TAOPC1, TAOPC2, TAOPD1, TAOPD2, TAOPE1, TAOPE2, TAOPF1, TAOPF2, TAOPG1, and TAOPG2 evaluate the effect of an IN OUT parameter size on task performance. They are not applicable to any relevant issue.

TPGTA2, TPGTB2, TPGTC2, and TPGTD2 evaluate the impact of using guards on select entries. These tests are applicable to Issue 9, but do not provide comprehensive testing of the issue.

TPITA1, TPITA2, TPITB1, TPITB2, TPITC1, TPITC2, TPITD1, and TPITD2 evaluates the effect of idle tasks on the performance of the executable code. They are applicable to Issue 14.

TPOTA2, TPOTB2, and TPOTC2 contain entry clauses ordered differently in each version. They are applicable to Issue 11.

TPSTA2 and TPSTB2 evaluate the effect the number of select choices has on performance. They are applicable to Issue 8.

TPTCA2, TPTCB2, TPTCC2, TPTCD2, TPUTA2, TPUTB2, TPUTC2, and TPUTD2 evaluate the effect of "chains" of tasks. Each chain task, within each cycle of the loop, calls an entry in the next task in a chain of tasks. They are not applicable to any relevant issue.

TPUTE2 evaluates whether a task gets starved and is applicable to Issue 12.

UAPAA1, UAPAA2, UAPAB1, and UAPAB2 evaluate the code efficiency of a call to a procedure that has a single formal IN OUT type parameter of an unconstrained array. They are not applicable to any relevant issue.

VFADA1, VFADA2, VIADA1, and VIADA2 measure the time required to add elements of vectors and are not applicable to any relevant issue.

WHETA2 and WHETA3 are the Whetstone benchmark test and are applicable to Issues 18 and 19 because of the pragma SUPPRESS.

WHLPA1 and WHLPA2 evaluate the code efficiency of a loop statement using the "while" iteration construct. They are not applicable to any relevant issue.

NOTE: A summary of the results of the ACEC analysis is contained in Appendix B.

Evaluation. Evaluating the performance efficiency of Ada compilers is not a trivial task. Previous studies in this area have been oriented to quantitative performance testing relying on approaches proven useful for evaluation of other language compilers. These approaches included writing a small set of well-established benchmark programs, writing representative application programs, and writing a synthetic benchmark in Ada and other HOL then comparing the resulting compilation and execution times. All three approaches yielded incomplete data. "The quantitative measures are not refined to a level of detail at which remedial action might be suggested to the compiler implementor or user so that improved results might be obtained" (Bassman and others, 1984). The key requirements for evaluating the performance efficiency of the code generated by an Ada compiler are:

1. To provide quantitative data on overall performance efficiency for a particular application domain;

2. To provide quantitative data on performance efficiency that promotes an informed interpretation of the above data;

3. To provide a qualitative assessment of the code generated by the compiler with respect to its immediate and future use; and

4. To counteract any specific effects or interactions, not explicitly required, that may invalidate an evaluation of code efficiency.

(Bassman and others, 1984).

One major problem in collecting an all-inclusive Ada test suite is that the composition of the test suite is still a matter of opinion in the areas of content, objectives, and substance. A related example of this problem is the discovery by at least one compiler implementor that, even though their Ada compiler was validated using tests of the ACVC, it still contained many software errors (Bowles and Olsson-Tapp, 1985).

The criteria used by IDA for including tests in the Prototype ACEC is subjective as is the list of key requirements for evaluating the performance efficiency of Ada compilers used by Basaman and his associates. An interested party could develop different criteria or requirements more suited to the domain with which he or she is familiar. Even so, the development and distribution of a common test suite is important.

The Prototype ACEC provides a valuable first step towards meeting these requirements with respect to the real-time avionics environment. Of the testable issues, only those numbered 10, 13, 15, and 16 do not have applicable tests in the Prototype ACEC. It is encouraging that the remaining eight testable issues are addressed by tests of the Prototype ACEC, especially since the initial thrust of the Prototype ACEC was aimed at establishing a suite of tests that address the most common general areas of concern. However, the purpose of this paper is to focus attention on issues of a specific domain in order that tests relevant to all of the issues become available to the domain. It is the author's view that such tests must be developed or obtained and included in future versions of the ACEC.

The support software that accompanied the Prototype ACEC was not used in this project so it was not evaluated. The test harnesses provided in the User's Manual were used on both the VAX 11/780 and the Data General and greatly facilitated compiling, executing, and timing the test programs.

<u>Other Test Programs</u>

Another major problem in collecting an all-inclusive Ada test suite is the lack of appropriate, quality, adaptable compiler/RTS tests available to the Ada community. In fact, one company recently sponsored a contest featuring cash rewards and publication as prizes for test programs that could be used in its compiler test suite. The contest called for "A suite of smallish Ada programs chosen to maximize the discovery of bugs in Ada compilers" (Bowles and Olsson-Tapp, 1985).

One reason for this problem is that some developed tests are considered to be proprietary information. Accordingly, if a company that intends to bid on Ada projects develops tests related to crucial issues, that company is usually not willing to release those tests to the public. This is because releasing tests could give the company's competitors an unfair competitive advantage by having the tests. Another reason is the difficulty of testing some of the Ada language features themselves, especially in the concurrent processing area. The difficulty arises because of the complexity of the language. Ada expresses a greater range of concepts than most previous languages (e.g., tasks, generics, overloading, packages, representation specifications, and exceptions). Testing of these features is even more difficult in the real-time avionics environment since software for this environment must satisfy real-time requirements and is often composed of multiple concurrent tasks. In addition, the software is usually developed on a host system providing program development services and then moved to the target machine which is normally dissimilar to the host machine (Taylor and Standish, 1984:119).

In spite of the previously mentioned obstacles, there are tests of the Ada language that can be obtained from public sources. Some of these tests were developed by private industry, others by universities, and still others by or for government sponsored projects. Five of these sources are presented below.

Sources of Public Domain Ada Tests. The yearly Ada Fair is sponsored by Los Angeles AdaTEC (now SIGAda). At the 1984 Ada Fair, a suite of tests selected by L. A. AdaTEC was made available to interested vendors. These tests, and others that have been added since, are available on the ARPAnet by logging into EV-INFORMATION at ECLB (with a password of EV to access the EV information area) and entering "HELP ADA-FAIR-PROGRAMS-85". A help menu is available in this area. The programs may be downloaded for use without special passwords. An alternative way to receive the tests is via mail over usenet. This can be arranged by contacting Ed Colbert at "trwrb!trwspp!colbert".

Other tests that were developed by SRI are available through the ARPAnet EV-INFORMATION. They can be accessed by entering "HELP TESTS-SRI".

As mentioned previously, the primary purpose of the ACVC is for validation testing of proposed compilers. While the ACVC is not intended to be a comprehensive evaluation test suite, some of the tests could be used or slightly modified to address specific issues. The office that controls the ACVC and handles inquiries regarding it is ASD/SIOL, Wright-Patterson AFB, OH 45433.

A fourth source of tests is from Telesoft. The suite of tests gathered from their recent contest is available to universities for a handling fee of $50.00, and to other businesses for a handling fee of $500.00. The contest coordinator and contact regarding the test suite is Kami Olsson-Tapp of Telesoft in Fairborn, Ohio (Bowles and Olsson-Tapp, 1985).

The last source of possibly applicable tests presented in this work is textbooks. <u>Software Engineering With Ada</u> by Booch, <u>Ada for Programmers</u> by Olsen and Whitehill, and <u>Ada, an Advanced Introduction</u> by Gehani all contain excellent examples of programs that could be used to test certain Ada constructs.

Applicable Tests From Other Sources. The following tests were subjectively determined to be applicable to one or more of the testable issues listed in Chapter III. The applicability of the test is discussed, and reference to the location of the source code is made. Appendix C contains the source code of several of the tests. Source code for other tests that are readily available from one of the five sources just listed is referenced but not duplicated.

All of the SRI tests listed below have been modified to be compatible with the Prototype ACEC support system and are included in it. Therefore, even though they are described here, the SRI tests were not included in the actual compile and execution phase of the project.

The group of SRI developed tests comprised of SELECT2, SELECT2E, SELECT20, AND SELECT20E are designed to determine if the number of select choices affects performance. This is the concern of Issue 8. In these tests, one task calls a single entry of a second task 1000 times, but the second task has a select statement encompassing some number of alternatives. The test was programmed for 2 and 20 alternatives with the desired entry being the first one in the select list (SELECT2 and SELECT20). It was then repeated with the desired entry being at the end of the select list (SELECT2E and SELECT20E). The ACEC tests TPSTA2 and TPSTB2 are based on these tests.

The results of SRI tests GUARD2, GUARD2E, GUARD2O, GUARD2OE, GUARD2OT, and GUARD2OET, when compared to the results of the tests in the previous paragraph, measure the impact of using guards on select entries. The guards were set in various patterns of true and false. These tests are applicable to Issue 9, but do not provide comprehensive testing of that issue. Further tests are required to adequately test Issue 9. The ACEC tests TPGTA2, TPGTB2, TPGTC2, and TPGTD2 are based on these tests.

The SRI tests MORETASKS, MORETASKS1, MORESELCT, and MORESELCTR are applicable to Issue 10. In the program MORETASKS, a master task calls each of 20 slave tasks, each of which contains a single entry. In MORETASKS1, each task again has a single entry, but it is embedded in a select statement to enable a comparison with the next test, MORESELCT. In this program, a master task calls each of the 20 entries in a single slave task, and the slave task has the 20 entries embedded in a large select statement. In the last program, MORESELCTR, the 20 entries are listed opposite from the order they are called by the master task.

The group of SRI tests comprised of ORDER31, ORDER31R, ORDER32, and ORDER100 are applicable to Issue 11. These tests are modifications of the MORETASKS program described above and contain an increased number of entry clauses with the order juggled in the different versions. The ACEC tests TPOTA2, TPOTB2, and TPOTC2 are based on these tests.

SCHEDTEST is an SRI test program in which a slave task with a two entry select statement is used independently by three other tasks. The test is run until the slave has been called 1000 times. Two of the tasks call the first entry, and the third task calls the second slave entry. Each task and the slave have print statements that report which task is running. The order and relative frequency these printouts appear help determine whether any of the tasks are starved or called more often than others. This test is applicable to Issue 12 and forms the ACEC test TPUTE2.

The objective of the program TEST_DEADLOCK is to determine system behavior concerning timeouts and deadlocks. The approach used was to create two tasks which call each other for entries so as to create a deadlock situation and establish two cases. The first case sets timeouts such that the deadlock is relieved after the timeout period. The second case does not have a timeout option thereby making deadlock inevitable. A time log of when entries are called and accepted in both tasks is provided to allow tracking of the timeouts and the eventual deadlock condition. This program is applicable to Issue 13, although it does not provide a comprehensive test of the issue. The source code of TEST_DEADLOCK and the package HEADER which is used by TEST_DEADLOCK are listed in Appendix C (Ruane and others, 1985: 68-72).

The SRI test programs IDLE1, IDLE5, IDLE10, and IDLE20 contain a "chain" of two tasks. It is called a chain because each chain task, within the cycle of the loop, calls an entry in the next task in the chain. The called entry contains a null statement and returns, and the task then waits to be called by a another task at a similar entry of its own. This chain of length two was cycled 10,000 times. Before the cycles are started, the number of idle tasks associated with the program's name are called at an "init" entry. They are left waiting at a "never" entry which is never called. This group of tests is applicable to Issue 14. The ACEC tests TPITA1, TPITA2, TPITB1, TPITB2, TPITC1, TPITC2, TPITD1, and TPITD2 are based on these tests.

