N86-30368

# OBSERVATIONS FROM A PROTOTYPE IMPLEMENTATION OF THE COMMON APSE INTERFACE SET (CAIS)

Mike McClimens, Rebecca Bowerman, Chuck Howell,
Helen Gill, and Robbie Hutchison
MITRE Corporation

## EXECUTIVE SUMMARY

This paper presents an overview of the Common Ada Programming Support Environment (APSE) Interface Set (CAIS), its purpose, and its history. The paper describes an internal research and development effort at the Mitre Corporation to implement a prototype version of the current CAIS specification and to rehost existing Ada software development tools onto the CAIS prototype. Based on this effort, observations are made on the maturity and functionality of the CAIS. These observations support the Government's current policy of publicizing the CAIS specification as a baseline for public review in support of its evolution into standard which can be mandated for use as Ada is today.

## CAIS HISTORY

The Ada programming language was developed by the United States Government to promote the maintainability, portability, and reusability of software. Although no special software tools are required to use the Ada language, a collection of portable and modern tools is expected to enhance the benefits of using Ada. The term Ada Programming Support Environment (APSE) is used to refer to the support (e.g., software tools, interfaces) available for the development and maintenance of Ada application software throughout its life cycle. The Common APSE Interface Set (CAIS) is the interface between Ada tools and host system services, which is being standardized to promote portability of tools among APSEs.

In 1980, the DoD sponsored two efforts to develop APSEs: the Ada Language System (ALS) contracted to Softech by the Army and the Ada Integrated Environment (AIE) contracted to Intermetrics by the Air Force. The DoD also funded publication of the document, Requirements for Ada Programming Support Environments, nicknamed "Stoneman". It is the Stoneman document that first defined layers within an Ada Programming Support Environment. The Ada Joint Program Office (AJPO) was formed in late 1980 to serve as the principle DoD agent for the coordination of all DoD Ada efforts.

Multiple DoD-sponsored APSEs threatened to undermine the Ada program's goal of commonality. In late 1981/early 1982 AJPO established the

Kernel APSE Interface Team (KIT) as a tri-service organization chaired by the Navy. The KIT was supported by an associated group consisting of members from industry and academia, called the KIT Industry and Academia (KITIA). The charter of the KIT and KITIA was to define the capabilities that comprise the Kernal APSE layer (KAPSE) and its interface to dependent APSE tools. The interface between the KAPSE and dependent APSE tools became called the Common APSE Interface Set and a subgroup of the KIT/KITIA called the CAIS Working Group was formed to define a standard for this set of interfaces.

The CAIS has been an evolving concept. It began as a bridge between the Army and Air Force APSEs but has become a more generalized operating system interface. However, issues such as interoperability, configuration management, and distributed environments have not yet been addressed. Significant changes have appeared with each iteration of the CAIS specification up to the submittal in January 1985 of CAIS Version 1 as a proposed Military Standard (MIL-STD-CAIS).

In response to concern from the Ada community that the CAIS, as defined in Version 1, is too premature for standardization, a policy statement was released along with the proposed MIL-STD-CAIS directing that use of the CAIS be confined to prototyping efforts. The policy clearly states that the CAIS should not at this time be imposed on development or maintenance projects where the primary purpose is other than experimentation with the CAIS.

Further refinement of the CAIS is planned, but a contract to produce Version 2 of the CAIS specification has not yet been competed. Potential future applications of the CAIS include several major government projects (e.g., STARS and the NASA Space Station).

## CAIS OVERVIEW

The CAIS is a set of Ada package specifications that serve as calls to system services. The implementation of these packages may differ between systems while the package specifications remain the same. These package specifications then become a system independent interface between software development tools and the host operating systems. The CAIS is composed of four major sections: a generalized node model, support for process management, an extended input/output interface, and an abstraction for the processing of lists.

