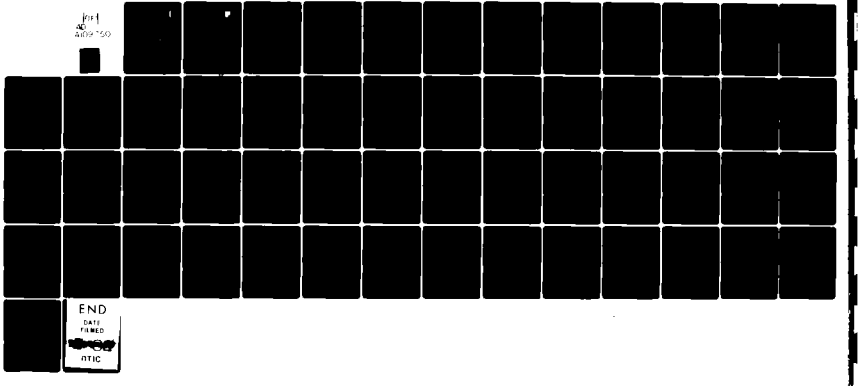
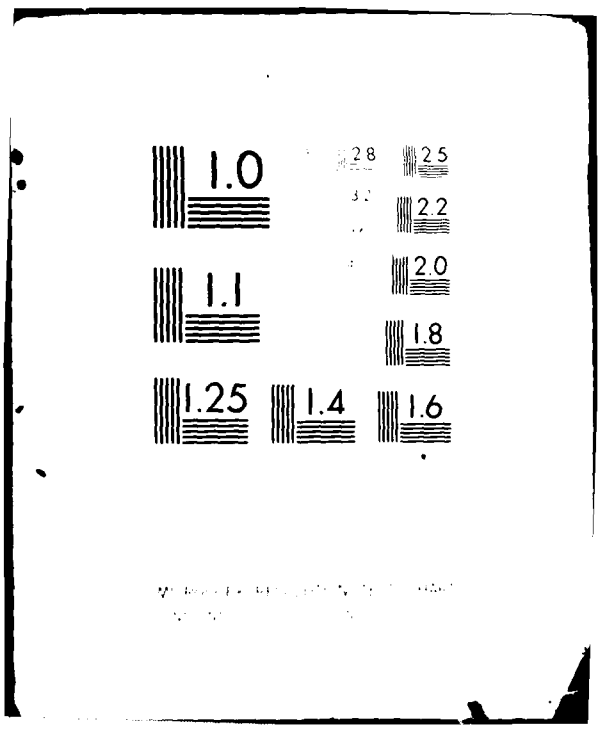
AD-A109 750 TEXAS INSTRUMENTS INC LEWISVILLE EQUIPMENT GROUP F/G 9/2
ADA INTEGRATED ENVIRONMENT III. SYSTEM SPECIFICATION.(U)
DEC 81 F30602-80-C-0293
UNCLASSIFIED RADC-TR-81-359 NL




1.0
2.8
2.5
2.2
1.1
2.0
1.8
1.25
1.4
1.6

PHOTOGRAPH THIS SHEET

AD A109750

DTIC ACCESSION NUMBER

LEVEL

INVENTORY

Texas Instruments, Lewisville, TX Equipment Group - ACSL

ADA Integrated Environment III System Specification Dec. 81

DOCUMENT IDENTIFICATION

Contract F30602-80-C-0293 RADC-TR-81-359

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

DISTRIBUTION STATEMENT

ACCESSION FOR	
NTIS	GRA&I
DTIC	TAB
UNANNOUNCED	
JUSTIFICATION	
BY	
DISTRIBUTION /	
AVAILABILITY CODES	
DIST	AVAIL AND/OR SPECIAL
A	

DISTRIBUTION STAMP

DTIC
ELECTE
S JAN 19 1982 D
D

DATE ACCESSIONED

DATE RECEIVED IN DTIC

PHOTOGRAPH THIS SHEET AND RETURN TO DTIC-DDA-2

DTIC FORM OCT 79 70A

DOCUMENT PROCESSING SHEET

RADC-TR-81-359
Interim Report
December 1981

ADA INTEGRATED ENVIRONMENT III SYSTEM SPECIFICATION

Texas Instruments Incorporated

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-81-359 has been reviewed and is approved for publication.

APPROVED:

ELIZABETH S. KEAN
Project Engineer

APPROVED:

JOHN J. MARCINIAK, Colonel, USAF
Chief, Command and Control Division

FOR THE COMMANDER:

JOHN P. HUSS
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (COES) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document requires that it be returned.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: RADC-TR-81-359	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): ADA INTEGRATED ENVIRONMENT III SYSTEM SPECIFICATION	5. TYPE OF REPORT & PERIOD COVERED: Interim Report 15 Sep 80 - 15 Mar 81
	6. PERFORMING ORG. REPORT NUMBER: N/A
7. AUTHOR(s)	8. CONTRACT OR GRANT NUMBER(s): F30602-80-C-0293
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Texas Instruments/Equipment Group-ACSL, P O Box 405, M.S. 3407, Lewisville TX 75067	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 62204F/62702F/33126F 55811919
11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (COES), Griffiss AFB NY 13441	12. REPORT DATE: December 1981
	13. NUMBER OF PAGES: 43
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office): Same	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Elizabeth S. Kean (COES)

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Ada	MAPSE	AIE
Compiler	Kernel	Integrated environment
Database	Debugger	Editor
KAPSE	APSE	

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

The Ada Integrated Environment (AIE) consists of a set of software tools intended to support design, development and maintenance of embedded computer software. A significant portion of an AIE includes software systems and tools residing and executing on a host computer (or set of computers). This set is known as an Ada Programming Support Environment (APSE). This system specification describes the basic design for a minimal APSE, called a MAPSE. The MAPSE is the foundation upon which an

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

APSE is built and will provide comprehensive support throughout the design, development and maintenance of Ada software. The MAPSE tools described in this specification include an Ada compiler, linker/loader, debugger, editor, and configuration management tools. The kernel (KAPSE) will provide the interfaces (user, host, tool), database support, and facilities for executing Ada programs (runtime support system).

TABLE of CONTENTS

SECTION 4 QUALITY ASSURANCE PROVISIONS

APPENDIX A GLOSSARY

LIST of TABLES

LIST of FIGURES

SECTION 1

SCOPE

1.1 Scope

This specification establishes the performance, design, development and test requirements for the Ada Integrated Environment software system.

SECTION 2

APPLICABLE DOCUMENTS

The following documents form a part of this specification to the extent specified herein. Unless stated otherwise the issue in effect on the date of this specification shall apply.

2.1 Program Definition Documents

[DoD80A] Requirements for Ada Programming Support Environments: "STONEMAN", DoD (February 1980).

[RADC80] Revised Statement of Work for Ada Integrated Environments, RADC, Griffiss Air Force Base, NY (March 1980).

[SOFT80A] Ada Compiler Validation Capability: Long Range Plan, SofTech Inc., Waltham, MA (February 1980).

[SOFT80B] Draft Ada Compiler Validation Implementers' Guide, SofTech Inc., Waltham, MA (October 1980).

2.2 Military Specifications and Standards

[DoD80B] Reference Manual for the Ada Programming Language: Proposed Standard Document, DoD (July 1980) (reprinted November 1980).

SECTION 3

REQUIREMENTS

3.1 System Definition

This section defines and provides a general description of a retargetable, rehostable Ada compiler and selected support software to create a minimal integrated software development and support facility for the Ada language. The Ada Integrated Environment specified here will include software in four major functional areas (system segments). Each system segment is further designated a Computer Program Configuration Item (CPCI) and contains several Computer Program Components (CPCs). The four system segments and related CPCs are shown in Table 3-1.

Table 3-1 System Segments and Computer Program Components

Segment 1.	ADA SOFTWARE ENVIRONMENT	C01
o	Execution Environment	C01.01
o	APSE Manager	C01.02
o	Program Management	C01.03
o	Task Management	C01.04
o	Storage Management	C01.05
o	Input/Output	C01.06
Segment 2.	ADA DATABASE SUBSYSTEM	C02
o	Database Directory Manager	C02.01
o	Database Dictionary Utility	C02.02
o	Database Archive Manager	C02.03
o	Database Utilities	C02.04
o	Database Interfaces	C02.05
o	User Information Manager	C02.06
o	User Information Utility	C02.07
o	Host Filesystem Interfaces	C02.08
Segment 3.	ADA LANGUAGE PROCESSOR	C03
o	Analyzer	C03.01
o	Optimizer	C03.02
o	Code Generator	C03.03
o	Library Utility	C03.04
Segment 4.	ADA PROGRAMMING TOOLSET	C04
o	General Text Editor	C04.01
o	Program Binder	C04.02
o	Source-level Debugger	C04.03

3.1.1 General Description

An Ada Programming Support Environment (APSE) is a collection of software tools which, when installed on a suitable host computer, provides facilities for the design, development, maintenance and management of software for one or more target computers. Generally, an APSE must support Ada applications software throughout its entire life cycle, with particular emphasis given to embedded computer applications. The Ada Integrated Environment APSE shall support itself and related Ada software tools on the host computer and must also support its own extension, retargeting and rehosting on selected other computers.

The architecture of an Ada Integrated Environment, installed on a host machine, will implement the STONEMAN concept of a layered APSE.

* The host machine, including input/output devices and mass storage, is the nucleus of the structure. It is convenient to view the host filesystem (database) at this level also.

* The native host operating system is the innermost software layer. Control of all hardware resources and primitive hardware interfaces is concentrated at this level.