LIDSA1 and LIDSA2 were adapted from the AIMS Interim Technical Report to address Issue 16. These tests attempt to declare an array which is too long to fit within the address space boundary. LIDSA1 forces the compiler/RTS to deal with the problem while LIDSA2 programmatically handles the context switch. A comparison of the execution times of the programs should indicate the cost in time for the compiler/RTS's solution to the problem of large indivisible data structures.

EXCEP2, which addresses Issue 15, is a modified version of the Prototype ACEC test ADDSA2. An EXCEPTION statement which is never executed was added as the last statement in the program. A comparison of the execution times of the two programs (a sufficient number of samples) should indicate the cost in time, if any, for the presence of the EXCEPTION code.

Summary

Tests that are available as part of the Prototype ACEC or from other public domain sources were examined to determine applicability to previously discussed issues of concern regarding the real-time avionics environment. Ten of the twelve issues deemed testable were addressed by readily available tests, although in some cases the tests did not provide comprehensive evaluation of the issue. Tests for the other two testable issues were easily generated.

The Prototype ACEC provides an excellent beginning for the task of assembling a common suite of tests for Ada compiler/RTS evaluation. It needs to be expanded to include more tests dedicated to evaluating the major concerns of the real-time avionics environment. Other sources of tests must continually be examined with this purpose in mind.

The tests that were determined to be applicable in the above discussion were compiled and executed using two different Ada compilers and computers. The results are presented in the next chapter.

V. Compilation and Execution Results

Each test program that was identified in Chapter IV as applicable to at least one of the issues numbered 8-19 identified in Chapter III was compiled using two different Ada compilers. These were the DEC VAX compiler and the ROLM/Data General Ada Development Environment (ADE) compiler. The DEC compiler is hosted on and targeted to the VAX 11/780 computer at AFWAL. The ROLM/Data General ADE compiler is hosted on and targeted to the Data General computer at the Aeronautical Systems Division, Information Systems and Technology Center, Wright-Patterson AFB, Ohio.

Unfortunately, neither of these compilers are targeted to an embedded system. They were used because an Ada compiler targeted to an embedded computer was not available in a suitable stage of implementation for use by this project. Therefore, the results obtained from compiling and executing the test programs on these compilers establish a baseline for comparison when these tests are compiled and executed on an appropriate host/target combination in the future.

Methodology

Each compiler has its own method of interacting with the user and the host operating system. The DEC VAX compiler on the VAX 11/780 interacts with the VMS operating system and requires some preliminary steps to be taken prior to using it. The first step is to enter ACS CREATE LIBRARY

[.ADALIB]. This creates the Ada library in one's directory. This library contains the standard Ada packages and will hold all newly compiled Ada programs, packages, and procedures. Once this step has been completed, it is not repeated because the library remains in the directory. Next, one has to enter ACS SET LIBRARY [.ADALIB]. This tells the Ada compiler which Ada library to use for compiling. After creating the source code for an Ada program, the compiler is invoked by entering ADA/(option) FILE_NAME. If the program successfully compiles, entering ACS LINK PROGRAM_NAME creates an executable file, for example, PROGRAM_NAME.EXE. The VMS command RUN PROGRAM_NAME will execute the program.

The ROLM/Data General ADE compiler on the Data General computer interacts with the OS/VS operating system and operates under the control of the Main Compiler Control (MCC). The MCC verifies the correctness of the command to execute the compiler, initializes functions, verifies the existence of specified objects, calls the front end and back end modules of the compiler, and generates a history and script file. The history file is a record of who performed the compile, why, and when. The compiler consists of the front end and back end modules. The front end checks the correctness of the program, while the back end generates optimized binary code for the target processor. After entering the ADE, compile of source code is accomplished by entering ADA FILE_NAME. When the program is successfully compiled, entering ADALINK PROGRAM_NAME creates an executable file, for example PROGRAM_NAME.EXE.

The Prototype ACEC User's Manual contains a "test harness" that greatly facilitated compilation, execution, and statistic collection on both machines. The test harness for use on the VAX 11/780 was modified to omit collecting execution time and is shown in Appendix D. In the author's opinion, execution time does not enhance evaluation of the program. CPU time provides the true cost in time of the program execution and is not affected by other programs executing along with the program of interest. The test harness for the Data General was also modified and is shown in Appendix E.

A summary of the compilation times from both compilers is contained in Appendix F. The CPU time and object code size figures reported for the Data General are not valid. They were obtained by using the test harness shown in Attachment D. For some undetermined reason, the difference in beginning CPU time and ending CPU time, which is supposed to be the combined compile and link time (in seconds and hundredths of seconds), and the reported object code size (in page sections) both decrease with each program. A sample of the file HARNESS_OUT, which contained the compilation statistics, is shown in Appendix G. Because of the questionable Data General statistics, a valid comparison of the two compiler/RTSs is not possible. Only the VAX 11/780 times should be used for comparison with results from other compiler/RTSs.

The execution times from the two machines are shown in Appendix H. They may be validly compared since the times for the Data General were gathered from the INSTR.DAT file instead of the HARNESS.OUT file of the harness. Times in this file were obtained by using a package called CPU_CLOCK which was generated especially for the Data General by IDA. Therefore, comparing the execution times is meaningful.

<u>Results</u>

The DEC VAX compiler did not allow the pragma SUPPRESS and ignored it during compilation of those tests that used it. This prohibited evaluation of the concerns stated in Issues 18 and 19. Two tests, OPBFA1 and OPBFA2, had an error during compilation which prevented object code generation for them. The statement "pragma main;" was not necessary in the tests that were compiled with the DEC VAX compiler but was required by the compiler as the last statement when the tests were compiled with the ROLM/Data General ADE compiler.

In general, the execution times from the Data General were faster except when tasking was used. For those programs that used tasks, a significant reversal was evident with the VAX 11/780 times being considerably faster. This leads to the conclusion that if a particular domain has many applications that use tasking, perhaps the DEC VAX compiler/RTS would be the better system to use. Otherwise, the ROLM/Data General would be better.

Although the compile and execution runs were not gathered under strict scientific conditions, and used newly developed procedures, they provide a first cut at measuring compiler/RTS systems. By refining the procedures to produce accurate and statistically valid results along with conducting the sample runs in a controlled environment (perhaps a dedicated machine), an interested party would be able to make a valid evaluation of different Ada compilers and RTSs from the results of the tests. That is the purpose of the ACEC, and the Prototype version evaluated in this project is a valuable contribution toward achieving that goal.

VI. Conclusions and Recommendations

The purpose of this thesis was to first examine the real-time avionics environment and identify Ada related issues of that environment, and then to determine if there are adequate tests available for evaluation of Ada compilers/RTS with respect to those issues. AFWAL, acting in its capacity as the lead organization of the DoD for evaluation and validation, is trying to establish a common test suite of programs called the ACEC to enhance Ada compiler/RTS evaluation. The first increment of this ACEC, called the Prototype ACEC, was the logical place to begin searching for tests that addressed the issues. An examination of test programs available from other sources followed. Unfortunately, an Ada compiler to MIL-STD-1750A machine was not available at the time of this thesis, so the tests that were determined to be applicable to any of the avionics environment issues were compiled and executed on other compilers. The results can serve as a baseline for comparison if the tests are used on a MIL-STD-1750A compiler in the future.

Although the project did not accomplish all of the original goals, some important conclusions can be drawn from the experience. Recommendations are also made where areas of future study would be beneficial, either as a continuation of this project, or in other areas that were identified during the project.

The Environment

The real-time avionics environment as described in Chapter II establishes the basis for identifying issues of concern regarding the use of Ada. Implementors of Ada compilers/RTSs as well as users are concerned about the ability of Ada to support their requirements within the constraints of the environment. The Ada constructs examined in the chapter are those most often associated with causing implementors of Ada compiler/RTSs extreme difficulty. The present lack of any validated Ada/MIL-STD-1750A run-time model is a testament to the difficulty of implementing the more complex Ada constructs. It seems unlikely that either the MIL-STD-1750A architecture or the current set of Ada language features will change in the near future. Therefore, the developers and implementors of Ada compilers/RTSs must find an efficient, effective method of implementing the full power of Ada on that architecture. This calls for further research in at least two areas. First, those issues identified in the Estes report should be examined in more detail with the purpose of solving the inconsistencies of the MIL-STD-1750A and the MIL-STD-1815A (Estes, 1985a). Second, research that evaluates other architectures purporting to be designed especially to support Ada should be accomplished with the purpose of finding either enhancements to or eventual replacements for the MIL-STD-1750A.

This project did not examine the use of distributed hardware. The distributed processing environment presents different, perhaps even more difficult challenges for Ada implementors. Further research in this complex area is certainly called for and should be supported.

The Issues

The author makes no claim that the set of issues identified in Chapter III encompass the entire set of concerns of the real-time avionics environment. The issues were compiled after research and through interviews with personnel familiar with the environment. It is important, however, that the set of issues be enhanced and made available to interested parties. Without these issues, it is possible that the areas of concern could be overlooked by implementors and developers of Ada compilers/RTSs or by those responsible for evaluating proposed Ada compilers.

Analysis of Available Tests

Issues 1-7. The first seven issues are those that were subjectively determined to be not suitable for empirical testing. Although these issues are of great concern to the real-time avionics environment, the development of programs that provide users with adequate tests for evaluation of these issues is certainly more than a trivial task. An example of a project that is concerned with testing and validation of tasking constructs is the Arcturus project of Taylor and Standish. They are using an integrated application of a static analysis technique, a dynamic analysis technique, and an interactive debugger in their attempt to assure the reliability of multi-tasking Ada software developed in a host-target environment (Taylor and Standish, 1984:119). However, even as those authors propose their methodology as essential for evaluating multi-tasking software, they also state: "Other analysis aids are surely necessary too. No claim is made that the techniques presented are sufficient" (Taylor and Standish, 1984:119).

The difficulty of developing adequate tests for these issues was created by Ada's complexity - a by-product of its flexibility. At the present time, there are no machines designed exclusively for Ada. Many architectures must be enhanced with new hardware or instruction sets simply to minimally adhere to Ada constructs. Often, a space or time penalty, or both, must be paid. Since Ada was developed primarily to support embedded software applications which commonly use the more advanced Ada constructs, the lack of adequate evaluation tests is unacceptable. Therefore, it is of the utmost importance that research be continued until a comprehensive test suite is available for developers, implementors, and prospective users of Ada compilers/RTSs.

<u>Issues 8-19</u>. These twelve issues were determined to be suitable for empirical testing. Programs that provide at least some testing for each of these issues were found and are readily accessible. The Prototype ACEC addresses all but two of these issues in its present form. Tests for these issues were obtained from other sources, however. This leads

to the recommendation that the developers of the ACEC attempt to identify specific issues of importance to many applications and include programs to test those issues in the expanded test suite. Since some of the issues reported in this project apply to other applications, all of the issues need to be thoroughly addressed by test programs.

Compilation and Execution

An original goal was to compile and execute all applicable test programs on an Ada/MIL-STD-1750A compiler. As the project progressed, it became apparent that the compiler would not be delivered in time. However, two validated compilers were chosen to use for test compilation and execution in order to establish a baseline of compile and execution times against which the MIL-STD-1750A compiler can be compared in the future. This should be accomplished at the earliest possible time.

Any testing should be supervised by a person knowledgeable about the operating system or RTS of the machine being evaluated in order to ensure the methods used to collect time and space statistics are accurate. The environment should be dedicated to the testing without having other processes executing on the machine to provide more accurate statistics.