The generalized node model is by far the most significant part of the CAIS. Processes, structures, and files may all be represented as nodes. Among other features, the node model provides a replacement for the host file system. As such it contains enough functionality to support the needs of tools rehosted from a wide range of file systems. The node model is a hierarchical tree augmented by secondary relationships between nodes. Attributes may be assigned to any node or relationship in the tree. The attribute and relationship facilities provide a powerful mechanism for organizing and manipulating interrelated sets of nodes. The node model also provides support for mandatory (secret, etc.) and discretionary access control (read only, etc.).

Process support and an extended set of I/O interfaces are integrated with the node model. Process support is not extensive but does include the facilities to spawn and invoke processes or jobs and facilities for communication of parameters and results between processes. The I/O interfaces, on the other hand, are quite voluminous. Although they constitute more of the specification than the node model, the I/O interfaces largely duplicate the I/O support provided in Ada. In addition to integrating I/O with the node model, CAIS I/O tightens some of the system dependencies left in Ada and defines standard interfaces for devices such as scroll terminals, page terminals, and tapes.

The CAIS defines an abstract data type for processing lists. CAIS Lists may be any heterogeneous grouping of integers, strings, identifiers, sublist, or floating point items. Items may be named or unnamed. Lists are used throughout CAIS for the representation of data such as attributes and parameter lists, and they provide a powerful abstraction for tool writers in general.

## MITRE'S PROTOTYPE CAIS

Under a three staff year (Oct 84 to 85) internal research and development effort, MITRE Corporation has implemented a large subset of the CAIS specification and has exercised both rehosted and newly-written tools on this prototype. The MITRE prototype includes the node model, the list utilities, Text_Io, Direct_Io, and Sequential_Io. Parts of the process model and scroll_terminal have also been implemented in support of a line editor and a menu manager rehosted from other systems. In the next year the prototype will be completed, additional tools will be rehosted, the CAIS will be rehosted to a second system, and an analysis of distributing the CAIS will be undertaken. The prototype CAIS was developed using the Verdix Ada compiler running under Ultrix on a DEC VAX 11/750. Of the two tools rehosted to the prototype, one was originally developed using the Data General Ada compiler, and the other, using the Telesoft compiler.

The objective of MITRE's prototype development was to submit the CAIS specification to the rigor of implementation and actual use. It was believed that implementation of a prototype would test the implementability of the CAIS specification, would identify the level of support that CAIS provided to existing tools, and would result in practical input to CAIS designers, DoD policy makers, and program managers. The primary focus was on evaluating the CAIS functionality and not on developing an efficient implementation.

The consensus from this study is that the CAIS, for the most part, is internally consistent and provides a good foundation for continued work in standardized operating system interfaces for Ada programming support environments. The next version of the CAIS must, however, be considerably more complete in its specification. Table 1 lists the specific observations made as a result of the prototype

| Section | Item | Scale | Scope |
|---|---|---|---|
| 3.1.1 | The conceptual model is consistent, except for the I/O packages. | N/A | N/A |
| 3.1.2 | Some of the semantics are ambiguous. | Major | Semantics |
| 3.1.3 | Redundant capabilities and alternate interfaces need tightening. | Medium | Both |
| 3.1.4 | The nesting of packages within the package CAIS is not explicitly required. | Minor | N/A |
| 3.1.5 | The use of limited private types implies a need for additional facilities. | Minor | N/A |
| 3.1.6 | The error handling model in the specification is insufficient. | Major | Both |
| 3.1.7 | Parameter modes and positions are sometimes inconsistent. | Minor | Interface |
| 3.1.8 | The use of functions versus procedures should be consistent. | Minor | Interface |
| 3.2.1 | Multiple definitions of subtype names exist. | Minor | Interface |
| 3.2.2 | Inconsistent descriptions of access synchronization constraints are given. | Minor | N/A |
| 3.2.3 | Unnecessary complexity is introduced with the predefined relation 'User. | Minor | Semantics |
| 3.2.4 | The description of implied behavior of open nodes is good but needs to be more explicit. | Medium | Semantics |
| 3.2.5 | Boundary conditions are undefined. | Medium | Semantics |
| 3.2.6 | Capabilities for node iterators are limited. | Medium | Both |
| 3.2.7 | Definition of node iterator contents is ambiguous. | Medium | Semantics |
| 3.2.8 | Pathnames are inaccessible from node iterators. | Minor | Both |