* The kernel APSE, the middle layer, presents a machine independent portability interface and provides database, communication and runtime support functions for executing Ada programs (including members of the APSE tool set).

* The outermost layer includes the APSE tool set and user programs. Part of this layer is called the minimal APSE, the minimum set of tools, written in Ada and supported by the KAPSE, necessary and sufficient for the development and continuing support of Ada programs.

The four principal features of the Ada Integrated Environment are its run-time software environment, its database, its language processors, and its software toolset. These features correspond to the four system segments named in the System Definition. The software environment and database segments constitute the Kernel APSE. The language processors and software toolset are components of the Minimal APSE.

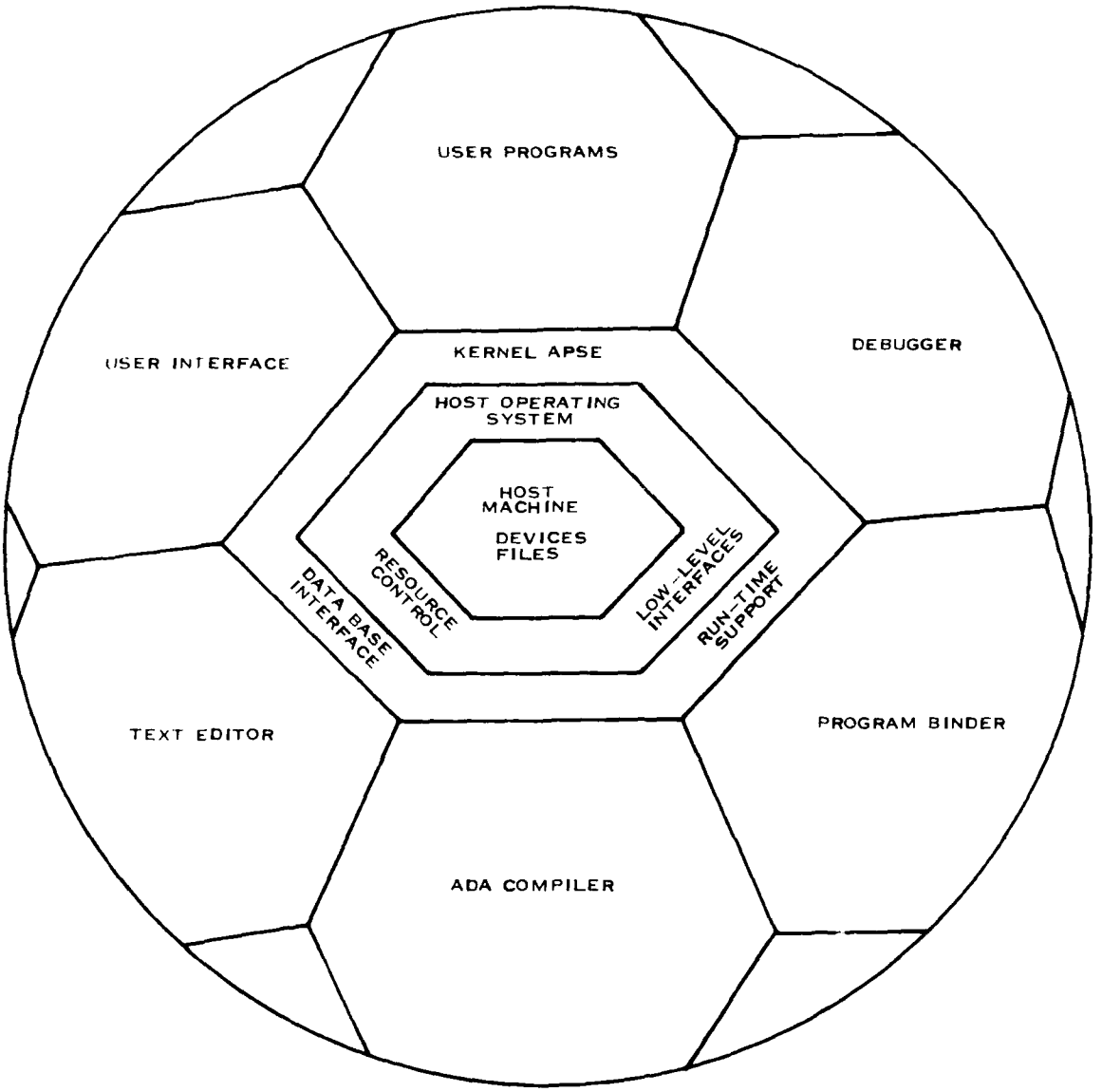


Figure 3-1 Ada Programming Support Environment - Concept

3.1.1.1 Ada Software Environment

The Ada Software Environment provides a command language with which the user invokes programs, manipulates database objects, and requests other system services. Other components of the Software Environment segment provide host-independent and device-independent communications interfaces between Ada programs, whether they be user-written or APSE tools, between these programs and their users, and between the programs and the underlying host operating system and equipment. Finally, the Software Environment provides capabilities to load a compiled Ada program into the address space of the host machine, allocate any run-time data structures required by the program, place the program into execution, monitor its progress, and assist as necessary upon program termination.

3.1.1.2 Ada Database Subsystem

The database serves as the repository for all APSE tools and for all user- and system-generated programs and data. It is also the primary medium of communication between APSE components. Each separately identifiable collection of information in the database is called an object. Each object has a unique name; it contains information; and it has attributes that describe its nature and its relationships with other objects. APSE tools and users must have controlled access to database objects and and their attributes and controlled capabilities to create, delete, modify, store, archive, and retrieve them. Components of the Database Subsystem define and control the interfaces between the database and its users, provide utilities for the handling of database objects, and provide facilities for the management of users, access controls, and security.

3.1.1.3 Ada Language Processors

Ada language processors transform the source text of program units into the machine code of target computers. Through the separate compilation facilities of the Ada language, Ada compilers also participate in the integration of source text into program units and subsequent integration of compiled program units into programs. Each Ada compiler performs the three major functions of source language checking and translation, optimization, and target machine code generation. Other components of the language processor segment create and maintain program library files, perform analyses on source text, or produce listings and reports from the compilation process.

3.1.1.4 Ada Programming Toolset

The software toolset contains all other system-provided or user-written programs used in software design, development, maintenance and management activities. The software tools included in the minimal Ada Programming Support Environment and described in this specification are:

* An interactive general purpose text editor, providing facilities for the creation and modification of text files containing Ada source text or other programs, data or documentary material. The editor includes interfaces to database and configuration management

tools.

* A program binder, to integrate Ada program units into complete programs and map the resulting programs from the Ada virtual machine to the architecture and configuration of a target machine.

* An interactive, source-language-level debugger, providing diagnostic facilities for analysis of runtime errors in programs executing on the Ada Integrated Environment host computer or on an appropriately connected target machine.

3.1.2 Mission

The efficiency and reliability of nearly all major computer system developments in Government and industry have historically been severely impacted by the lack of integrated, high quality software tools and computing environments to support developers and managers through all phases of the software product life cycle.

While many excellent tools have been developed and are in use in various projects, few of them are integrated into complete development systems and even fewer are ever successfully transported from the host system on which they were developed. In some cases, sophisticated support tools have not been used because their host computers lacked the capacity to run them efficiently on real projects. Much emphasis has been placed on high order languages and structured programming techniques, but computing environments that provide only machine-level debugging tools have largely neutralized the benefits of these and other modern programming practices. Finally, software developers are nontransportable because each host system presents a different user interface.

Potential benefits of the widespread use of an Ada Integrated Environment include the following:

* Improved productivity of software producers, through standardization of the interfaces between users and host systems, powerful interactive tools, and automation of most routine software development and documentation activities.

* Improved software product quality and reliability, through support of modern design, development, testing, and analysis methodologies.

* Improved support of software products in use, through improved program readability and documentation, automated trouble reporting, change control, and configuration management.

* Improved management of software projects, through automated project planning and scheduling, tracking and analysis of activities.