General

The subject of evaluation testing is debated by scholars, researchers, and within the industry. The definitions of completeness, correctness, and applicability of evaluation testing are controversial. However, the evaluation of computer programs is necessary and desirable. The Prototype ACEC is the correct first step towards providing a common, useful test suite for evaluating Ada compilers and RTSs. It is not, however, as comprehensive as it should be. The Ada community must support the effort to enhance the ACEC by developing evaluation tests that might be specific to their applications and by contributing them to the ACEC. Only in this way will the evolving ACEC improve its applicability to all areas and further benefit of Ada users everywhere. As previously mentioned, developing tests for some of the areas of concern is not a trivial task. Support for continued research in this area must be provided.

Bibliography

Armitage, James W. and James V. Chelini. "Ada Software on Distributed Targets: A Survey of Approaches," Ada Letters, 4: 32-37 (January/February 1985).

Avionics Laboratory. Ada Information Management System (AIMS). Interim Technical Report. Boeing Military Airplane Company, Wichita, KS, 14 November 1984.

Avionics Laboratory. Advanced Avionics Computer Architecture. Volume 1 - Executive Summary. Final Report for Period May 1980 - November 1984. Lawrence Greenspan and Ronald Singletary, Sanders Associates, Inc., Nashua, New Hampshire, May 1985.

Baker, T. P. and G. A. Riccardi. "Ada Tasking: From Semantics to Efficient Implementation," IEEE Software, 2,: 34-46 (March 1985).

Barnes, J.G.P. Programming In Ada. London: Addison-Wesley Publishing Company, 1982.

Bassman, M.J. and others. "Evaluating the Performance Efficiency of Ada Compilers," Proceedings of the Washington Ada Symposium, 1985.

Ben-Ari, M. Principles of Concurrent Programming. Englewood Cliffs, New Jersey: Prentice/Hall International, 1982.

Biswas, Prasenjit. "A Capability Architecture for Ada," IEEE 1984 ADA Applications And Environments Conference, 1984: 23-32.

Booch, Grady. Software Engineering With Ada. Menlo Park, California: The Benjamin/Cummings Publishing Company, 1983.

Bowles, Ken, Chairman, and Kami Olsson-Tapp, Contest Coordinator, Telesoft. "Challenge Contest: An Invitation," Public letter, 1 May 1985.

Cornhill, Dennis. "Four Approaches to Partitioning Ada Programs for Execution on Distributed Targets," IEEE 1984 ADA Applications And Environments Conference, 1984: 153-162.

Department of Defense. Requirements for Ada Programming Support Environments: STONEMAN. Washington, D. C., February 1980.

Department of Defense. Military Standard: Ada Programming Language - ANSI/MIL-STD-1815A. Washington, D. C., January 1983.

Estes, Nelson. "Ada and 1750A: The Challenges of Two Standards," Report. Department of the Air Force, Joint ASD(AFSC) - AFALC Deputy for Avionics Control, Wright-Patterson Air Force Base, Ohio 45433-6503, 3 July 1985.

Estes, Nelson, Embedded Computer Standardization Program Office, Deputy for Avionics Control. Personal interview. AFALC-ASD/AXTS, Wright-Patterson AFB, OH, 23 August 1985.

Fogle, Gary, Avionics Application Programmer. Personal interview. AFWAL/AAA-T, Wright-Patterson AFB, OH, 16 August 1985.

Helmbold, David and David Luckham. "Debugging Ada Tasking Programs," IEEE 1984 ADA Applications And Environments Conference, 1984: 96-105.

Hook, Audrey A., Institute for Defense Analyses, and G. A. Riccardi, Florida State University. "The Prototype Ada Compiler Evaluation Capability (ACEC)". Briefing to the Evaluation and Validation Conference, held at Wright-Patterson AFB, OH, 4 September 1985.

Hook, Audrey A. and others. "Draft User's Manual for the Prototype Ada Compiler Evaluation Capability (ACEC) Version 0 (BETA TEST)". Prepared for the Evaluation and Validation (E & V) Team, Ada Joint Program Office (AJPO) by the Institute for Defense Analyses. August 1985.

Kean, Elizabeth. Evaluation Criteria For Ada Compilers. Pamphlet. Rome Air Development Center, 11 September 1984.

Kramer, John F. and Catherine W. McDonald. Ada Joint Program Office Objectives and Progress - Through 1983. Contract MDA 903 84 C 0031. Institute for Defense Analyses, Alexandria, VA, September 1984 (AD-A149 436).

Leathrum, J. F. "Design of an Ada Run-time System," IEEE 1984 ADA Applications And Environments Conference, 1984: 4-13.

Lindquist, Timothy E. and Richard C. Joyce. "Ada Task Synchronization in a Multiprocessor System with Shared Memory," Journal of Pascal, Ada & Modula-2, 4: 9-19 (January/February 1985).

Mellichamp, Duncan A. "Digital Computing and Real-Time Digital Computing," Real-Time Computing, edited by Duncan A. Mellichamp. Van Nostrand Reinhold Company, New York, 1983.

Olsen, Eric W. and Stephen B. Whitehill. Ada for Programmers. Reston, Virginia: Reston Publishing Company, Inc., 1983.

Phillips, Stephen P. and Peter R. Stevenson. "The Role of Ada in Real Time Embedded Applications," Ada Letters, 3: 99-111 (January/February 1984).

Riccardi, G. A., and T. P. Baker. "A Runtime Supervisor to Support Ada Task Activation, Execution, and Termination (Preliminary Report)," IEEE 1984 ADA Applications and Environments Conference, 1984: 14-22.

Ruane, M. F., and others. Ada Run-time Environment Characterization for JAMPS. Technical Report. Mitre, Bedford, Massachusetts, March 1985.

Taylor, Richard N., and Thomas A. Standish. "Steps to an Advanced Ada Programming Environment," Proceedings of the 7th International Conference on Software Engineering, March 26-29, 1984: 117-125.

Weatherly, Richard M. "A Message-Based Kernal to Support Ada Tasking," IEEE 1984 ADA Applications And Environments Conference, 1984: 136-144.

APPENDIX A

List of Issues

Issues Not Suitable for Empirical Testing

1. Is the overhead associated with an effective rendezvous efficient to the point that the time sequenced operations are not disrupted?

2. How does the RTS system deal with the interaction between tasks and lexical scopes?

3. "If multiprocessing is supported by the implementation, are Ada tasks mapped to a single underlying processor, or is each task mapped to a separate processor?" (Kean, 1984:7)

4. Are shared variables protected by the rendezvous?

5. What impact on performance does run-time constraint checking have?

6. How is dynamic type checking of parameters handled and what impact on performance does it have?

7. What is the range of typical context switching times?

Issues Suitable for Empirical Testing

8. "Does the number of select choices affect performance?" (Kean, 1984:7)

9. "How does using select alternatives affect the performance of the executable code?" (Kean, 1984:7)

10. "Is it better to have many small tasks with single entry choices or a few large tasks with many select choices?" (Kean, 1984:7)

11. "Does the ordering of entry clauses in a SELECT impact execution speed?" (Kean, 1984:7)

12. "Can the Ada scheduler starve a task?" (Kean, 1984:7)

13. Are there any aids in the compiler or RTS to assist the programmer find deadness errors in tasking programs?

14. "Do idle tasks impact the performance of the executable code?" (Kean, 1984:7)

Issues (Con't)

15. How much overhead in execution time does an exception take if it is never invoked?

16. Does the compiler effectively deal with indivisible data structures exceeding memory boundaries?

17. What is the effect of using each of the following three options: OPTIMIZE = none? OPTIMIZE = space? OPTIMIZE = time? (Kean, 1984:7)

18. How does the use of a SUPPRESS pragma affect execution time?

19. For which checks is the pragma SUPPRESS implemented?

APPENDIX B

Summary of Test Group Status

Test Group	Applicable Issue	Test Subject
ADDSA1, ADDSA2, ADDSA3	18,19	Addition
AKERA2, AKERA3	18,19	Ackermann
LVRNA2, LVRNA1, LVRNB2, LVRNB1	N/A	Loc var, scal
LVRAA2, LVRAA1, LVRAB2, LVRAB1	N/A	Loc var, acc
BRUAA2, BRUAA1, BRUNA2, BRUNA1	N/A	Block ref
PRUAA2, PRUAA1, PRUNA2, PRUNA1	N/A	Proc ref
GVRAA2, GVRAA1, GVRNA2, GVRNA1	N/A	Glob var
FPRAA2, FPRAA1, FPRNA2, FPRNA1	N/A	IN OUT parm
LRR1A2, LRR1A1	N/A	Loc rec var
LRR2A2, LRR2A1	N/A	Loc rec var
LRR3A2, LRR3A1	N/A	Loc rec var
LAVRA2, LAVRA1, LAVRB2, LAVRB1	N/A	Loc array
NLOOA1, NLO7A2, NL65A2	N/A	Nested loops
MULTA1, MULTA2, MULTA3	18,19	Multiply
CHSSA1, CHSSA2, CHSSA3	18,19	String srch
SIEVA1, SIEVA2	N/A	Eratosthenes
AOCEA1, AOCEA2, AOIEA1, AOIEA2	17	Optimize code
ASSIA2, ASSIA3, ASSIA4, ASSIA5, ASSIB2	N/A	Assignment statements
BALPA1, BALPA2	N/A	1 loop stmt
BLEMA2	N/A	Embedded blks
BSRCA2, BSRCA3	N/A	Binary srch

Test Group	Applicable Issue	Test Subject
C31PA2	N/A	Parm passing
CAPAA1, CAPAA2, CAPAB1, CAPAB2	N/A	Proc call
CASEA2	N/A	Case stat
CENTA2, CENTB2	N/A	Enumeration
CSBTA1, CSBTA2, CSCTA1, CSCTA2, CSDTA1, CSDTA2, CSETA1, CSETA2, CSSTA1, CSSTA2, CSSTB1, CSSTB2, CSSTC1, CSSTC2, CSSTD1, CSSTD2, CSSTE1, CSSTE2	N/A	Case stmts
DRPCA1, DRPCA2	N/A	Recursion
F1IUA1, F1IUA2, FL2RA1, FL2RA2, FLP1A1, FLP1A2, FLP2A1, FLP2A2	N/A	Loop stmnts
FACTA1, FACTA2	N/A	Factorial
FPAAA1, FPAAA2, FPAAB1, FPAAB2 FPANA1, FPANA2, FPANB1, FPANB2, FPANC1, FPANC2, FPAND1, FPAND2	N/A	Proc calls
HSDRA2	N/A	Heapsort
IADDA1, IADDA2, IDIVA1, IDIVA2, IEXPA1, IEXPA2, IMIXA1, IMIXA2, IMIXB1, IMIXB2, IMIXC1, IMIXC2, IMIXD1, IMIXD2, IMIXE1, IMIXE2, IMODA1, IMODA2, IMULA1, IMULA2, IREMA1, IREMA2, ISUBA1, ISUBA2	N/A	Integer Expressions
INTDA2, INTDB2, INTDB3	N/A	Declr stmts
INTQA2	N/A	Int queue
ISEQA2, MTCQA2, PGQUA2, SQ1OA2, SQPGA2, VPGSA2	N/A	I/O seqs
LFIRA1,LFIRA2,LFSRA1,LFSRA2	N/A	Loops
LOAEA1, LOAEA2, LOECA1, LOECA2, LOFCA1, LOFCA2, LONEA1, LONEA2, LOSCA1, LOSCA2, LOUIA1, LOUIA2, LOUSA1, LOUSA2	17	Optimize loops
MINIA2	N/A	Minimal pgm