| Section | Item | Scale | Scope |
|---|---|---|---|
| 3.3.2 | Ability to specify initial values for path attributes is missing. | Minor | Both |
| 3.3.3 | Error in sample implementation of additional interface for Structural_Nodes.Create_Node. | Minor | N/A |
| 3.4.1 | Treatment of files departs from the node model. | Major | Both |
| 3.4.2 | Consequences are implied by a common file type. | Medium | Both |
| 3.4.3 | Initialization semantics are incomplete. | Medium | Semantics |
| 3.4.4 | Mode and Intent are coupled. | Minor | Both |
| 3.4.5 | Additional semantics are needed for multiple access methods that interact. | Medium | Semantics |
| 3.4.7 | Import_Export of files is under-specified. | Medium | Both |
| 3.4.8 | Semantics of attribute values are conflicting. | Minor | Semantics |
| 3.4.9 | Interfaces diverge from Ada IO. | Minor | Interface |
| 3.5.1 | Clarification of dependent processes is needed. | Minor | Semantics |
| 3.5.2 | Support for process groups is needed. | Medium | Both |
| 3.5.3 | Proliferation of process husks is implied by the interfaces. | Minor | Semantics |
| 3.5.4 | Disposition of handles following process termination needs to be clarified and restricted. | Medium | Semantics |
| 3.5.5 | Parameter passing and inter-tool communication need to be re-evaluated. | Major | Both |

| Section | Item | Scale | Scope |
|---|---|---|---|
| 3.5.6 | Response is undefined when attempting to spawn a process that requires locked file nodes. | Minor | Semantics |
| 3.5.7 | Clarification of IO_Units and IO_Count with respect to meaning of Get and Put operations is needed. | Minor | Semantics |
| 3.6.1 | The use of predefined attributes should be clarified. | Medium | Semantics |
| 3.6.2 | Attribute values should not be restricted to List_Type. | Medium | Both |
| 3.6.5 | The order of Key and Relationship parameters should be reversed. | Minor | Interface |
| 3.7.1 | Enclosing string items in quotes decreases readability and is unnecessary. | Minor | Semantics |
| 3.7.2 | List_Utilities should present a textual rather than a typed interface. | Medium | Both |
| 3.7.3 | Token_Type should include all list items, not just identifiers. | Minor | Both |
| 3.7.5 | The Position parameter should never be required for operations on named lists. | Minor | Interface |
| 3.7.6 | Nested packages names conflict with Item_Kind enumerals. | Minor | Interface |
| 4.3 | Handling of control characters remains poorly defined. | Medium | Semantics |
| 4.4 | The Scroll_Terminal package provides improvements over Ada IO packages. | N/A | N/A |

implementation. Many of these comments reflect ambiguities in the text. Some major refinement of exception handling, input/output, and the list utilities is recommended. Other comments reflect specific technical areas and may be addressed by simple modification or addition to existing interfaces. While the required changes certainly appear to be within the scope of the planned upgrade, Version 2.0 of the CAIS will likely contain significant changes to the operational interfaces for tools. The most difficult problems to evaluate are the ambiguous areas of the specification which may simply disappear or which may result in considerable conflict depending upon the nature of the resolution that is adopted.

## MAJOR OBSERVATIONS AND RECOMMENDATIONS

The results of MITRE's prototype implementation of the Common APSE Interface Set support the Government's current policy for promulgating the CAIS. The CAIS provides a relatively consistent set of interfaces which address portability issues, but it is not refined to the degree that it can be mandated as a standard. The non-binding Military Standard CAIS issued 31 January 1985 publicizes the direction that the CAIS is taking. It can be used as guidance for current development efforts and provides a baseline for public critique.

An upgrade of the current definition of CAIS is planned. The new document, CAIS Version 2.0 will be an input to the Software Technology for Adaptable Reliable Systems(STARS) Software Engineering Environment program. It is intended that CAIS Version 2.0 have the quality and acceptance required of a true military standard. To achieve this quality, the upgrade will have to add rigorous precision to the current document, will have to refine several existing technical areas, and will have to include technical areas previously postponed.