3.1.3 System Diagrams

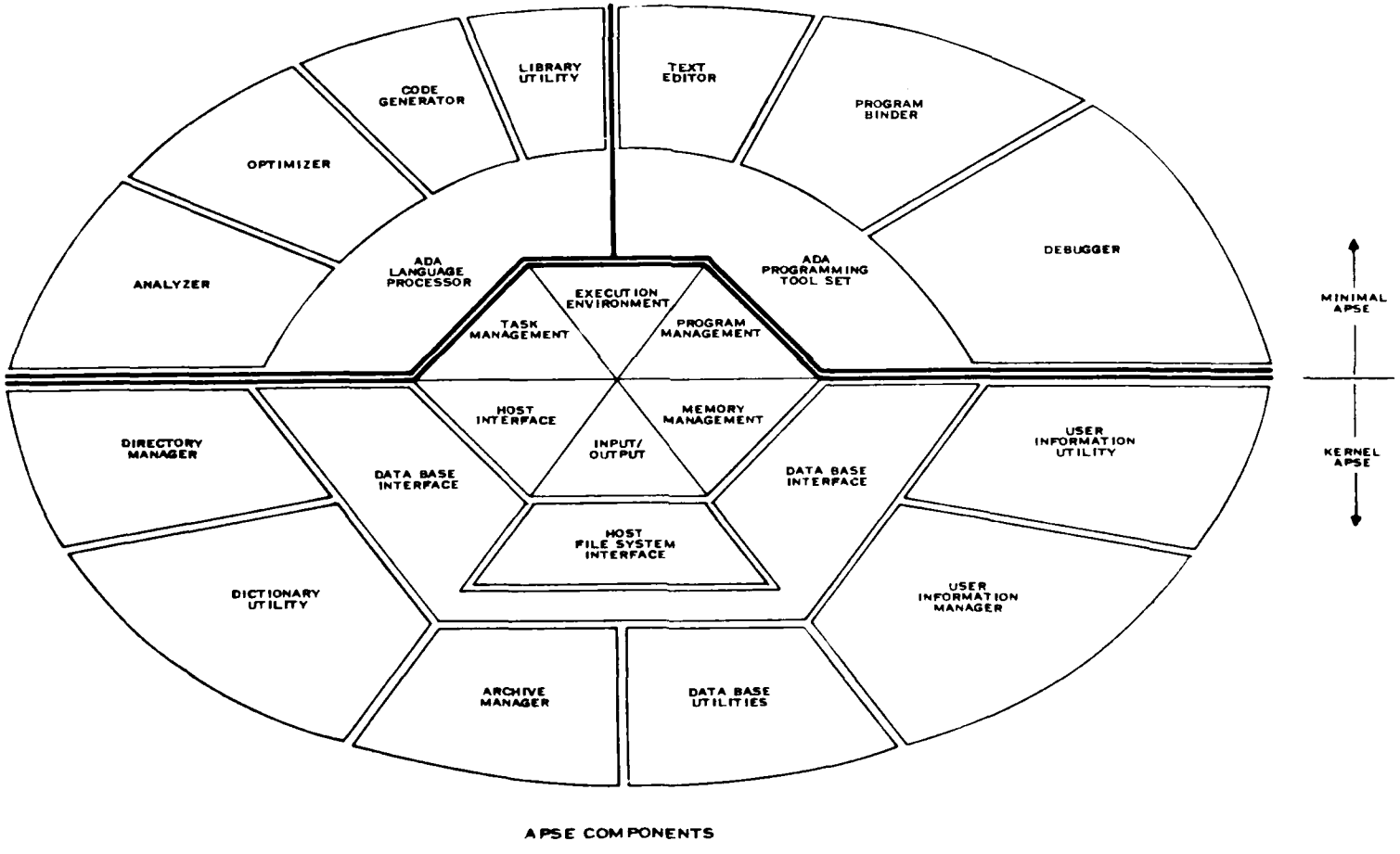


Figure 3-2 Ada Integrated Environment - General Structure

3.1.4 Interface Definition

The Ada Integrated Environment must interface with its interactive and non-interactive users, with its host computer and operating system, and with sister APSE systems. Provisions must also be made for future interfaces with computing networks, such as ARPANET. Within the environment, APSE tools and all other Ada programs must interface with their users, with a database, and with each other. Some APSE tools must interface with target computers for testing of software products.

3.1.4.1 User Interfaces

Users access the Ada Integrated Environment through batch job entry or interactive terminals. A single machine independent Command Language is the user interface in all cases. The "job" of a batch user and the "session" of an interactive user are both sequences of statements in the Command Language. These sequences may invoke stored Ada programs or stored Command Language procedures, which in turn may invoke other programs and procedures.

In addition to a uniform mechanism for program invocation and transmission of parameters, the Command Language shall provide controlled facilities for:

* Allocation of system resources to programs
* Attachment of database objects or I/O devices to programs
* Conditional and iterative execution of programs
* Transmission of data values between programs
* Transmission of special messages to the user.

Facilities shall be provided for the creation, storage and use of libraries of user-written Command Language procedures and for user definition of constants and other global parameters that constitute a personalized programming environment.

3.1.4.2 Host System Interfaces

The Ada Integrated Environment shall interface with its host computer and operating system, if one is present, to manage resources and provide basic services to all Ada programs and users. Typical host system services include:

* Interrupt handling
* Management of physical memory
* Management of system mass storage
* Processor scheduling
* Low-level data transfer to devices

* Recording of resource status and use

* Management of system spooling devices

* Telecommunications interfaces

Host system dependencies shall be isolated in these interfaces, to insure APSE portability.

3.1.4.3 Sister System Interfaces

Interfaces may be established between the Ada Integrated Environment and other APSE systems. It is anticipated that off-line media, such as magnetic tape, will be a principal method of transmitting copies of database objects between APSE systems. The Ada Integrated Environment shall provide utilities to integrate copies of database objects, written by other systems, into its own database and to prepare copies of database objects, including the values of all appropriate attributes, for transmission to other systems.

3.1.4.4 Ada Program Interfaces

A running Ada program communicates with its environment by assignment of values to parameters at program invocation, or by the use of standard subprograms and packages that interface with the underlying operating system to provide requested services, such as:

* Logical input/output

* Dynamic storage allocation

* Inquiries concerning system status

* Values of program, task or file attributes.

A logical input/output package shall be provided to implement a uniform, device-independent approach to data transfer between a program and its environment. Specifically, logical input/output capabilities shall be provided for database objects, input/output devices, interactive terminals, and inter-program communications.

A standard package shall provide Ada programs with controlled facilities to interrogate or modify the values of the user-maintained attributes of database objects and relations between them. This package shall be integrated with the logical input/output package to provide a uniform treatment of the attributes of all logical files.

The interface between a user and a running Ada program is a control file, containing detailed instructions that specify the processing to be performed by the program. All tools provided in the Ada Integrated Environment shall incorporate standard provisions for control input from an interactive terminal or from a file stored in the database.

3.1.5 Operational and Organizational Concepts

The operational and organizational concepts of an Ada Programming Support Environment are stated in the STONEMAN specification. The Ada Integrated Environment must provide a useful Ada programming environment and support its own extension with new tools written in Ada. Special emphasis must be given to the host-target software development concept and a well defined model of the software development process

3.1.5.1 Host-Target Software Development Concept

The computer supporting a software development effort is called a host machine. The computer on which a particular software product runs is called a target machine for that product. Generally, a host computer is expected to offer extensive software support, computing power and storage facilities. A host computer is expected to provide a general-purpose operating system with file management, resource management, scheduling and other run-time support for all user programs. A host computer is also the target machine for those programs it executes.

In general target computers may have many properties that differ from those of typical host machines. Many target machines are designed to run only a single application program, and they may be designed to provide only enough computing power and storage capacity to support that application. For the same reason, instruction sets and processor/memory configurations are frequently specialized and limited. Finally, the operating systems of target computers usually provide only those run-time support functions essential to a limited number of specific application programs.

The Ada Integrated Environment shall be designed to operate efficiently on host computers that provide reasonable processing power and storage capacity.

The Ada Integrated Environment shall provide uniform treatment to any software products designed to execute on the host machine and to the software products that must be transferred to target machines of different architecture for execution.

The Ada Integrated Environment shall provide standardized interfaces for communication between software testing and debugging tools running on the host machine and software products under test in target machines.

3.1.5.2 Model of the Software Development Process

A definition of the functional characteristics of each of the tools of a minimal APSE may be derived from a view of the software development process as a sequence of integrating and transforming steps that begins with the simplest lexical units of a programming language and progresses to produce a complete program. Figure 3-3 shows the general process.

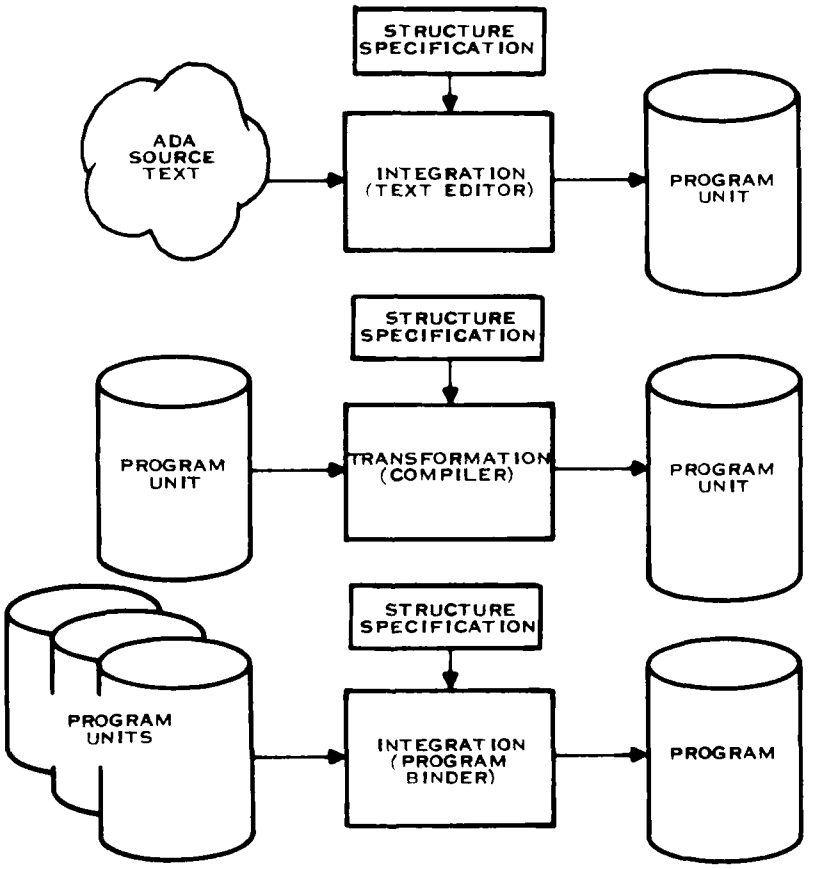


Figure 3-3 Model of the Software Development Process

At each level of integration, constituent units are brought together by a software tool to form a larger entity. These integrating tools are primary components of the MAPSE toolset.

* A text editor is an example of a tool used to form program units from smaller constituent parts. The editing process may include direct text entry, deletion or changes by an interactive user or may merge text from several source files to create the desired program unit.