Test Group	Applicable Issue	Test Subject
MTESA2, MTISA2	N/A	Empty set
NPPCA1, NPPCA2	N/A	Proc call
NRPCA1, NRPCA2	N/A	Recursive call
NULLA1, NULLA2	N/A	Null procs
OPAEA1, OPAEA2, OPBFA1, OPBFA2, OPCEA1, OPCEA2, OPCFA1, OPCFA2, OPDSA1, OPDSA2, OPISA1, OPISA2, OPLEA1, OPLEA2, OPNFA1, OPNFA2, OPSCA1, OPSCA2, OPSEA1, OPSEA2	17	Optimize
PIALA2	N/A	PI test
PKGEA1, PKGEA2, PKGSA1, PKGSA2	N/A	Packages
PRCOA2	N/A	Buffering task
PRPCA1, PRPCA2	N/A	Parallel recur
PUZZA2, PUZZA3	18,19	Puzzle
RANDA2	N/A	Rand num gen
RCDSA2	N/A	Record fields
RENDA1, RENDA2	1	Rendezvous
SHARA2	N/A	Access to glbl
SORTA2	N/A	Sort shell
SRCRA1, SRCRA2	N/A	Composite obj
SRTEA1, SRTEA2	N/A	Simple rec typ
TAIPA1, TAIPA2, TAIPB1, TAIPB2, TAIPC1, TAIPC2, TAIPD1, TAIPD2, TAIPE1, TAIPE2, TAIPF1, TAIPF2, TAIPG1, TAIPG2	N/A	IN parm to tsk
TAOPA1, TAOPA2, TAOPB1, TAOPB2, TAOPC1, TAOPC2, TAOPD1, TAOPD2, TAOPE1, TAOPE2, TAOPF1, TAOPF2, TAOPG1, TAOPG2	N/A	IN OUT parm to task
TPGTA2, TPGTB2, TPGTC2, TPGTD2	9	Guards

Test Group	Applicable Issue	Test Subject
TPITA1, TPITA2, TPITB1, TPITB2, TPITC1, TPITC2, TPITD1, TPITD2	14	Idle tasks
TPOTA2, TPOTB2, TPOTC2	11	Entry clauses
TPSTA2, TPSTB2	8	Num selects
TPTCA2, TPTCB2, TPTCC2, TPTCD2, TPUTA2, TPUTB2, TPUTC2, TPUTD2	N/A	Task chains
TPUTE2	12	Starved task
UAPAA1, UAPAA2, UAPAB1, UAPAB2	N/A	IN OUT as parm
VFADA1, VFADA2, VIADA1, VIADA2	N/A	Vector add
WHETA2, WHETA3	18,19	Whetstone
WHLPA1, WHLPA2	N/A	Loop while

APPENDIX C

Source Listings of Other Programs

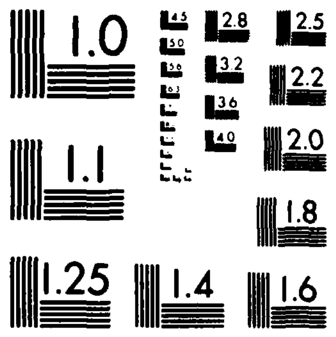
This is the source listing for TST025. It was obtained from another project (Ruane and others, 1985:69-71).

```
--*********************************************************************--
--Ada Run Time Environment Test Mitre Dept D-67                         --
--Test Issue #25                                      Project 4100      --
--                                                                      --
--                                                                      --
--Description:                                                          --
--                                                                      --
--  Create deadlock situation to determine whether system will          --
--  recognize deadlock and acknowledge the problem.                     --
--                                                                      --
--                                                                      --
--                                                                      --
--                                                                      --
--Programmer   : Michael Ruane, Joseph Galia                            --
--Version      : 1.0                                                    --
--Date         : 31 July 1984                                           --
--Filename     : TST025.ADA                                             --
--Computer     : ROLM/ADE                                               --
--
--*********************************************************************--


 with TEXT_IO,
      CALENDAR;

use TEXT_IO, CALENDAR;

procedure TEST_DEADLOCK is

  DEAD_LOCK : BOOLEAN := FALSE;
  SECS      : constant := 1.0;
  PASSES    : INTEGER := 2;

  package INT_IO is new INTEGER_IO (INTEGER);
  use INT_IO;

  procedure PUT_TIME is

    package DUR_IO is new FIXED_IO (DURATION);

  begin
    DUR_IO.PUT (ITEM=> SECONDS (CLOCK));
    NEW_LINE;
  end PUT_TIME;
```

```
task ONE is
  entry STARTUP;
  entry INTO_ONE;
end ONE;

task TWO is
  entry STARTUP;
  entry INTO_TWO;
end TWO;

      task body ONE is
begin
  for I in 1 .. PASSES loop
    accept STARTUP;            -- synchronize test
    PUT_TIME;
    SET_COL (TO => 20);
    PUT_LINE ("Task ONE Startup");
    NEW_LINE;
    PUT_TIME;
    SET_COL (TO => 20);
    PUT_LINE ("Call to task TWO");
    TWO.INTO_TWO;
    PUT_TIME;
    SET_COL (TO => 20);
    PUT_LINE ("Return to Task ONE");
    PUT_TIME;
    SET_COL (TO => 20);
    PUT_LINE ("Accept Task Two entry or wait 15 secs");

    select
      accept INTO_ONE do
        PUT_TIME;
        SET_COL (TO => 20);
        PUT_LINE ("Accepted Task TWO entry");
      end INTO_ONE;
   or
      delay 15 * SECS;
      PUT_TIME;
      SET_COL (TO => 20);
      PUT_LINE ("15 sec wait over");
    end select;
  end loop;
end ONE;

task body TWO is
begin
  for I in 1 .. PASSES loop
    accept STARTUP;            -- synchronize test
    PUT_TIME;
    SET_COL (TO => 40);
    PUT_LINE ("Task TWO Startup");
    NEW_LINE;
```

AD-A172 343 EVALUATION AND VALIDATION (E&V) TEAM PUBLIC REPORT VOLUME 2(U) AIR FORCE WRIGHT AERONAUTICAL LABS WRIGHT-PATTERSON AFB OH R SZYMANSKI 30 NOV 85 5/5

UNCLASSIFIED AFWAL-TR-85-1016-VOL-2 F/G 14/2 NL


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3

```
    accept INTO_TWO do
      PUT_TIME;
      SET_COL (TO => 40);
      PUT_LINE ("Entry accepted in Task TWO");
      PUT_TIME;
      SET_COL (TO => 40);
      PUT_LINE ("Timed entry call into Task ONE: 20 sec");

      if not DEAD_LOCK then
        select
          ONE.INTO_ONE;
        or
          delay 20 * SECS;
          PUT_TIME;
          SET_COL (TO => 40);
          PUT_LINE ("Waited too long. Finished Task TWO");
        end select;
      else
        -- DEADLOCK *?*?*?*?*?*?*??*??*??*???--
        SET_COL (TO => 40);
        PUT_LINE ("Deadlock about to be initiated");
        ONE.INTO_ONE;
      end if;
    end INTO_TWO;
  end loop;
end TWO;

begin
-- TEST DEADLOCK

  for PASS in 1 .. PASSES loop
    PUT ("Pass number: ");
    PUT (PASS);
    NEW_LINE;
    if PASS = 2 then
      DEAD_LOCK := TRUE;
      PUT ("Deadlock!!");
    end if;
    NEW_LINE (SPACING => 2);
    ONE.STARTUP;
    TWO.STARTUP;
    PUT_TIME;
    NEW_LINE;
    PUT_LINE ("Main program gets CPU and I/O");
    NEW_LINE;
  end loop;
  -- PASSES
end TEST_DEADLOCK;
pragma MAIN;
```

```
--  This program illustrates a large indivisible data
--    structure where the application program is not
--   concerned about address space boundries but forces
--   the Ada compiler/RTS to deal with them.
--

procedure LIDSA1(X: LONG_FLOAT);

procedure LIDSA1(X: LONG_FLOAT) is
      MAP : array (INTEGER range 1..32767) of LONG_FLOAT;
         --  array size > 64 K words => won't fit in
         --    one address space


begin
  for I in 1..32767 loop

    LIDSA1(MAP(I));

    -- This loop forces an address change somewhere in it.
    -- The compiler/RTS must generate a BEX that checks
    -- every time through the loop for an address space
    -- change.

  end loop;

end LIDSA1;

-- NOTE: This program adapted from an example in the
--       AIMS Interim Technical Report
--       (Avionics Laboratory, 1984:4-24)
```

```
  --  This program is a modified version of LIDSA1 where
  --    the application program takes care of address space
  --    boundries. It was modified to take advantage of
  --    where the start of the array is physically located.
  --

procedure LIDSA2(X: LONG_FLOAT);

procedure LIDSA2(X: LONG_FLOAT) is

       MAP : array (INTEGER range 1..32767) of LONG_FLOAT;
           --  array size > 64 K words => won't fit in
           --    one address space


  begin
    for I in 1..16384 loop

      LIDSA2(MAP(I));

      -- MAP(1) should be physically located at the low end
      --   of address space 1, so there won't be any context
      --   switching in this loop (so no BEX generated).

    end loop;

    LIDSA2(MAP(16385));
      -- Context switching done here (one BEX).

    for I in 16386..32767 loop

      LIDSA2(MAP(I));

      -- There should be no context switching at all in this
      --   loop since everything accessed is in address space
      --   2 (so no BEX required).

    end loop;

  end LIDSA2;

  --  NOTE: This program adapted from an example in the AIMS
  --        Interim Technical Report
  --        (Avionics Laboratory 1984:4-25)
--          his is the Prototype ACEC test ADDSA2 with the
--
--          EXCEPTION
--            WHEN NUMERIC_ERROR =>
--            Y := X;
--
--          lines  added.  When run times of this program  are
--          compared to those of ADDSA2, an indication of the
--          cost of an exception that is never invoked is
--          obtained.

  WITH INSTRUMENT; USE INSTRUMENT;
PROCEDURE EXCEP2 IS

  X  : FLOAT := 0.000_001;
  Y  : FLOAT := 0.0;

BEGIN
START("ADDSA3","ADDSA2 WITH NUMERIC_ERROR EXCEPTION  ");
  FOR I IN 1..1000 LOOP
    FOR J IN 1..10 LOOP
      Y := Y + X + X + X + X + X + X + X + X + X + X;
          -- included in test version
      Y := FLOAT(IDENT_INT(I+J));
    END LOOP;
  END LOOP;
STOP;
    EXCEPTION
      WHEN NUMERIC_ERROR =>
        Y := X;
END EXCEP2;
```

APPENDIX D

Test Harness Used On the VAX 11/780

This is the file harness_many.com. It was executed by entering "submit harness_many/parameters = (addsa.lst,[witt.acvtst])/noprint".

```
$!  This VAX/VMS command file loops through a file
$!    containing Ada source benchmark test file names and
$!    submits them to the test harness for the collection of
$!    the various statistics. For this implementation, this
$!    COM file must be submitted as a batch job.
$!
$!  The name of the file containing the test names is given as
$!    the first parameter to this command procedure.
$!
$!  The second parameter is the directory in which these tests
$!    must reside.
$!
$!
$!  Set the default Ada library
$!
$ acs set lib [witt.adalib]
$!
$!  Set the default directory to [witt.acvtst]
$!
$ set def [witt.acvtst]
$!
$!  Create the three statistic files
$!
$      create comp.dat
$      create instr.dat
$      create run.dat
$!
$!  Open the file with the test names
$!
$      open/read in_file 'p1'
$!
$!  Loop through the file of tests, submitting each test to
$!    the harness for the collection of the various data.
$!
$ loop:
$     read/end_of_file=done in_file test
$     @harness 'test' 'p2'
$     goto loop
$!
$!  At the end of the input file, close the file and
$!    terminate this command procedure.
$!
$ done:
$ close in_file
$ write sys$output "All tests have been submitted for testing"
```