* A program binder (or linkage editor) is an example of a tool used to form a complete program from specified constituent program units. The binding process may merge program units from several libraries to create the desired program.

Various transformations may be performed on the constituent units of a software product at any level of integration. Transformation of a program or program unit changes its representation without changing its meaning.

* A translator is an example of a tool used to transform the declarations and statements of a program unit from the human-written source text of a programming language to an intermediate representation more suitable for analysis, optimization and other processing.

* An optimizer is an example of a tool used to analyze and transform a program unit to improve its performance or its use of computing resources.

* A code generator is an example of a tool used to transform a program unit from an intermediate representation to a form compatible with the instruction set architecture of a target machine. This tool usually performs some additional optimizations.

* A composite transformation tool consisting of a translator, optimizers, and a code generator is usually called a compiler.

Static and dynamic analysis tools may be applied to the constituent units of a software product at any level of integration.

* A cross reference analyzer is an example of a static analysis tool that locates the definition of each symbol in a program unit and identifies the program statements that refer to the symbol.

* A source language level debugger is an example of a dynamic analysis tool that maps the memory image of an executing Ada program to the source program text and data definitions, allowing a user to examine or modify data values and control program execution.

Configuration management tools attempt to record and control the changes made to constituent units of a software product so that (re)construction of a product is consistently done from known, compatible parts. These tools use a database that provides data structures and facilities for the storage and retrieval of information, generated by MAPSE tools, concerning the constituent units of a software product.

The Ada language provides several features that support separate compilation of program units. As a result, construction of a correct context for compilation of a program unit may require access to other program units in the database. The relationships between program units are defined by the static lexical (nested) structure of the program.

The several steps of the compilation process produce intermediate results which must be stored. Listings and data for debuggers or other analysis tools may also be generated, and these must be related to the original source text.

A special file is necessary to specify the structure of a program, name its constituent program units, and provide the mapping from internal program unit names to the external names of database objects. This file is called a library file. It also contains information on the compilation status of each program unit and the names of derived files. The library file is thus a key element in Ada software configuration management.

An integrated software development process must begin with specification of a program structure, which then controls all subsequent construction, transformation and integration of constituent units to form the final program. Program structure information is stored in the library file.

Each of the software tools described in this model obtains its processing instructions from a control file, which may be an interactive user's keyboard in appropriate cases. One or more program units may be processed as input. The processing context for each program unit is determined by reference to the library file. Upon completion of processing, the library file may be updated to show appropriate results. The output -- an integrated or transformed program unit -- is stored in the database with appropriate attribute values and relations set to indicate its position in the set of derived program products.

3.2 Characteristics

The following paragraphs present requirements for the performance, reliability, maintainability, availability, and transportability of the Ada Integrated Environment system.

3.2.1 Performance Characteristics

The performance of the Ada Integrated Environment shall be measured in terms of its use of host system resources and in the efficiency of the software products it generates. The productivity of the users of the Ada Integrated Environment is an important indirect performance measure.

3.2.1.1 Use of Host System Resources

The Government shall specify the machine and operating system configurations for the initial Ada Integrated Environment host systems. Acceptance test plans shall specify performance requirements, in terms of processing speed and memory use in these host systems, for each CPCI and for the system as a whole. Interactive response time criteria shall be specified for appropriate system components. Criteria for efficiency of database mass storage use shall be specified for appropriate components.

3.2.1.2 Efficiency of Generated Software Products

Acceptance test plans shall specify performance requirements, in terms of processing speed and memory utilization on the host systems, of selected test programs generated from input Ada source text by the Ada Integrated Environment compiler and software toolset.

3.2.2 Reliability

Reliability of the Ada Integrated Environment shall be measured by its ability to protect itself from user and system errors, to recover from unexpected operating conditions, and to provide meaningful diagnostics.

3.2.2.1 Protection from Errors

System integrity shall be maintained by implementing standard packages for access to all system resources, allocation mechanisms, and data structures.

* Access control shall be available for each database object and for each defined attribute of a database object.

* The Database Subsystem shall control all access to shared database objects and its own data structures.

* The memory space allocated to each user program for execution

shall be logically isolated from the space occupied by all other programs.

* The memory protection features provided by host machines and operating systems shall be used to the greatest extent possible.

* System exception handling routines shall deal with exceptions by isolating the fault to the smallest possible set of executing programs.

3.2.2.2 Recovery

Recovery from an unexpected operating condition shall attempt to restore the state of the system, user programs and all data structures to conditions in effect at the time of interruption.

* System executive programs interfacing with users shall preserve the user environment and other data necessary for an attempt to restart and re-attach interactive users.

* The Database Subsystem shall incorporate internal checks for database integrity and shall be designed with data structures that tend to localize, rather than propagate, damage.

* APSE tools shall generally be designed to provide restart and recovery capabilities in the data structures they create or use.

3.2.2.3 Diagnostics

The system shall include software instrumentation at all levels to collect data for analysis of system performance and for recording of software development activities and resource usage. Most data collection will be accomplished routinely by the operating system. .

* The system shall include centralized facilities for recording of program initiations and terminations, error conditions, resource usage, and other pertinent operating data.

* Standard interfaces shall be provided to APSE tools and user programs for access to recording facilities and to recorded data.

3.2.3 Maintainability

APSE software maintainability shall be achieved by strict adherence to Ada-based standards for software design, development, testing and management.

* Ada programming standards and conventions emphasizing program structure, modularity, data structures and interfaces shall be applied to all APSE software.

* APSE software shall be designed to be modular and reusable, and shall be documented accordingly.

3.2.4 Availability

Availability of a software development facility is measured by its ability to provide satisfactory, timely service to its users. Interactive response time, batch job turnaround time, time to recover from system errors or interruptions, and downtime for scheduled or unscheduled maintenance are all factors affecting APSE availability.

* The system shall provide self-monitoring capabilities to analyze its performance under varying resource loading conditions, with optional facilities to limit system loading, if necessary, to maintain satisfactory levels of performance.

* System software shall be designed so that replacement of an APSE component or repairs to APSE data structures has minimal impact on parts of the system not directly involved in such repairs.

3.2.5 Transportability

The Ada Integrated Environment shall be designed to be transported to a variety of host systems, and shall support the development of Ada programs for a variety of target computer instruction set architectures.

* The Ada Integrated Environment shall include a clearly defined kernel that isolates host machine and operating system dependencies and provides host-independent database, communications and runtime support facilities to all Ada programs, including APSE tools.

* The user interface shall be implemented in a command language that is independent of the characteristics of the host system.

* The Ada compiler is the primary APSE retargeting and rehosting tool. The compiler shall be designed and implemented to facilitate its own retargeting, and must be capable of functioning in a non-APSE environment with minimal runtime support.

3.3 Design and Construction

The following paragraphs specify general system design and construction standards for the Ada Integrated Environment. They cover software design and development procedures, program structures, and programming conventions.

3.3.1 Software design and development procedures

Design and development procedures shall be specified in the Computer Program Development Plan (CPDP) associated with this specification.

3.3.2 Computer program structures

Each Ada Integrated Environment CPCI is a collection of functionally related computer programs and data structures. Each computer program component, in turn, will be a hierarchical set of program units (subprograms and packages) organized to conform to Ada program structuring rules.

* The structure of each computer program is defined by its associated program library file. Each program shall be partitioned into library units and subunits for separate compilation.

* A standard protocol shall be followed for the inclusion of machine dependent program units in Ada programs.

3.3.3 Programming conventions

Ada programming standards and coding conventions shall be specified in the CPDP and applied to all APSE software.

* All MAPSE tools shall be written in the Ada language.

* Certain machine-dependent KAPSE program units may be written in languages other than Ada to satisfy low level interface requirements. Such program units shall conform to programming standards specified for the selected languages and incorporated in Ada Integrated Environment documentation.

* Uniform naming conventions, attribute definitions and protocols shall be specified for the primary data input, control input, data output, message output and listing output files used by all APSE tools.

3.4 Documentation

This paragraph gives general requirements for documentation of the Ada Integrated Environment system. It specifies formal and informal documents for design, development, testing, maintenance, training and management.

3.4.1 Contractually Required Documents

The Ada Integrated Environment contract requires the preparation and delivery of the technical documentation shown in Table 3-2, which gives the schedule for draft (D), final (F) and update (U) versions of each document for each contract phase.

Table 3-2 Ada Integrated Environment Technical Documentation

		Phase 1	Phase 2	Phase 3
System Specification		D	F	U
Development Specification	(CPCI)	D	F	U
Test Plan	(CPCI)	D	F	
Product Specification	(CPCI)		F	U
Test Procedures	(CPCI)		F	
User Manual			F	U
Maintenance Manual			F	U
Rehostability Manual			F	U
Retargetability Manual			F	U

Certain management documents are also required by the contract. These include an interim Technical Report, submitted at the end of Phase 1, and a Computer Program Development Plan (CPDP) for the system. Detailed requirements for documentation of the system are provided in the CPDP.

3.4.2 Computer Program Development Specifications and Test Plans

A Computer Program Development (B5) Specification and a formal Computer Program Test Plan shall be prepared for each CPCI of the Ada Integrated Environment. The draft B5 specifications and test plans submitted at the end of the Design phase shall be updated and resubmitted as documentation supporting the system Preliminary Design Review (PDR) at the beginning of the Development phase of the contract. Once approved by the Government, these documents shall be placed under configuration control.