This is the file harness.com which was executed from harness_many.

```
$!  This VAX/VMS command file performs functions necessary to
$!    collect various data about Ada source test files.
$!    These data are put into the files 'comp.dat'
$!    (compilation statistics), 'instr.dat'
$!    (instrumentation statistics) and 'run.dat' (run-time
$!    statistics).
$!
$!  Record the current elapsed and cpu times (before
$!    compilation)
$!
$ beg_cpu_time = f$getjpi("","cputim")
$ beg_time = f$time()
$!
$!  Compile and link'the test
$!
$ ada/nocopy_source 'p2''p1'
$ acs link 'p1'
$!
$!  Record the current elapsed and cpu times (after
$!     compilation)
$!
$ end_cpu_time = f$getjpi("","cputim")
$ end_time = f$time()
$!
$!  'file' => file_spec of the object file created by the
$!    compilation
$!
$ file = "[witt.adalib]" + p1 + ".obj"
$!
$!  Calculate the number of bytes in the object file
$!
$ blocks_used = f$file_attributes(file,"eof")
$ block_size = f$file_attributes(file,"bls")
$ file_size = blocks_used * block_size
$!
$!  Calculate elapsed cpu time ( in hundredths of seconds )
$!
$ cpu_time = (end_cpu_time - beg_cpu_time)
$!
$!  Divide the elapsed cpu time into seconds and
$!    hundredths-seconds
$!
$ cpu_time_secs = cpu_time / 100
$ cpu_time_hundsecs = cpu_time - 100 * cpu_time_secs
$!
$!  Put the compilation cpu statistics in an output line
$!    NOTE: I filled in the elapsed time stuff--not relevant
$ out_line = "''p1'                 " + -
  "''cpu_time_secs'.''cpu_time_hundsecs'       ''file_size'"
```

```
$!
$!
$!  Append the output line onto the file
$!
$ open/append comp comp.dat
$ write comp out_line
$ close comp
$!
$! Record the current cpu time (before execution)
$!
$ beg_cpu_time = f$getjpi("","cputim")
$!
$!  Run the executable file
$!
$ run 'p1'.exe
$!
$!  Record the current cpu time (after execution)
$!
$ end_cpu_time = f$getjpi("","cputim")
$!
$!  Append the instrumentation statistics to instr.dat
$!
$ append instr.; instr.dat
$!
$!  Calculate elapsed cpu time (in hundredths of seconds)
$!
$ cpu_time = (end_cpu_time - beg_cpu_time)
$!
$!  Divide the elapsed cpu time into seconds and
$!    hundredths-seconds
$!
$ cpu_time_secs = cpu_time / 100
$ cpu_time_hundsecs = cpu_time - 100 * cpu_time_secs
$!
$!  Put the available execution statistics in one output line
$!    NOTE: I filled in the elapsed time stuff--not relevant
$ out_line = "''p1'                 " + -
             "''cpu_time_secs'.''cpu_time_hundsecs'"
$!
$!  Append the output line onto the file
$!
$ open/append run run.dat
$ write run out_line
$ close run
$!
$!  Delete unnecessary files
$!
$ del instr.;*
$ del 'p1'.exe.*
$acs delete unit 'p1'
$!
```

APPENDIX E

Test Harness Used On the Data General

This is the file HARNESS_MANY.CLI:

```
CREATE HARNESS.OUT
CREATE INSTR.DAT
QBATCH/QOUTPUT=%1%.LOG/NOTIFY  [LINE CONTINUES]
 :ACCOUNTS:<acct number>:FILES:HARNESS_A %1%
```

It was executed by entering HARNESS_MANY TESTIN where TESTIN is the name of the file that lists the programs to be compiled and executed in the following manner:

```
                        (PROGR1 &
                         PROGR2 &
                PROGR3)
```

This is the file HARNESS_A.CLI. It's purpose is to limit the number of compiles or executions to just one at a time.

```
:ACCOUNTS:<acct number>:FILES:HARNESS [%1%]
```

This is the file HARNESS.CLI. It controls the actual compilation, execution, and statistic collection.

```
WRITE/L=%1%.STAT %1%
RUNTIME/L=%1%.STAT
ADA/MAIN_PROGRAM %1-%
ADALINK %1%
RUNTIME/L=%1%.STAT
PAUSE 15
FI/LENGTH/NHEADER/L=%1%.STAT %1%.PR
COPY/A HARNESS.OUT %1%.STAT
DELETE  %1%.HST  %1%.LOG  %1%+.OB  %1%.PR  %1%+.SR  %1%.STAT
DELETE  %1%+.STR %1%+.TREE
X %1%
COPY/A/1=WARNING/2=WARNING INSTR.DAT INSTR
DELETE INSTR
```

APPENDIX F

Compilation Results

TEST NAME	DEC COMPILER		DATA GENERAL COMPILER	
	CPU TIME	OBJECT CODE SIZE (BLOCKS)	CPU TIME	OBJECT CODE SIZE (PAGE SECTS)
ADDSA1	6.42	1024	10.06 *	881 *
ADDSA2	6.96	1536	10.05 *	879 *
ADDSA3	7.22	1536	10.03 *	878 *
AKERA2	7.20	1024	10.02 *	876 *
AKERA3	7.62	1024	10.00 *	875 *
MULTA1	6.36	1536	9.97 *	874 *
MULTA2	6.77	1536	9.94 *	872 *
MULTA3	7.21	1536	9.93 *	870 *
CHSSA1	10.00	4096	9.92 *	869 *
CHSSA2	10.20	4096	3.90 *	867 *
CHSSA3	10.68	4096	9.91 *	865 *
AOCEA1	12.46	2560	9.89 *	864 *
AOCEA2	12.45	2560	9.88 *	863 *
AOIEA1	12.69	3072	9.86 *	861 *
AOIEA2	12.63	3072	9.84 *	860 *
LOAEA1	12.21	2560	9.81 *	858 *
LOAEA2	12.23	2560	9.79 *	857 *
LOECA1	12.70	2560	9.76 *	855 *
LOECA2	12.28	2560	9.73 *	853 *
LOFCA1	12.78	3072	9.69 *	852 *
LOFCA2	12.66	3072	9.65 *	850 *
LONEA1	12.70	3072	9.50 *	849 *
LONEA2	12.57	3072	9.47 *	848 *
LOSCA1	14.80	4096	9.36 *	846 *
LOSCA2	14.88	4096	9.32 *	845 *
LOUIA1	15.43	3584	9.24 *	843 *
LOUIA2	15.40	3584	9.19 *	841 *
LOUSA1	12.90	3072	9.14 *	840 *
LOUSA2	12.11	3072	9.11 *	840 *
OPAEA1	13.19	3072	9.09 *	839 *
OPAEA2	13.17	3072	9.06 *	839 *
OPBFA1	COMPILE ERROR		COMPILE ERROR	
OPBFA2	COMPILE ERROR		COMPILE ERROR	
OPCEA1	14.17	3072	9.03 *	837 *
OPCEA2	14.21	3072	8.99 *	836 *
OPCFA1	14.10	3072	8.97 *	834 *
OPCFA2	14.40	3072	8.96 *	833 *
OPDSA1	13.38	3072	8.94 *	831 *
OPDSA2	13.64	3072	8.93 *	830 *

NOTE: Results marked with * are invalid

Compilation Results (Con't)

TEST NAME	DEC COMPILER		DATA GENERAL COMPILER	
	CPU TIME	OBJECT CODE SIZE (BLOCKS)	CPU TIME	OBJECT CODE SIZE (PAGE SECTS)
OPISA1	14.15	3072	8.91 *	829 *
OPISA2	13.94	3072	8.90 *	828 *
OPLEA1	14.44	3072	9.38 *	882 *
OPLEA2	14.29	3072	9.37 *	882 *
OPNFA1	12.15	2560	9.35 *	879 *
OPNFA2	12.31	2560	9.33 *	877 *
OPSCA1	15.51	3584	9.32 *	876 *
OPSCA2	14.93	3584	9.30 *	874 *
OPSEA1	13.78	3072	9.29 *	873 *
OPSEA2	14.30	3072	9.28 *	872 *
PUZZA2	38.83	14336	9.27 *	871 *
PUZZA3	38.78	14336	9.23 *	867 *
RENDA1	8.22	2560	9.22 *	866 *
RENDA2	8.19	2560	9.21 *	865 *
TPGTA2	9.11	3072	9.19 *	864 *
TPGTB2	8.86	3072	9.18 *	862 *
TPGTC2	15.80	6656	9.16 *	861 *
TPGTD2	14.94	6656	9.14 *	859 *
TPITA1	9.65	3584	9.12 *	858 *
TPITA2	9.86	4096	9.11 *	856 *
TPITB1	13.23	6656	9.09 *	855 *
TPITB2	14.94	7168	9.07 *	854 *
TPITC1	18.27	10240	9.06 *	852 *
TPITC2	21.32	11776	9.05 *	850 *
TPITD1	27.47	17408	9.03 *	849 *
TPITD2	35.63	20480	9.01 *	848 *
TPOTA2	17.53	8192	8.99 *	846 *
TPOTB2	17.52	8704	8.98 *	844 *
TPOTC2	49.80	21504	8.96 *	842 *
TPSTA2	8.73	3072	8.94 *	841 *
TPSTB2	12.93	5632	8.93 *	839 *
TPUTE2	10.49	4096	8.91 *	838 *
WHETA2	15.85	3584	8.90 *	836 *
WHETA3	16.79	3584	8.81 *	834 *

NOTE: Results marked with * are invalid

APPENDIX G

Sample of File HARNESS.OUT

```
ADDSA1
ELAPSED  0:00:04, CPU  0:00:00.196, I/O BLOCKS   45, PAGE SECS 14
ELAPSED  0:03:21, CPU  0:00:10.252, I/O BLOCKS  219, PAGE SECS 886
  =ADDSA1.PR        323584
ADDSA2
ELAPSED  0:03:51, CPU  0:00:10.499, I/O BLOCKS  311, PAGE SECS 909
ELAPSED  0:07:15, CPU  0:00:20.551, I/O BLOCKS  488, PAGE SECS 1844
  =ADDSA2.PR        323584
EXCEP2
ELAPSED  0:07:48, CPU  0:00:20.795, I/O BLOCKS  582, PAGE SECS 1867
ELAPSED  0:11:08, CPU  0:00:30.822, I/O BLOCKS  791, PAGE SECS 2809
  =EXCEP2.PR        323584
ADDSA3
ELAPSED  0:11:40, CPU  0:00:31.066, I/O BLOCKS  887, PAGE SECS 2832
ELAPSED  0:14:59, CPU  0:00:41.092, I/O BLOCKS 1065, PAGE SECS 3775
  =ADDSA3.PR        323584
AKERA2
ELAPSED  0:15:32, CPU  0:00:41.335, I/O BLOCKS 1158, PAGE SECS 3798
ELAPSED  0:18:55, CPU  0:00:51.353, I/O BLOCKS 1335, PAGE SECS 4739
  =AKERA2.PR        323584
AKERA3
ELAPSED  0:19:27, CPU  0:00:51.595, I/O BLOCKS 1432, PAGE SECS 4762
ELAPSED  0:22:54, CPU  0:01:01.592, I/O BLOCKS 1612, PAGE SECS 5702
  =AKERA3.PR        323584
MULTA1
ELAPSED  0:23:25, CPU  0:01:01.836, I/O BLOCKS 1718, PAGE SECS 5725
ELAPSED  0:26:42, CPU  0:01:11.803, I/O BLOCKS 1893, PAGE SECS 6662
  =MULTA1.PR        323584
MULT[illegible]
ELAPSED  0:27:14, CPU  0:01:12.045, I/O BLOCKS 1988, PAGE SECS 6684
ELAPSED  0:30:36, CPU  0:01:21.982, I/O BLOCKS 2166, PAGE SECS 7618
  =MULTA2.PR        323584
MULTA3
ELAPSED  0:31:09, CPU  0:01:22.222, I/O BLOCKS 2264, PAGE SECS 7641
ELAPSED  0:34:27, CPU  0:01:32.154, I/O BLOCKS 2444, PAGE SECS 8575
  =MULTA3.PR        323584
CHSSA1
ELAPSED  0:35:00, CPU  0:01:32.399, I/O BLOCKS 2556, PAGE SECS 8598
ELAPSED  0:38:33, CPU  0:01:42.318, I/O BLOCKS 2731, PAGE SECS 9530
  =CHSSA1.PR        323584
CHSSA2
ELAPSED  0:39:18, CPU  0:01:42.556, I/O BLOCKS 2824, PAGE SECS 9552
ELAPSED  0:42:39, CPU  0:01:52.453, I/O BLOCKS 3001, PAGE SECS 10483
  =CHSSA2.PR        323584
CHSSA3
ELAPSED  0:44:37, CPU  0:01:52.694, I/O BLOCKS 3100, PAGE SECS 10505
ELAPSED  0:48:33, CPU  0:02:02.606, I/O BLOCKS 3277, PAGE SECS 11437
  =CHSSA3.PR        323584
```

APPENDIX H

Execution Results

TEST NAME	DEC COMPILER CPU TIME	DATA GENERAL COMPILER CPU TIME
ADDSA1	0.93	0.17
ADDSA2	4.60	1.93
ADDSA3	4.30	1.93
AKERA2	1.60	0.25
AKERA3	1.70	0.25
MULTA1	3.84	1.72
MULTA2	4.40	2.06
MULTA3	4.11	2.06
CHSSA1	1.72	7.32
CHSSA2	11.54	32.90
CHSSA3	11.51	24.57
AOCEA1	1.84	1.15
AOCEA2	1.79	1.17
AOIEA1	1.89	1.15
AOIEA2	1.91	1.16
LOAEA1	3.51	2.11
LOAEA2	2.71	2.10
LOECA1	7.86	6.92
LOECA2	8.14	7.38
LOFCA1	1.79	0.84
LOFCA2	1.78	0.84
LONEA1	7.14	6.15
LONEA2	6.92	6.20
LOSCA1	13.32	12.34
LOSCA2	12.43	12.31
LOUIA1	7.54	6.93
LOUIA2	7.57	7.26
LOUSA1	7.12	6.08
LOUSA2	7.25	6.35
OPAEA1	0.99	0.38
OPAEA2	0.99	0.38
OPCEA1	1.13	0.42
OPCEA2	1.10	0.38
OPCFA1	1.19	0.50
OPCFA2	1.19	0.51
OPDSA1	1.4	0.49
OPDSA2	1.15	0.49
OPISA1	1.15	0.49
OPISA2	1.17	0.49
OPLEA1	1.38	0.67
OPLEA2	1.37	0.67
OPNFA1	0.83	0.18
OPNFA2	0.84	0.18

Execution Results (Con't)

	DEC COMPILER	DATA GENERAL COMPILER
TEST NAME	CPU TIME	CPU TIME
OPSCA1	0.80	0.19
OPSCA2	0.85	0.19
OPSEA1	1.33	0.67
OPSEA2	1.35	0.67
PUZZA2	8.42	11.65
PUZZA3	8.34	12.01
RENDA1	1.60	0.18
RENDA2	3.42	6.77
TPGTA2	3.37	11.48
TPGTB2	3.55	11.57
TPGTC2	3.76	12.11
TPGTD2	3.83	12.11
TPITA1	3.48	11.03
TPITA2	3.53	10.96
TPITB1	3.40	10.87
TPITB2	3.61	11.08
TPITC1	3.72	11.04
TPITC2	3.72	11.03
TPITD1	3.98	11.06
TPITD2	4.20	11.21
TPOTA2	19.60	54.70
TPOTB2	19.96	60.67
TPOTC2	152.31	431.38
TPSTA2	3.46	11.71
TPSTB2	3.48	15.71
TPUTE2	2.50	5.98
WHETA2	3.10	12.63
WHETA3	2.95	9.51
EXCEP2	4.55	1.94
LIDSA1	ERROR DURING LINK	COMPILE ERROR
LIDSA2	ERROR DURING LINK	COMPILE ERROR
TEST_DEADLOCK	UNDETECTED DEADLOCK	UNDETECTED DEADLOCK

VITA

Donald J. Witt was born on 27 December 1946 in Cisco, Texas. He graduated from Cisco High School in 1964 and enlisted in the Air Force in June 1964. His twelve year enlisted career included assignments in Japan, the Philippine Islands, Vietnam and at Keesler AFB, Mississippi, and Gunter AFS, Alabama. After receiving a Bachelor of Science Degree in Business Administration from Troy State University at Montgomery, Alabama, in 1976; he was commissioned through Officer Training School in October 1976. His first assignment as an Air Force officer was to the Manpower and Personnel Center at Randolph AFB, Texas. While there, he was instrumental in designing, coding, and implementing the Promotions and Placement Referral System of the Civilian Automated Personnel Data System. In December 1980 he was assigned to the 6002 Computer Services Squadron, Headquarters Pacific Air Forces (PACAF), Hickam AFB, Hawaii. His duty was as Chief, Analysis and Programming Division, with his primary responsibility being the transition of PACAF unique software under the guidelines of the base level capital replacement program, Phase IV. He left Hawaii when assigned to the Air Force Institute of Technology School of Engineering at Wright-Patterson AFB, Ohio in May of 1984.

Permanent address: Rt 1, Box 246
Grand Bay, Alabama 36541

APPENDIX I

E&V POSITION PAPERS

Table of Contents

SINGLE PROJECT/MULTIPLE APSES

JERRY BROOKSHIRE
TEXAS INSTRUMENTS

POSITION PAPER: SINGLE PROJECT/MULTIPLE APSES

BACKGROUND

The purpose of this paper is to propose a topic for extended consideration at the next annual APSE E and V Workshop, currently scheduled for July 1985. The proposed topic has many facets, but the fundamental underlying question is the concern for distributed APSEs and their communication mechanisms. Many of the issues and problems of conventional distributed processing are encountered here, plus a set of concerns unique to multiple APSEs.

It is apparent that most very large scale defense system developments will include components of very large scale mission critical software development requirements, requiring substantial concurrent design and development activies, potentially over a variety of target computers which have been selected for their functional characteristics as matched against desired/required system performance needs. It is to be expected that overall system operational requirements would often dictate a large volume of multi-way concurrent communications between the various functional nodes.

While the integration of such software developments into a cohesive effort would be a highly desirable and preferred goal, such a luxury will most often be infeasible if not impossible. The concern then becomes one of coordination, at least for those elements which, in the operational environmment, will have to be made aware of each other. What this in turn implies is a need to have (possibly) heterogeneous APSEs communicating with each other as a side effect during the software development phase of the project. When this situation is encountered, what are the special requirements for APSE E and V that result?

In the "moving-target" environment of a large-scale software system development, another ugly problem candidate presents itself, especially at this relatively early stage of the Ada language lifetime - the concurrent use of different Ada compilers for the different stages of a single system development. This same problem might be expected at the point of phase-over from a development-host based system to the development-station phase of the effort.

SINGLE PROJECT / MULTIPLE APSEs or NETWORKED APSEs

Another of the many circumstances that are to be encountered in the field of large-scale projects requiring the development of significant amounts of mission-critical software, one particular type stands out as representing special and unique problems for multiple-APSE evaluation and validation. This is the case where a large host computer is used for primary embedded software development, but the total effort also calls for the concurrent development of a deliverable work station/development station with some variety of APSE, to be used by the customer in the life-cycle software support for the project. This circumstance places a special emphasis on the above-mentioned communication problem early in the system development phase - how can the transition from large-scale multi-user host to smaller-scale, fewer-user development stations be made while insuring compatibility of the software products being so generated?

The special issues that this situation brings to E and V include:

* Original-host : Development-station processor compatibility?

* Should the Development station APSE be fully compatible with that of the large-scale host? If not, how would the minimum APSE subset be defined?

* Should these problem types be treated as two separate concerns, with two possibly independent APSEs and related E and V efforts? All of the interface/who-does-what questions are just postponed slightly by this position.

* One of the most significant aspects of all of the anticipated issues is the question of the database - its structure, its residence, and its distribution over the development network. How can a remote development station have access to the necessary information to support link- and run-time data confirmation, etc.?

* Development station selection/specification criteria: the MAKE/BUY decision processes for the system developer. If a good development station exists for the target, this may be a trivial question - but can such an eventuality be used to drive the selection of the target processors?

* The possibility of having different compilers on the host and on the supporting development station. At which level do we place the integration and test debugging; where to provide the simulation support; how to handle the potential incompatibilities of PRAGMAs and target-processor dependent representation specifications.

* Assuming at some stage an established working relationship between the original development host computer, and the used-destined development station. Should the host be accessed by the workstation in order to provide the user with a consistent

interface?

PROPOSED APPROACH - ISSUE STUDIES

If this proposal is accepted as a topic for the workshop, we would propose the development of a set of studies based on the refined and final issues to be delineated as a result of the workshop. The most direct way of conducting such studies would seem to be in conjunction with preparations for large-scale mission-critical software developments, such as the TI (Army) LHX development, just underway. Other projects would have even more source for support, as they would initially involve two or more major undertakings by different organizations/contractors (eg., WIS and SDI).

APPENDIX A

REFERENCES

NOTE The following references are listed in alphabetical order by ascending year

[TI77A] Equipment Group Software Development Guide (SF24-EG77), December 1977

[TI77B] Equipment Group Software Management Standard (SF23-EG77), December 1977

[TI80A] Model 990 Computer TI Pascal User's Manual, Texas Instruments, Part 946290-9701, 15 January 1980

[TI80B] The Microprocessor Pascal System (User's Manual), Texas Instruments, 1980.

[TI80C] Survival Kit For Software Producers, Texas Instruments, March 1980.

[MS80A] Sixteen-Bit Computer Instruction Set Architecture, MIL-STD-1750A, 2 July 1980.

[SOF81A] SofTech Inc., "Draft Ada Language System VAX-11/780 VAX/VMS Runtime Support Library B5 Specification," Waltham, MA, July 1981.

[SOF81B] SofTech Inc., "Preliminary Draft Ada Language System KAPSE B5 Specification," Waltham, MA, August 1981.

[TI81A] Texas Instruments, "Ada Integrated Environment," Lewisville, TX, March 1981. Prepared for Rome Air Development Center (RADC) under DoD Contract F30602-80-C-0293.

[TI81B] Equipment Group Programming Standards for Computer Programs, Advanced Computer Systems Laboratory, Texas Instruments, July 1981.

[NOSC82A] Kernal Ada Programming Support Environment (KAPSE) Interface Team Public Report Volume 1, Naval Ocean Systems Center San Diego, Ca, 1 April 1982.

[NOSC82B] Kernal Ada Programming Support Environment (KAPSE) Interface Team: Public Report Volume 2, Naval Ocean Systems Center San Diego, Ca, 28 October 1982.

[SOF82] SofTech Inc., "Draft Ada Language System Specification," Waltham, MA, August 1982.

[TI82A] Texas Instruments, Advanced Computer Systems Laboratory, "Proposal for Development of Ada Software Tools and Interface Standards," Lewisville, TX, February 1982.

[TI82B] Texas Instruments, Advanced Computer Systems Laboratory, "A Training Program for Ada: Issues and Motivations", Lewisville, TX, 29 July 1982.