3.4.3 Computer Program Product Specifications and Test Procedures

A Computer Program Product (CS) Specification and detailed Computer Program Test Procedures shall be prepared for each CPCI. These documents shall be derived from the associated higher-level specifications and shall provide details at the Computer Program Component level. These documents shall be submitted to support the Critical Design Review of each CPCI. Once approved by the Government, these documents shall be placed under configuration control.

3.4.4 User Manual

A User Manual shall be prepared for the Ada Integrated Environment system. It shall contain operational and reference information essential for use of the system by an application programmer, including a complete description of the parameters, inputs, command languages, functional capabilities, and outputs of all delivered APSE tools.

3.4.5 Maintenance Manual

A Maintenance Manual shall be prepared for the Ada Integrated Environment system. It shall contain system construction, operational, and reference material essential for maintenance of the system by qualified system programmers. The manual shall include concise descriptions of the program structures, interfaces, and specifications of all constituent packages and subprograms for each computer program component of the APSE.

3.4.6 Rehostability Manual

A Rehostability Manual shall be prepared for the Ada Integrated Environment to provide instructions for the transportation of the APSE software to a new host computer system. Detailed rehosting procedures will be specified for each APSE component.

3.4.7 Retargetability Manual

A Retargetability Manual shall be prepared for the Ada Integrated Environment to provide instructions for the implementation of APSE software to support a new target computer instruction set architecture. Detailed retargeting procedures will be specified for each affected APSE component.

3.5 Training

This paragraph gives requirements for training programs for Ada Integrated Environment users and maintenance personnel. All courses shall provide hands-on interactive practice where feasible.

3.5.1 User Training

A one-week training program shall be provided for users of the Ada Integrated Environment. This course shall assume proficiency in the Ada language as a prerequisite, and shall provide detailed instruction in:

* Use of the Command Language
* Use of the Database Subsystem
* Use of the Text Editor
* Compiling and binding Ada programs
* Source language level debugging.

3.5.2 System Programmer Training

A two-week training program shall be provided for system programmers. This course shall assume full proficiency in the Ada language and completion of the user training course as prerequisites, and shall provide detailed instruction in:

* Kernel APSE organization, operation, and interfaces
* Database organization, operation and interfaces
* Compiler organization, operation and interfaces
* General organization of other APSE tools and data structures
* Configuration management of APSE software
* System initialization procedures
* Rehosting and retargeting procedures

3.6 Functional Area Characteristics

3.6.1 Software Environment

The Ada Software Environment provides a user command language, host- and device-independent communications interfaces, and facilities to load an Ada program, allocate resources, monitor program progress, and assist upon program termination. The Ada Software Environment is a kernel APSE (KAPSE) according to the STONEMAN definition. Its components are the execution environment, the APSE Manager, program management, task management, storage management, and low level input/output.

3.6.1.1 Execution Environment

The Ada execution environment is the component that defines how features of the Ada abstract machine will be implemented on a particular processor. It is essentially a specification for the code generation phase of the Ada optimizing compiler for the code sequences to be generated for particular language features. For features whose semantics are too complex to be translated inline, out-of-line routines are supplied to be called by the code generator.

3.6.1.2 Storage Management

The storage management package contains the routines with which objects whose lifetimes are dynamic will be allocated and deallocated. Such objects are either ACCESS objects that can be manipulated directly by the user via the allocator NEW (and possible the deallocator UNSAFE_DEALLOCATION) or are created indirectly by KAPSE in response to user statements (e.g., task elaboration requires allocation of data structures by the task management package.)

3.6.1.3 Task Management

This component of KAPSE implements the tasking features of Ada. Included are task elaboration and termination, rendezvous, and interrupt handling.

3.6.1.4 Program Management

The Ada Integrated Environment requires that KAPSE support the execution of one or more programs by a user; this component provides the interface between the user and the invoked programs. Included are the executive control language with which the user can control the configuration of his terminal, the command language with which programs can be invoked interactively, and the program invocation package which supports the invocation of one program by another.

3.6.1.5 APSE Manager

The APSE manager is a collection of programs associated with the Ada Integrated Environment instead of a particular user. When the AIE is brought up on a machine, the APSE manager is created as an anonymous user having the same structure (i.e., an executive program that is controlling other programs) as other users but is not associated with a particular terminal. Within the APSE manager, there are programs such as the database manager and batch monitor which provide a system-wide service.

3.6.1.6 Input / Output

The input / output component of KAPSE provides both the low-level I/O routines with which data transfers can be made and the high-level routines, oriented to the needs of an Ada programmer. Conversion between internal and external representations can be specified using templates.

3.6.2 Database Subsystem

The Ada Database Subsystem contains the following major components: Database Interface, Database Directory Manager, Database Dictionary Utility, Database Utilities, Host Filesystem Interfaces, User Information Manager, User Information Utility, and Database Archive Manager.

3.6.2.1 Database Interface

The Database Interface package consists of the routines for general operations on file attributes and relations. This is the interface between all programs and the Ada Database Subsystem. The package contains type declarations, functions to handle database names, functions to obtain the values of file attributes and relations, and procedures to assign file attributes and relations.

3.6.2.2 Database Directory Manager

The Database Directory Manager is a program consisting of routines to read from and write to the Database Directory, and to read the Database Dictionary and Database Map. It resolves File Parameter Blocks created by the compiler and the Command Language Interpreter into the appropriate File Attribute Blocks.

3.6.2.3 Database Dictionary Utility

The Database Dictionary Utility is a program consisting of routines to define and check attributes and relations, making appropriate entries in the Database Dictionary.

3.6.2.4 Database Archive Manager

The Database Archive Manager is a collection of management routines and utilities to manage on-line storgae resources efficiently, while preserving the integrity, consistency, and availability of all database objects.

3.6.2.5 Database Utilities

The Database Utilities consists of miscellaneous routines for the manipulation of database objects and directories.

3.6.2.6 User Information Manager

The User Information Manager is a program consisting of routines to extract information in the form of User Attributes and relations from the User Information Directory and User Information Dictionary. This information is placed into a data structure, the User Attribute Block, and communicated to the Command Language Interpreter or to the Database Directory Manager in response to a request. These routines are "read only" forms of the routines in the Database Directory Manager and Database Dictionary Utility.

3.6.2.7 User Information Utility

The User Information Utility is a program consisting of routines to define User Attribute Definitions making entries in the User Information Directory and User Information Dictionary. These routines are "write only" forms of the routines in the Database Directory Manager and Database Dictionary Utility.

3.6.2.8 Host Filesystem Interfaces

The Host Filesystem Interfaces consists of routines to pass data and control between the Ada Database Subsystem, running Ada programs, and the host filesystem. The general routines permit the exchange of data between the Database Directory Manager and Host Filesystem Interface in the form of Database names, host file information and in the form of system tables. A special utility called the Database Map Utility communicates within the Ada Database Subsystem, internally passing information in the form of Database names and pathnames between the Database Directory Manager and Database Map.

3.6.3 Language Processor

The Ada optimizing compiler consists of three major passes: the analyzer, the expander/optimizer, and the code generator.

3.6.3.1 The Analyzer

The analyzer consists of four major tasks:

* The lexical analysis task converts Ada source text for a language construct into tokens and enters information about certain declared tokens (symbols) into a symbol table.

* The syntax analysis task checks that the tokens form a legal derivation based on the grammar using a recursive descent parsing algorithm and constructs an abstract syntax tree of the language structure.

* The semantic analysis task checks that the language structure is valid based on the static semantic rules of the language. Generic instantiations and inline subprograms are expanded.

* The intermediate language generator task writes out the intermediate language (abstract syntax tree and symbol table) for the compilation unit in an external representation for use by other tools (in particular, the expander/optimizer and the code generator).

3.6.3.2 The Expander/Optimizer

This pass consists of the interleaving of intermediate language expansion and local machine independent optimization, followed by global machine independent optimization. Its major tasks and subtasks are:

* The function of the expander is to expand the high-level machine independent intermediate representation of a compilation unit to expose the details of computations and specify run-time representations of data types and control structures. Responsibilities of the expander task include:

  - Collection of information about the compilation unit by traversing the abstract syntax tree.

  - Make run-time representational choices for data and control structures based on the collected information and the Ada virtual machine for the target machine.

  - Incorporate the representational choices explicitly into the high-level abstract syntax tree by expanding implicit computations into the abstract syntax tree.

* The purpose of the machine-independent optimizer is to perform optimizations on the expanded intermediate representation of the compilation unit by generating and analyzing flowgraphs of the unit. Responsibilities of the optimizer task are:

  - Perform machine-independent optimizations on the expanded abstract syntax tree, e.g., constant folding, constant propagation, and local common subexpression elimination.

  - Analyze the flowgraph.

  - Perform global machine-independent optimizations.

  - Save the optimized intermediate language.

3.6.3.3 The Code Generator

The function of the Ada Language Code Generator is to map the characteristics of the target computer onto the intermediate representation of a compilation unit to provide a basis for the generation of code for the

target machine. The code generator is responsible for the following major tasks:

* Map the machine characteristics onto the intermediate language by transforming the intermediate language into a lower-level form more suitable for processing.

* Allocate storage (displacement in stack frames/heap, registers) for variables and literals in accordance with the abstract machine model and the characteristics of the target machine.

* Generate code for target machine.

* Perform machine dependent optimizations.

* Assemble code into an object module.

* Produce information required by Ada source level degugger.