[MS83A] Nebula Instruction Set Architecture, MIL-STD-1862B, 3 January 1983.

[AJF083] Evaluation and Validation (E and V) Test Plan, Version 1.0, November 1983

[BOO83] Booch, Grady, Software Engineering With Ada, Benjamin/Cummings Publishing Co, Inc. Menlo Park, Ca, 1983.

[DAY83] JD Day and H. Zimmerman, "The OSI Reference Model", Proceedings of the IEEE, v.71, n.12, December, 1983, pp 1334-1340

[HOU83] "A Taxonomy of Tool Features for the Ada Programming Support Environment (APSE)", U.S. Department of Commerce, National Bureau of Standards, December 1982, Issued February 1983

[TI83A] Texas Instruments, "AIM (APSE Interactive Monitor) Program Performance Specification," Lewisville, TX, 19 September 1983. Prepared for Naval Ocean Systems Center (NOSC) under contract _ N66001-82-C-0440.

[TI83B] Texas Instruments, "AIM (APSE Interactive Monitor) Interim Report on Interface Analysis and Software Engineering Techniques" Lewisville, TX, 16 May 1983 Prepared for Naval Ocean Systems Center (NOSC) under contract _ N66001-82-C-0440.

[TI83L] Texas Instruments, "APSE Interactive Monitor (AIM) Interim Report on Interface Analysis and Software Engineering Techniques II (IR2)", Contract _ N66001-82-C-0440. 01 December 1983.

[CAIS84] KIT/KITIA CAIS Working Group, "Proposed Military Standard Common APSE Interface Set (CAIS)", 01 August 1984

ADA PROGRAM LIBRARY SYSTEMS

THOMAS LEAVITT
BOEING

POSITION PAPER

Ada Program Library Systems

After the compiler proper, the most important Ada Programming Support Environment (APSE) design issue is the program library system. The library system provides the structure in which operations take place. Its major properties are not revealed by listing the explicit library management functions that can be performed. Most of the references to the library will be implicit with the operation of the compiler: such as finding "with"ed units and the parents of subunits; or storing objects. However, the library system is not part of the compiler. Libraries exist independent of a particular compilation, and there are operations performed on them not related to compilation, like deletions of objects, and status inquires.

With validated Ada systems, running the ACVC test suite shows that it is possible to get the separate compilation facility to work. This is a minimal demonstration of sufficiency of a library system and does not imply that the system is easy to use, or even practical to use in a production environment. Various suppliers have taken very different approaches to library system design. When evaluating an APSE, it is important to consider how the library system will affect the program development process. Even with the limited experience to date, it has been clear that some library systems are awkward to use, and would be impractical on all but small projects.

The criteria for evaluating a library system are somewhat subjective. The criteria include ease of use, effort required to learn, economy of use of disc storage, elegance, efficiency of operations, robustness of operations, and adaptability to project requirements. A good library system should encourage controlled sharing. It must be easy to keep modules private and to share them. A programmer constructing a test version of a program should not impact other users of the library until explicit steps are taken to make the updated version(s) visible to other users. When a shared unit is updated, all the affected users should see the new version without requiring any special action; in particular, it should not be necessary to send a message to every user telling them to copy the new version into their private libraries.

A good way to evaluate the library system of a proposed Ada system is to require each supplier to respond with how their system will operate to perform a list of scenarios. Each scenario would be rated with respect to ease of use, and a minimum level determined. Systems which do not perform up to the minimal level would be rejected (or at least be noted as unacceptable without repair). Care should be taken that the minimum standards do not unnecessarily disqualify off-the-shelf systems which are workable if unpleasant. When actually procuring an APSE it is not desirable to find that the lowest cost bidder, who has proposed what would have been an acceptable system, has been disqualified, or worse that all bidders have been disqualified.

Requesting such a list is important. Experience has shown that trying to determine the capability of a suppliers library system from a less specific request for information is frustrating. Proposals can be very unclear, probably in part because the suppliers are so familiar with their systems that they think their descriptions are obvious: they know what the descriptions mean, but readers unfamiliar with the system can find the wording extremely cryptic. The main issue to consider is that the system is straightforward and simple to use. Some consideration should be given to other issues like space usage, similarity to existing systems familiar to users, and execution time overheads.

The following list of thirteen operations should be stated in an RFP and each supplier would respond with a description of how their system would perform them.

1. Create a new main program 'M' which "with"s a preexisting unit 'A', where 'A' is shared by several programs and is visible to several accounts. The new program 'M' is to be local in visibility to the creating account. Outline the steps to be taken to edit, compile and execute the program.

   As in STONEMAN 4.A.4, every version in a library should be accessible. The suppliers need to explain how this is done. In particular, where units are named by "with" clauses, how do programmers indicate which versions and variations are desired? There are at least three ways this might be done: (a) by a directory structure and associated search strategy such that programmers can place objects in the directory structure where they will be found before other versions or variations of the object (this might work implicitly through host OS directory structure, or explicitly by having the programmer provide a list of directories as a compiler parameter); (b) by some form of JCL or compiler options which specify for each Ada unit name the identity of the version and variation to use; or (c) by forcing the programmer to copy a version of the desired object into a local structure before invoking the compiler.

2. Create a new library unit 'C', which "with"s no other units. Outline the steps to be taken to compile the program and place it into a local library so that it could be "with"ed by another unit at a later time.

3. Consider a main program 'M' which "with"s units 'A' and 'B', which are all globally visible. Outline the steps a programmer would follow to create a test version of the program which uses a modified version of the library unit 'A'. Assume that the only changes are to the body of 'A', not to the specification: this should permit the user to avoid recompiling 'M' in some cases and these steps should be described in the proposal. Where the specification and body of 'A' are separate (i.e. there is a subunit for the body of 'A') and the specification is unchanged, recompilation of 'M' should not be required unless the body contains a subprogram specified "inline". An optimizing compiler might note when specifications are unchanged even when the body is not a subunit, and so avoid obsoleting 'M'. The modified library unit 'A' should be local, and so should the resulting executable program. This new executable program version must not obsolete the preexisting version visible to other users. The modifying programmer must be able to access both the new local version and the preexisting external version. The supplier should describe how this is done.

4. Consider the situation similar to 3. above when, after being convinced of the correctness of the modification, the global version of 'A' is to be updated. When done, is any notification given about units which depended on 'A' and are now obsolete? Or is such notification only given when a reference to an outdated unit is made?

5. Outline what steps would be taken to see if a unit is "current": that is, if it has been compiled more recently than the units on which it depends and they in turn are also current. If versions or variations are supported, how does the user identify the ones to use when integrating the library system? It is not tolerable to recompile a unit just to see if the compiler generates any "obsolete unit" error messages.

6. For a program 'P', for each unit, list all other units which depend on the unit. Also, for each unit, list the units which it depends on. The context of the program is important, since with units shared between programs, the programmer could be swamped by extraneous unit names. A popular unit such as a math library may be used in hundreds of different programs.

In a similar manner, it is desirable to list all the units which directly or indirectly depend on a specified unit, within a program. Such a list indicates the units which would be made obsolete if the specified unit were modified.

Such lists of dependent units might be provided by annotations in a link map or by a separate library inquiry utility.

7. Delete a unit 'A' from the library. If there are other units which depend on 'A', does the system indicate at this time that units have been obsoleted, or does it wait until programmers reference the obsoleted units?

STONEMAN requirement 4.A.6 calls for a mechanism to insure that "all the database objects needed to recreate a specified object will continue to be maintained in the database as long as the specified object itself remains in the database". If such a mechanism is provided, how will the system respond to an improper delete request? In particular, will it list the units which depend on the specified unit?

8. List the items in a library and their properties. The minimal properties include: names: disc space occupied by the items: the Ada unit names associated with the items (also the operating system source file names should also be given since an Ada compilation can contain several Ada compilation units, and the name of the source file need not be related to any of the contained units): the time of creation or last update: and the time of last reference.

9. Trace back a machine address to the corresponding Ada source text. Addresses may come from target machine assembly level debuggers, post mortem dumps, hardware performance monitors, or test panel operations. The expected mode of operation will be to refer to a link map to isolate to an Ada unit, and if finer resolution is required, use a compiler option to produce a assembly code listing. It is desirable for the link map provided to isolate address ranges to the Ada subprogram name level.

even if these names are not visible outside the units they are declared in.

This may seem to reflect a linker requirement rather than a library management system issue. However, many APSE's have integrated linking into the compilation process and all linking may be done as a side effect of compiling the main program, making linking a compiler/library issue.

This conflicts somewhat with STONEMAN 2.B.7 which encourages the use of Ada source terms, rather than assembly or machine level terms. However, it is doubtful that APSE's which will become available in the near future will be able to satisfy this STONEMAN goal. Some may argue that the availability of a source level debugger will remove the nee' for mapping from addresses to source. However, if the debugger requires any information in the object code to identify unit names or line numbers, or if the use of the debugger involves a compiler option which produces any different code (e.g. inhibits some optimization), the user is still faced with the possible problems of debugging a program which works when the "DEBUG" compile option is specified, but does not work when it is not specified.

10. Copy a set of items in a library to another library.

It must be possible to create an ANSI interchange tape of source texts to ship to another site which may be running an APSE of a totally different design philosophy. It is highly desirable to be able to transfer intermediate objects to other sites which run the same APSE so that it is not necessary to recompile everything. It is also desirable to be able to transfer objects between library systems in different accounts on the same host system.

11. Some systems support versions of items. If the system supports them, outline what steps are taken to restore a library to a prior version. In particular, is this any different than creating a new version of source of a unit, which happens to correspond to the text of a prior version, and compiling it?

12. For a particular executable program it is necessary to be able to determine the identity of the units from which it is constructed. This includes the version and variation of the unit and the date of each unit's creation or last modification.

This information may be provided on a link map. It should be possible to retrieve this information without relinking, since users may have deleted some of the objects used in the creation of the executable.

13. How much time does it take for typical users familiar with other high level languages and with the Ada Language Reference Manual (LRM), to learn to do all of the operations listed above from the user guide and other supplied documentation?

It is not sufficient that the system developers can operate their own system. It must be assured that programmers using the system can make it work. It should not be necessary for the supplier to provide hands on training for users to discover how to effectively use a system. The issues involved relate to the clarity of the documentation and to the complexity of the system.

This is an important point. A proposal for a system which supports a required capability but does not provide sufficient documentation to enable users to invoke it is not acceptable as it stands. Similarly, a function in an APSE which serves no immediately obvious purpose should be explained.

Although a long learning time is not desirable, a successful APSE will be used for a long time on several projects allowing a programmers learning time to be amortized over years of use. Even then, unnecessary complexity will complicate operations. If programmers do not understand a system, they will not use it well.

As in STONEMAN 3.C, simplicity is desirable. A good design will use the minimum number of additional concepts which still satisfy all requirements. New arbitrary operations and unclear utilities should be avoided if possible.

It is important that an Ada program library system be easy to use and powerful enough to provide support for the program development process. By requiring suppliers to describe how to perform a number of specific operations it is possible to determine how easy programmers will find their systems to use. It is required that the library system support controlled sharing. Even though a system which doesn't share might be made very easy to use it would not satisfy project requirements. Projects need to have programmers create test versions of a system without impacting other users, and programmers should pick up the most current version of shared units. In all cases, users need to specify what units to use. These are some capabilities associated with a library system which will facilitate use but which may be packaged in several ways - either as separate tools, as part of a linker, or as part of the compilation system proper. No matter where they are packaged, they form an important part of the operations of an APSE and are critical to the evaluation of an APSE. These capabilities include: dependency list creation; identification of sources (including version identification and creation dates) which went into the construction of an executable program; currency checking; and the user documentation of the system.