3.6.4 Programming Toolset

The software toolset contains all other system-provided or user-written programs used in software design, development, maintenance and management activities. The software tools included in the minimal Ada Programming Support Environment and described in this specification are:

* An interactive general purpose text editor

* A program binder

* An interactive, source-language-level debugger

3.6.4.1 Interactive Text Editor

The Interactive Text Editor consists of four packages of procedures and functions. The packages divide the labor of editing along the lines of functionality in order to make future modification of the editor a simple task. The four packages are:

* the Presentation Manager handles communications with the terminal to which the editor is interfacing. Since there are many types of terminals of different capabilities, this portion of the editor will handle all terminal dependent communications. The complexity of these communications ranges from the complicated, such as the management of character level I/O from a device such as a dumb video display terminal, to the simple, such as transmitting and receiving text one line at a time with, for instance, a teleprinter terminal.

* The Edit Command Interpreter translates editor commands entered by a user into a sequence of actions required by the editor. The commands, received from the Presentation Manager, are strings which

are translated into calls to procedures in the Edit Command Processor.

* The Edit Command Processor is a package of procedures performing the majority of the operations required by the editor. The Edit Command Processor performs many of the operations by itself and translates others into calls to routines in the Edit List Manager.

* The Edit List Manager is a package of routines managing the internal representation of a text file while it is being edited.

3.6.4.2 Program Binder

The function of the Program Binder is to construct executable programs by binding together sets of program segments into complete self-contained programs. The Program Binder accomplishes its function in two phases. In the first phase, individual program units are gathered into groups, called 'segments'. These segments are the smallest entities to be used for the building of complete programs. The second phase, the program binding, organizes bound segments into executable programs. The user is able to control the organization and building of both the segments and the programs.

3.6.4.2.1 Segment Binding

The segment binding phase is the first step in the transformation of object modules produced by the compiler into executable programs which belong to a system. The segment binding phase is responsible for the formation of program segments. Program segments, being the smallest entities which may be explicitly included into a program, are planned compositions of named program units. Segments are capable of being shared among multiple programs and a well organized segment may be useful in many executable programs. To provide this capability, the user is given adequate controls to include selected program units, to set segment partition sizes and to determine overlay structures. These controls are effected through a set of commands which direct the structure and composition of the bound segment.

3.6.4.2.2 Program Binding

The program binding phase generates program units which are suitable for target machine execution. This binding phase functions in such a way as to allow the option of load-and-go binding or the saving of the program image within the program library for later, multiple executions. The program binding phase also supports the formation of program overlays. The binding of an Ada program is based upon a user designated "main" program segment. The program binding phase also prepares tables and data blocks required by the execution environment for proper execution.

3.6.4.3 Interactive Debugger

The Ada Interactive Debugger provides the user with facilities to monitor a running Ada program and modify its target program flow, display and change the contents of its data structures, and display diagnostic information. The debugger is designed to provide these capabilities in interactive and

non-interactive modes.

The design of the debugger permits the user to debug programs without any modification to the code generated by the Ada compiler. The debugger may be invoked at any point during the execution of the target program. The target program may be executing in the host machine or, if appropriate communications interfaces are provided, in some other target machine. The debugger may also be used as an interactive dump analyzer, when applied to a database object containing the image of the memory address space occupied by an Ada program.

The Ada Interactive Debugger consists of two major components. The Debugger Executive Program resides on the host computer and interfaces with the user, processes information, and accesses database objects. The Debugger Interface Package resides in the address space of the target program and provides the interface between the executive and the target program.

SECTION 4

QUALITY ASSURANCE PROVISIONS

4.1 Introduction

Testing of the Ada Integrated Environment system shall be in accordance with the schedule, procedures and methods set forth in the following documents:

1. Contractor's Computer Program Development Plan (CPDP)
2. Computer Program test Plan for each CPCI
3. Computer Program Test Procedures for each CPCI.

Testing of each CPCI shall be performed at three levels:

1. Computer program component test and evaluation
2. Integration test, involving all components of the CPCI
3. Computer program acceptance testing, involving the APSE

4.1.1 Computer Program Component Test and Evaluation

This level of testing supports development. Each program component of each CPCI shall be tested as a stand-alone program before integration. This testing shall concentrate on areas where new algorithms have been developed or where there is relatively high risk. Examples of such areas are:

* Database management and utilities
* Ada program library files and utilities
* Ada compiler separate compilation features
* Source language level debugging

An overall system test plan and schedule shall identify the parts of the system that must be available for testing of each component.

Test results shall be recorded in informal documentation; formal test reports are not required.

4.1.2 Integration Testing

This level of testing supports integration and prepares for acceptance testing of each CPCI. Each CPCI shall receive separate integration testing, using available components of the complete system. The system shall be constructed from tested CPCI's for integration testing. Testing at this level shall follow the approved test plans and procedures. Formal test reports are not required.

4.1.3 Formal Acceptance Testing

This testing assures that the Ada Integrated Environment system and its constituent CPCIs conform to all requirements in the Type A and B5 specifications. A formal test plan and test procedures shall be generated and used to insure satisfaction of all requirements. Acceptance tests shall be defined to incrementally test major functional components of the compiler. Acceptance testing shall be witnessed by the Government. Test results shall be documented in accordance with the Computer Program Development Plan and Computer Program Test Plans, and delivered to the Government with final system documentation.

All Ada compilers delivered shall be validated by the Government using the Ada Compiler Validation Facility. Government acceptance of each compiler shall be contingent on the the results of this validation and certification by the Ada Configuration Control Board.

4.2 Test Requirements

Unit testing and integration testing shall be performed using the developed computer program components or CPCIs and needed drivers. While testing shall not use formal test plans, testing shall keep the final acceptance tests in mind. Unit tests and integration tests consist primarily of an exercise of each specified feature of each computer program component. For all APSE tools that use control languages, each language feature shall be separately tested.

The test plans for each CPC shall specify the completeness of testing to be achieved by describing all logical paths through the code of each developed program and identifying testing conditions that will traverse the appropriate paths.

4.2.1 Rehosting tests

Parallel sets of tests at all levels shall be developed for each CPC to be run on the IBM 370 and the Interdata 8/32 host systems. Components of the 370 version may be used to simulate or provide drivers for components not yet rehosted on the Interdata 8/32, during unit testing.

4.2.2 Performance Requirements

The performance of each system component shall be measured in terms of its use of host system resources and in the efficiency of the software products it generates.

The Government shall specify the machine and operating system configurations for the initial Ada Integrated Environment host systems. Acceptance test plans shall specify CPCI performance requirements in terms of processing speed and memory use in these host systems. Interactive response time criteria and criteria for efficiency of database mass storage use shall be specified for appropriate system components.

Acceptance test plans shall specify performance requirements, in terms of processing speed and memory utilization on the host systems, of selected test programs generated from input Ada source text by the Ada Integrated Environment compiler and software toolset.

4.3 Independent Validation and Verification

An independent validation and verification (IV&V) contractor, if one participates in the Ada Integrated Environment program, may perform independent testing of the Ada compiler using any of the tests descibed above or additional procedures.

APPENDIX A

GLOSSARY

Abstract Syntax Tree -- An abstract syntax tree (AST) is a data structure built by the analyzer phase of a compiler to define the syntactic relationships between the tokens of the program unit.

Accept Statement -- An accept statement defines the actions to be performed when an entry of a task is called.

Access Type -- An access type is a type whose objects are pointers to dynamically created objects. The object itself is created by an allocator.

Access Value -- An access value designates an object pointed to by an entity of an access type.

Ada Database Subsystem -- The Ada Database Subsystem (ADS) provides data structures and facilities for storage and retrieval of information. Its components provide interfaces between the database and its users, provide utilities for the handling of database objects, and provide facilities for the management of users, access controls and security.

Ada Language Processors -- The Ada Language Processors transform the source text of program units into the machine code of target computers. These tools consist of an Analyzer, an Expander/Optimizer, and a Code Generator.

Ada Software Environment -- The Ada Software Environment (ASE) provides an interface between the user of the Ada Integrated Environment and the host system on which it is installed.

Ada Programming Support Environment -- An Ada Programming Support Environment (APSE) is a collection of software tools which provides facilities for the design, development, maintenance and management of software for one or more target computers.

Ada Programming Toolset -- The Ada Programming Toolset provides miscel tools or the development of Ada software; the minimal toolset includes a Text Editor, a Program Binder, and an Interactive Debugger.

Address Space -- Address space is a set of memory locations available for storage of programs and/or data.

Aggregate -- An aggregate is a written form denoting the value of an object of a composite value. An array aggregate denotes a value of an array type; a record aggregate denotes a value of a record type. The components of an aggregate may be specified using either positional or named association.

Allocator -- An allocator creates a new object of an access type and returns an access value designating the created object.

Analyzer -- An analyzer is a language translator that accepts source text for a compilation unit, performs lexical analysis, checks the syntax and static semantics of the compilation unit, and produces an intermediate representation that is more convenient for processing by compiler components and other tools. See also Front end.

Array Aggregate -- See Aggregate.

Array Type -- An array type is a collection of similar components addressed by one or more indices.

Asynchronous -- An event is said to be asynchronous if its occurrence is independent of other events in a system; e.g., depressing the break key causes an asynchronous program interrupt.

Attribute -- An attribute is a predefined characteristic of a named object.

Back End -- The back end of the Ada Optimizing Compiler is a language translator which accepts the DIANA dialect produced by the front end for a compilation unit and produces an object module for the compilation unit. The back end consists of the Expander/Optimizer and Code Generator passes.

Binder -- See Program Binder.

Block -- A block defines the scope of identifiers and other entities within an Ada program. A block statement contains an optional declarative part, followed by a sequence of statements, with an optional exception handler. Its body must be delimited by the BEGIN and END reserved words.

Body -- A body is a program unit defining the execution of a subprogram, package, or task. A body stub is a replacement for a body that is compiled separately.

Bootstrap Compiler -- A bootstrap compiler is an intermediate Ada compiler used for the development of the Ada Optimizing Compiler (which will compile itself).

Break Key -- A break key is a terminal keyboard key that interrupts execution of the current program.

Breakpoint, -- A breakpoint is an event in a target program which causes execution to be suspended and passes control to the Debugger.

Call Handler -- The Call Handler is the Ada Execution Environment routine that implements subprogram calls; i.e., it transfers control from one subprogram to another.

Code Generator -- A code generator is a tool used to transform the declarations and statements of a program unit from an intermediate representation to a form compatible with the instruction set architecture of

a target machine.

Code Section -- The Code Section is the portion of a bound program segment which contains the executable object code.

Code Section Dictionary -- The Code Section Dictionary is the portion of a bound program segment which contains entries indicating the locations of internal and external subprograms.

Collection -- A collection is the set of allocated objects of an access type.

Command File -- A command file is a file which contains a sequence of command language statements.

Command Language -- A command language is a collection of instructions to the Ada Integrated Environment specifying the execution of Ada programs; as such, it provides the user interface to the Ada Software Environment.

Command Language Interpreter -- The Command Language Interpreter (CLI) is a task within the Executive Program that is instantiated to translate and interpret the command language with which the Ada Integrated Environment user specifies the execution of Ada programs.

Command Procedure -- A command procedure is a file containing a sequence of command language instructions that is written in a form identical to an Ada procedure. A command procedure has its own name space; it may have parameters.

Compilation Unit -- A compilation unit is a program unit presented for compilation as an independent text. It is preceded by a context specification, naming the other compilation units on which it depends. A compilation unit may be the specification or body of a subprogram or package.

Compiler -- A compiler is a composite transformation tool consisting of a translator, optimizers, and a code generator. See also Ada Language Processors.

Component -- A component denotes one of a group of related objects known as a composite object. An indexed component names a component in an array or an entry in an entry family. A selected component is the identifier of the component, prefixed by the name of the entity of which it is a component, for instance, a discriminant within a record.

Composite Type -- An object of a composite type is a group of related objects known as components. An array type is a composite type, all of whose components are of the same type and subtype; the individual components are selected by their indices. A record type is a composite type whose components may be of different types; the individual components are selected by their identifiers.

Computer Program Component -- A Computer Program Component (CPC) is a

functionally or logically distinct part of a Computer Program Configuration Item (CPCI) distinguished for purposes of convenience in designing and specifying a complex CPCI as an assembly of subordinate elements.

Computer Program Configuration Item -- A Computer Program Configuration Item (CPCI) is an aggregation of hardware/software which satisfies an end use function; a system segment.

Configuration -- A configuration is a collection of database objects that are related by some common property or requirement.

Configuration Management Tools -- Configuration Management Tools are used to record and control the changes made to constituent units of a software product so that the product is consistently constructed from known, compatible parts.

Constant Handler -- The Constant Handler is an Ada Execution Environment routine which determines the location of the constant section associated with a program unit in a code section.

Constant Section -- The Constant Section is the portion of a bound program segment which contains blocks of constants (read-only data) associated with program units.

Constant Section Dictionary -- The Constant Section Dictionary is the portion of a bound program segment whose entries indicate the location of internal constant blocks.

Constraint -- A constraint is a restriction on the set of possible values of a type. A range constraint specifies lower and upper bounds of the values of a scalar type. An index constraint specifies lower and upper bounds of an array index. A discriminant constraint specifies particular values of the discriminants of a record or private type.

Context Specification -- A context specification defines the other compilation units upon which a compilation unit depends.

Control File -- A control file provides an interface between a user and a running Ada program. It contains detailed instructions that specify the processing to be performed by the program.

Control Program -- The Control Program (CP) is the component of the VM/370 which acts as the virtual machine monitor. It simulates multiple virtual machines on a single physical machine.

Cross Reference Analyzer -- A cross reference analyzer is a tool that locates the definition of each symbol in a program unit and identifies the program statements that refer to the symbol.

Database Name -- The database name is the unique internal name of an object assigned by the Ada Database Subsystem when the object is created.

Debugger -- A source level debugger is a dynamic analysis tool that maps the

memory image of an executing Ada program to the source program text and data definitions, allowing a user to examine or modify data values and to control program execution.

Declarative Part -- A declarative part is a sequence of declarations and related information such as subprogram bodies and representation specifications that apply over a region of a program text.

Demand Segmentation -- Demand segmentation is a method of memory management in which segments are loaded into memory as they are referenced. See also Segmentation.

Derived Type -- A derived type is a type whose operations and values are taken from those of an existing type.

DIANA -- DIANA is a high level intermediate language produced from the source code by the front end phase of the Ada Optimizing Compiler. This intermediate language is later optimized by the Expander/Optimizer and translated to machine language by the Code Generator.

Discrete Type -- A discrete type has an ordered set of distinct values. The discrete types are the enumeration and integer types. Discrete types may be used for indexing and iteration, and for choices in case statements and record variants.

Discriminant -- A discriminant is a syntactically distinguished component of a record. The presence of some record components (other than discriminants) may depend on the value of a discriminant.

Discriminant Constraint -- See Constraint.

Editor -- See Text Editor.

Elaboration -- Elaboration is the process by which a declaration achieves its effect. For example, it can associate a name with a program entity or initialize a newly declared variable.

Embedded Computer -- An embedded computer is designed for a specific function and resides in the system that performs the function.

Entity -- An entity is anything that can be named or denoted in a program. Objects, types, values, and program units are all entities.

Entry -- An entry is used for communication between tasks. Externally, an entry is called just as a subprogram is called; its internal behavior is specified by one or more accept statements specifying the actions to be performed when the entry is called.

Enumeration Type -- An enumeration type is defined by explicitly listing the values which that element may assume. These values may be either identifiers or character literals.

Exception -- An exception is an event that causes suspension of normal

program execution. Bringing an exception to attention is called raising the exception.

Exception Handler -- An exception handler is a section of program text specifying a response to the exception.

Expander/Optimizer -- The expander/optimizer is the component of the Ada Optimizing Compiler which performs the expanding and optimizing functions within one pass. See also Back End.

Expression -- An expression is a part of a program that computes a value.

Executive Program -- The Executive Program is the component of the Ada Software Environment that provides the interface between a user and the program invoked from the user's terminal. The Executive includes the user's terminal interface and the Command Language Interpreter.

Front End -- The front end of the Ada Optimizing Compiler is a language translator which accepts the Ada source text for a compilation unit, performs lexical analysis, checks the syntax and static semantics of the compilation unit, and produces the intermediate representation (DIANA) of the compilation unit.

Garbage Collection -- Garbage Collection is a memory management technique that attempts to reclaim allocated memory space as soon as it is no longer designated by any variable.

Generic Clause -- See Generic program unit.

Generic Program Unit -- A generic program unit is a subprogram or package specified with a generic clause. A generic clause contains the declaration of generic parameters. A generic program unit may be thought of as a possibly parameterized model of program units. Instances (that is, filled-in copies) of the model can be obtained by generic instantiation. Such instantiated program units define subprograms and packages that can be used directly in a program.

Generic Expansion -- Generic expansion is the replacement of generic formal parameters in the Intermediate Language template for the generic declaration with the actual parameters.

Generic Instantiation -- Generic instantiation is the substitution of the actual parameters for the generic formal parameters in a copy of the generic dynamic specification.

Generic Optimization -- Generic optimization is accomplished by sharing code between different instantiations of a generic definition.

Global Package Handler -- The Global Package Handler is an Ada Execution Environment routine which determines the locations of visible parts of packages global to the program.

Global Package Table -- The Global Package Table is a table containing the

locations of visible parts of packages global to the program.

Heap -- A heap is an area of memory reserved for dynamic variables. In Ada, dynamic variables are of the access type and are created by an Allocator.

Host Computer -- A host computer is a computer which supports a software development effort. It is expected to provide a general-purpose operating system with file management, resource management, scheduling, and other run-time support for all user programs.

Image Binding -- Image binding is a method of program binding by which the bound program is stored on disk in exactly the form it will have when loaded in memory; for example, the program contains all the inter-segment reference tables that will be needed at execution time.

Information Hiding -- Information hiding is the restriction of the visibility of an object or process to protect it from external influence. This function is handled in Ada by the private type.

Indexed Component -- See Component.

Interface -- An interface is a common design that allows communication between programs, tasks or data structures. See also KAPSE Virtual Interface.

Intermediate Language -- An Intermediate Language is a translation of the Abstract Syntax Tree generated by the Analyzer. This translation is usually machine-independent and may be further translated to object code.

Interrupt -- An interrupt is a response to an asynchronous or exceptional event that automatically saves the current CPU status (to allow later resumption) and causes an automatic transfer to a specified routine (called an interrupt handler).

Kernel Ada Programming Support Environment -- The Kernel Ada Programming Support Environment (KAPSE) provides the database, communication, and run-time support functions that enable the execution of an Ada program; these functions are a "kernel" in the sense that they provide a machine and operating system independent interface whose implementation on a host system uffices to install the Ada Integrated Environment. This interface is called the KAPSE Virtual Interface.

KAPSE Interface Task -- The KAPSE Interface Task (KIT) provides for interaction among the various components of the Ada Software Environment.

KAPSE Virtual Interface -- See Kernel Ada Programming Support Environment.

Lexical Descendents -- Lexical descendents are the subprograms that are nested within a parent subprogram.

Lexical Unit -- A lexical unit is one of the basic syntactic elements making up a program. A lexical unit is an identifier, a number, a character literal, a string, a delimiter, or a comment.