SECURITY IN APSE

M.B. SURY and E.W. MARTIN
LOCKHEED MISSILES AND SPACE COMPANY

SECURITY IN APSE

M. B. SURY and E. W. MARTIN
Lockheed Missiles And Space Company
AUSTIN Division, Austin Texas

1.0 POSITION STATEMENT

SINCE software security is an essential feature of mission critical applications, the Evaluation and Validation (E&V) Task should include software security considerations in evaluating Ada* Programming Support Environment (APSE) Components and interfaces.

Section 2 elaborates on the above position statement along with supporting arguments. A brief overview of software security is also included. Section 3 presents an overview of the approaches to security in the proposed Common APSE Interface Set (CAIS), Ada Integrated Environment (AIE), Ada Language System (ALS), and ALS/NAVY (ALS/N). Impacts on E&V Requirements are briefly discussed. We conclude with the observation that the access control mechanisms described in CAIS Version 1.4 are supportive of the various security needs and recommend that validation of an APSE be contingent on conformance to CAIS in this regard.

2.0 BACKGROUND AND DISCUSSION

2.1 Relevant Security Concepts

Security can be described in general terms as the protection of valuable assets. It is a recognized fact that software is a valuable asset having a direct value (such as proprietary material) as well as an indirect value (such as in the case of software which controls a mission critical system). Mission critical (Tactical) systems have stringent requirements for information security and system availability. Specific measures are required to ensure the privacy, reliability and integrity of the system - while it is being developed as well as after it is operational. The software must resist penetration, malicious damage and possible takeover. There is a need to protect software from unauthorized modifications and destructions, whether accidental or malicious. Two proven approaches to prevent unauthorized access are (i) isolation and (ii) mediation. Isolation means separating the object

* Ada is a Registered Trademark of the Department of Defense, Ada Joint Program Office.

being protected from its threat. Mediation means that a protected object can be accessed only via a mediating entity called the "reference monitor". When a user (or a process) wishes to access some data, the user makes an "access request" to the reference monitor after properly identifying himself. The reference monitor verifies that the user is allowed the requested access and if all the necessary checks have been successful, then - and only then - the request is granted. The reference monitor uses a prespecified authorization policy to determine a user's capabilities and access permissions. It is to be noted that a "user" in this context, may be a person in an interactive session with a computer or a process within one computer acting on behalf of a user or a computer communicating with another.

We classify security concerns into two categories: (1) those that are related to the development environment and (2) those related to the operational environment.

We observe that the software to be protected involves both data and the routines (modules). We believe that the privacy and integrity of the data (as opposed to the routines) are more critical during operations than during development. Thus, during development we emphasize the protection of the code more than the test data. This is because an unauthorized modification of the software might be the insertion of a "trojan horse", which makes the software behave unpredictably. The concern is compounded by the possibility that the unpredictable behavior of the software might happen at the most inopportune moment.

Mission critical applications involve the basic functions of collection, processing, distribution and storage of data. However, these functions (collection etc.) might be distributed over a wide geographical area, involving a multitude of processors of different types connected by various types of communication links. Furthermore, these functions have to be performed more securely, more reliably, in shorter durations and in hostile environments.

In either of the environments - development and operations - certain issues arise due to the distributed nature of the processing. Distributed processing is supported by APSEs and has the benefits of improved performance, protection through physical separation and confinement of sensitive information within selected boundaries. However, it should be noted that security controls on any one node in the network become irrelevant if the information is available to other hosts that do not have effective controls. Thus, security in a distributed system is only as strong as its weakest link.

2.2 Security Issues In The Development Setting

The complex software needed for today's tactical systems is usually developed by several vendors in a combined cooperative effort. Besides the usual issues of integration, such an environment raises sensitive security issues. To provide accountability, maintainability, privacy, and integrity of the software, standard physical and administrative security controls need to be imposed in a way similar to a data processing environment.

The development environment needs to be secured using standard physical and administrative controls. All changes to development software should be traceable and the tracking should be automated. The measures to ensure functionality of the software can also ensure the integrity and thus the security of the software. We need to adopt good software engineering practices. Object oriented design is one such process that supports the principles of information hiding, traceability, and maintainability. We need to verify also the design and code of the developed software to ensure that the software behaves as it is expected to. Towards this, program verification techniques might be employed using tools such as the Gypsy language. To control the costs, these techniques may be applied to verify only the critical parts of the software (e.g. KAPSE) rather than the entire software.

Tools are needed to prevent the penetrator from (i) exploiting known vulnerabilities, (ii) recovering information from the system, (iii) subverting security controls, (iv) denying service to legitimate users, and (v) reprogramming the operating system to do similar functions. The system must provide a privileged set of commands that can be executed only by well-controlled "critical" processes. Most users and user-processes will not have access to these privileged instructions.

The system must prevent users from directly accessing or executing critical functions such as resource allocation, maintenance routines, utilities, and data transfer control. The system should have exclusive control over these critical functions and it should be able to detect and report to the system security officer any unauthorized attempts to execute any of the above. The system should be able to detect all error conditions that might jeopardize the system's integrity.

Rigorous procedures should be implemented to identify and authenticate users and remote terminals. These access checks would be performed at least once at the beginning of a session and might be repeated (or augmented by additional checks) several times during the session.

2.3 Security Issues In The Operational Setting

Data has to be protected while it is stored (dormant) in a processor and while it is being processed. Data resident in a processor might influence future transactions and user-decisions and so the integrity and privacy of this data is very critical.

Unauthorized access to mission critical data might be passive (mere retrieval of data) or active (modification of data). Passive access results in a loss of privacy and active access results in a loss of integrity. This loss of integrity might adversely influence future transactions. Access attempts (of both passive and active types) can be thwarted by ensuring that the operating system performs frequent integrity checks on the data. Furthermore, stringent identification and authentication measures should be implemented for database updates.

The level of access control imposed on change-requests should vary (and be commensurate) with the level of software in the system that is being changed. For example, changes to the operating system will be allowed only to the super-user and only from the system console located at a predetermined, physically secure location and only after the user passes stringent authentication procedures. All changes to the operating system should be recorded on a hard-copy or a non-erasable medium.

2.4 Ada Programming Support Environment (APSE)

In the previous subsection, the need for security in the development and operational environments has been detailed. This section discusses the APSE approach to security.

The purpose of APSE is to provide an environment for the design, development, documentation, testing, management, and maintenance of embedded computer software written principally in the Ada Programming Language. In particular, the management and maintenance aspects warrant integrity, reliability, robustness, and correctness of the software. Major emphasis in the current work on APSE has been on increasing the inter-operability and user-friendliness of the software. Inter-operability makes an application more user-friendly, but it creates a security vulnerability in terms of unauthorized (uncontrolled) use of the software. Also, increased access potential implies increased exposure to threats, and this warrants a need to place more trust in the software during the design and development phase. A "trojan horse" might be incorporated during the early development of the mission critical software which goes unnoticed and which might surface in an entirely unexpected and unsuspected manner. This might be triggered by an unusual combination of the events such as voltage surges and memory overloads. Such trojan horses can be best caught by a close scrutiny of the developed code by a group of individuals that has the least chance of collusion. This means that the APSE has to support the policing of the use of the environment.

There are several features of the current APSEs that support the auditing, monitoring or simply organizing development activities. By supporting good software engineering practices, Ada and APSEs can support and contribute to good security.

It is our position that these features be given at least as much value as those that make the environment user-friendly. For example, the auditing features would discourage the occasional perpetrator and at the same time can also aid the detection and isolation of undesirable events such as maintenance problems.

Some of the features that would aid in monitoring the code development are: (i) Capabilities to identify the user and the module; (ii) Logging of all activities on critical objects (i.e. modules critical to the success of the mission); (iii) Analysis and Data Reduction techniques to be applied to the audit reports; (iv) Query capability to investigate patterns of unauthorized access; and (v) Negative testing (where the software is tested to check that it does not do what it is not supposed to).

These features are included in the discussion on Access control in the recent CAIS Document (Revision 1.4). Further discussion on current work on APSEs is given in the next section.

3.0 CURRENT STATUS

Based on a review of CAIS, ALS, ALS/N, E&V Requirements draft documents, it is seen that CAIS (Rev. 1.4) had the most comprehensive consideration of security. Following is an overview of the security discussion in each of these efforts.

3.1 CAIS

Section 4.4 of the CAIS document discusses access control in detail and meets all the needs briefed in section 2.1. In particular, for each unauthorised access request (whether accidental or not), an Ada exception is raised. By reviewing the audit log (that provides the details of this activity), the system administrator can determine whether there is a threat to the software and possibly the extent of the threat.

One aspect that needs elaboration here is the capability to change the CAIS_Access_Control Package. It is our recommendation that only the system administrator be able to modify this package and that all changes be logged on a non-erasable medium.

3.2 Ada Integrated Environment (AIE)

The AIE database is a collection of objects that have attributes and content. Some of these objects, called "window objects", provide a cross reference to a partition of another object in the database. This mechanism permits access to relevant parts of the database to authorized users and processes. The attributes of an object that are system defined include access control information (as defined in STONEMAN).

Independent and modular program development is supported by the Program Integration Facility (PIF). The PIF provides functions to allow the compiler to access the program library during the processing of single compilation units. Security concern in this case is that the units in the library might have been tampered with. This would be a possible entry for a trojan horse.

AIE has a MAPSE Command processor that allows commands to be interrupted and restarted by the user. This can be used by the System administrator to selectively terminate the sessions of unauthorized users before substantial damage is done.

3.3 Ada Language System (ALS)

The ALS environment database is self contained and independent of the host file system. It stores all the project-relevant information and allows the tracking of changes to it. Access control in ALS is provided by setting attributes to objects in the database. Initially, when an object is created, the creator and everyone in his "team" have privileges to read and execute. The creator can then set additional restrictions.

3.4 Ada Language System/NAVY Description (ALS/N)

The ALS/N is implemented as an extension of ALS and it includes a functional area for user access support. Included in the program support environment are configuration management, report generator, and inter-host telecommunications interface. These tools can be used by the system administrator to monitor system activity, detect unauthorized accesses, and take corrective actions.

3.5 E&V Requirements.

The E&V Task Force adopted the Taxonomy of tool features for APSE and divided an APSE into Components and Interface sets. Among the components that could help enforce security (e.g. by monitoring the system activity), are the Command Language Interpreter (CLI), the Configuration Management (CM), Control Flow Static Analyzer, Set-Use Static Analyzer, and the Dynamic Analysis Tool. There are four interface sets (CAIS, Ada Packages, Inter-Tool Data interface, and User-to-APSE interface). The E&V requirements (as detailed in October 1984 issue) do not emphasize following security related attributes of components: integrity, reliability, robustness,and correctness. We recommend the inclusion of these attributes among those to be evaluated for the Compiler and CLI.

Security falls in the Category A of APSE E&V Categories since no standard for the security of an APSE component is available and no technique for evaluating conformance has been developed. In other words, the security of a component needs to be evaluated subjectively. In this context, the E&V Team needs to interface with the DoD Computer Security Center. Our proposal is that the KAPSE part of an APSE should be a trusted system as per the Trusted Computer System Evaluation Criteria.

4.0 CONCLUSION

We presented an overview of the security needs in the development and operations of mission critical systems. We feel that the current work on APSEs is not providing adequate emphasis on software security. We believe that the access control mechanism described in the proposed CAIS document (Version 1.4) can serve as a preliminary standard for evaluating the security (i.e. robustness, correctness, integrity, and reliability) of the software in the APSE Components and Interface sets.

*U.S. Government Printing Office: 1986 646-067/40923