Library -- See Program Library.

Library File -- A library file is a separate database for maintaining the compilation state of a program or family of programs.

Library Unit -- A library unit is a compilation unit that is not a subunit of another compilation unit.

Literal -- A literal denotes an explicit value of a given type, for example a number, an enumeration value, a character, or a string.

Load-And-Go -- Load-And-Go is a method of program binding such that the resultant bound program is in a form ready for immediate execution.

Machine-Dependent Optimization -- Machine-dependent optimization includes optimizations performed on a program that are dependent on the target machine.

Machine-Independent Optimization -- Machine-independent optimization includes optimizations performed on a program that are language and target machine independent. Basically, they represent source-to-source transformations.

Main Program -- The main program of an Ada system is a designated subprogram which acts as a driver to the remainder of the package.

Minimal Ada Programming Support Environment -- The Minimal Ada Programming Support Environment (MAPSE) includes the compiler, text editor, debugger, terminal interface routines, project/ configuration control functions, and program binder.

Model Number -- A model number is an exactly representable value of a real numeric type. Operations of a real type are defined is terms of operations on the model numbers of the type. The properties of the model numbers and of the operations are the minimal properties preserved by all implementations of the real type.

Name -- A name denotes a declared entity, a result returned by a function call, or a label, block name, or loop name.

Named Association -- Named association indicates the value of an object by pecifying its identifier. See also Positional association.

Object -- Within the database, an object is a separately identifiable collection of information. W.thin an Ada program, an object can denote any kind of data element, whether a scalar value, a composite value, or a value in an access type.

Optimizer -- An optimizer is a tool used to analyze and transform a program unit to improve its performance or its utilization of computing resources.

OS/32 -- The OS/32 is the operating system of the Perkin-Elmer (Interdata) 8/32 computer.

Overlay -- An overlay is a portion of a program that resides on disk until it is referenced, at which time it is loaded into memory. Program Binder -- commands are provided to partition a program into overlays and to specify which overlays will share logical memory.

Overloading -- Overloading is the property that literals, identifiers, and operators can have several alternative meanings within the same scope. For example, an overloaded enumeration literal is a literal appearing in two or more enumeration types; an overloaded subprogram is a subprogram whose designator can denote one of several subprograms depending upon its parameter types and returned value.

Package -- A package is a program unit specifying a collection of related entities such as constants, variables, types and subprograms. The visible part of a package contains the entities that may be used from outside the package. The private part of a package contains structural details that are irrelevant to the use of the package but that complete the specification of the visible entities. The body of a package contains implementations of subprograms or tasks (possibly other packages) specified in the visible part.

Parameter -- A parameter is one of the named entities associated with a subprogram, entry, or generic program unit. a formal parameter is an identifier used to denote the named entity in the unit body. An actual parameter is the particular entity associated with the corresponding formal parameter in a subprogram call, entry call, or generic instantiation. The parameter mode specifies whether the parameter is to be passed into and/or returned by the program unit. A positional parameter is an actual parameter passed in positional order. A named parameter is an actual parameter passed by naming the corresponding formal parameter.

Parser -- A parser is a phase of a compiler that considers the context of each token returned by the syntax analyzer and classifies groups of tokens into larger entities such as declarations, statements and control structures; also referred to as lexical analyzer.

Partial Binding -- Partial binding is a technique of segment binding which allows the building of a program segment in multiple phases.

Pathname -- A pathname is a sequence of Ada identifiers that specifies the unique path through the directory hierarchy from the base or root to the specified object.

Positional Association -- Positional association specifies the value of an object based on its positional order. See also Named association.

Pragma -- A pragma is an instruction to the compiler, and may be language defined or implementation defined.

Primitives -- Primitives are functions which are accomplished directly by the Command Language Interpreter. They are indicated by the prefix "SYS." in the procedure name.

Private Type -- A private type is a type whose structure and set of values are clearly defined, but not known to the user of the type. A private type is known only by its discriminants and by the set of operations defined for it. A private type and its applicable operations are defined in the visible part of a package. Assignment and comparison for equality or inequality are also defined for private types, unless the private type is marked as limited.

Program Binder -- The program binder is a tool used to form a complete program from specified constituent program units. The binding process may merge program units from several libraries to create the desired program.

Program Library -- A program library is a collection of the compilation units of a program.

Program Parameter Area -- The Program Parameter Area (PPA) is an associative storage area used for passing parameters between program units.

Program Parameter Descriptor -- The program parameter descriptor is a block containing a list of parameter names and types required by the program. The program manager uses this information to obtain the parameter values from the user and pass them to a program.

Program Segment -- The subprograms that comprise an Ada program may be partitioned (by the program binder) into collections called segments. Intra-segment references are resolved, and inter-segment references are made through tables that facilitate sharing of code. A program segment consists of a Code Section, a Code Section Dictionary, a Constant Section and a Constant Section Dictionary.

Program Unit -- A program unit is the basic units of which programs may be composed. Units may be subprograms, packages, or tasks.

Qualified Expression -- A qualified expression is an expression qualified by the name of a type or subtype. For example, it can be used to state the type or subtype of an expression for an overloaded literal.

Raising An Exception -- See Exception.

Range -- A range is a contiguous set of values of a scalar type. A range is specified by giving the lower and upper bounds for the values.

Range Constraint -- See Constraint.

Record Aggregate -- See Aggregate.

Record Type -- A record type is a collection of similar or dissimilar components.

Rehost -- To rehost is to transport and adapt software from one host system to another.

Relation -- A relation is a labeled, directed arc that connects any two

database objects.

Relative -- A relative is a database object associated with another database object through a relation.

Retarget -- To retarget is to adapt software which was designed to execute on a given target computer to run on another target machine.

Rendezvous -- A rendezvous is the interaction that occurs between two parallel tasks when one task has called an entry of the other task, and a corresponding accept statement is being executed by the other task on behalf of the calling task.

Representation Specification -- A representation specification defines the mapping between a data type and its implementation on the underlying machine. In some cases, it completely specifies the mapping, in other cases, it provides criteria for choosing a mapping.

Scalar Type -- A scalar type indicates an ordered set of values by enumerating the identifiers which denote the values. Scalar types comprise discrete types (that is, enumeration and integer types) and real types.

Scope -- The scope of a declaration is the region of text over which the declaration has an effect.

Segmentation -- Segmentation is the technique for managing segments in memory. A segment is a logical grouping of information, such as a subprogram. A Segment Table indicates the address of each segment in memory.

Selected Component -- See Component.

Semaphore -- A semaphore is an abstraction operated on by synchronization primitives to coordinate concurrent access to a resource.

Slice -- A slice is a one-dimensional array denoting a sequence of consecutive components of a one-dimensional array.

Stack -- A stack is a sequence of memory locations in which data may be stored or retrieved on a last-in-first-out (LIFO) basis. Storage for a task is allocated in a structure called a stack region, which is subdivided into stack frames. These stack frames are allocated on a LIFO basis as control enters and exits subprograms.

Static Expression -- A static expression is one whose value does not depend on any dynamically computed values of variables.

Subprograms -- A subprogram is an executable program unit, possibly with parameters for communication between the subprogram and its point of call. A subprogram declaration specifies the name of the subprogram and its parameters; a subprogram body specifies its execution. A subprogram may be a procedure, which performs an action, or a function, which returns a result.

Subtype -- A subtype of a type is obtained from the type by constraining the set of possible values of the type. The operations over a subtype are the same as those of the type from which the subtype is obtained.

Subunit -- A subunit is a body of a subprogram, package or task declared in the outermost declaration part of another compilation unit) which may be compiled separately.

Symbol Table -- A symbol table is a table built by a compiler which contains the characteristics of the identifiers used in the program.

Target Computer -- A target computer is the machine on which the specified software is designed to execute.

Task -- A task is a program unit that may operate in parallel with other program units. A task specification establishes the name of the task and the names and parameters of its entries; a task body defines its execution. A task type is a specification that permits the subsequent declaration of any number of similar tasks.

Text Editor -- A text editor is a tool used to form program units from smaller constituent parts. The editing process may include direct text entry, deletion or changes by an interactive user, or may merge text from several source files to create the desired program unit.

Type -- A type defines the structure of a data element (enumeration, integer, real, array, record, or access), the values which the element may assume, and the operations which may be performed on the element. A type definition is a language construct introducing a type. A type declaration associates a name with a type introduced by a type definition.

Use Clause -- A use clause opens the visibility to declarations given in the visible part of a package.

Variant -- A variant part of a record specifies alternative record components, depending on a discriminant of the record. Each value of the discriminant establishes a particular alternative of the variant part.

Virtual Machine -- A computer architecture is said to support a virtual machine if it permits multiple instances of the architecture to be simulated on a single processor. Each user is given the full capabilities of the processor.

Virtual Terminal -- A virtual terminal is a logical terminal to which the input/output of an executing program may be directed; a virtual terminal may be connected to an actual terminal through a command to the Executive Program.

Visibility -- The declaration of an entity with a certain identifier is said to be visible at a given point in the text when an occurrence of the identifier at this point can refer to the entity, that is, when the entity is an acceptable meaning for this occurrence.

VM/370 -- The VM/370 (Virtual Machine /370) is an operating system of the IBM/370 computer that supports virtual machines.

Window -- A window is a portion of a physical terminal which may be connected to a virtual terminal. In peephole optimization, a window is the sequence of instructions being viewed.

With Clause -- A with clause is used to create an implicit declaration of the named library units.

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of technical competence is provided to ESD Program Offices (POs) and other ESD elements. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.


FILMED