CAIS Version 2.0 should be expected to contain major refinements and additions to the current document. The MITRE prototype effort has found five major issues that must be addressed in the next revision of the current document:

* The current document is ambiguous and imprecise--more rigor and precision is required.
* The List_Utilities abstraction can be made simpler, more complete, and more consistent.
* A central model is required for CAIS exception facilities.
* The CAIS IO model is not uniform-- it is inconsistent with Ada and with the CAIS node model
* The CAIS does not adequately address interactions between itself and the host operating system.

## RESOLUTION OF AMBIGUITIES

The precision with which the CAIS is specified in the current document leaves many issues open to the interpretation of the implementor. The semantics of many routines are not specified in detail; implications of alternate interfaces and suggested implementations are not addressed in text; broad statements are made in introductory sections and then are not reflected in discussions of specific routines; information on specific topics (such as predefined attributes) is dispersed throughout the document; and interactions among routines are not qualified. Together these deficiencies result in confusing the intentions of the CAIS and in giving an impression that the CAIS is not completely thought out. Unless corrected, they will make implementation of the CAIS difficult and standardization across CAIS implementations improbable. Clarification of the specification is also necessary to achieve the widespread acceptance necessary for adoption of CAIS as a standard.

## LIST UTILITIES REFINEMENT

During the most recent revision of the CAIS document, the List_Utilities package underwent significant modification. Further refinement is necessary. The List_Utilities package provides an abstraction that is used throughout the CAIS. Our recommendation is that the definition of Token_Type be expanded so that it can represent any of the list items currently supported (lists, integers, floating points, strings, and identifiers). This will allow the removal of redundant subprograms, will provide a more consistent interface, and will provide more functionality with less complexity. Enhancements to List_Utilities may allow the CAIS features that rely on List_Utilities to also be enhanced.

## CENTRAL EXCEPTION MODEL

The treatment of exceptions in the current document is inadequate. The Ada specifications do not correspond to the text, and the text references exceptions by unqualified names. The same exception name is used to refer to several different error conditions. Thus it is difficult to determine the complete set of CAIS exceptions and their relationships. It appears that exceptions were considered only on a procedure-by-procedure basis. A CAIS user will expect a single exception model that is consistent across the entire CAIS. We have proposed a candidate set of exceptions that addresses the entire CAIS and that reduces the instances of exceptions with multiple meanings. The method of exception handling in the Ada I/O packages could be adopted as a model for coordinating exceptions across several packages, or all exceptions could be declared in the package CAIS. However, the CAIS must evolve to one, consistent, well-engineered model for exception handling.

## CLARIFICATION OF THE I/O MODEL

The co-existence of both node handles and file handles makes the CAIS file nodes inconsistent with either process or structural nodes. The entire treatment of I/O facilities in CAIS suffers from its unclear relationship with Ada I/O facilities. Large sections of the CAIS I/O packages currently refer to Ada I/O packages without addressing specific effects of differences. While Ada defines distinct file types for Text_Io, Direct_Io, and Sequential_Io, the CAIS defines a single file type and indicates that operations from different I/O modes may be intermixed. However, many implications arising from this capability are not adequately addressed. The description of CAIS I/O would be greatly improved by discussing its intended compatibilities and differences with Ada I/O.

## CAIS AND THE HOST OPERATING SYSTEM

For an indefinite time, CAIS environments will be required to co-exist with the environment of the host operating system. It is unreasonable that all host facilities be converted to interface with a newly installed CAIS. Military Standard CAIS simply does not address issues related to this co-existence. Even the procedures for importing and exporting files between the two systems disregard important properties of host files and of CAIS files. Methods need to be established for reporting host errors, activating host processes, and making the contents of file nodes available to non-CAIS programs. Unless standards are established to integrate the host and CAIS environments, users of each CAIS will develop their own methods, and portability across CAIS implementations will be impacted.

**THE VIEWGRAPH MATERIALS**

**for the**

**M. McCLIMENS PRESENTATION FOLLOW**

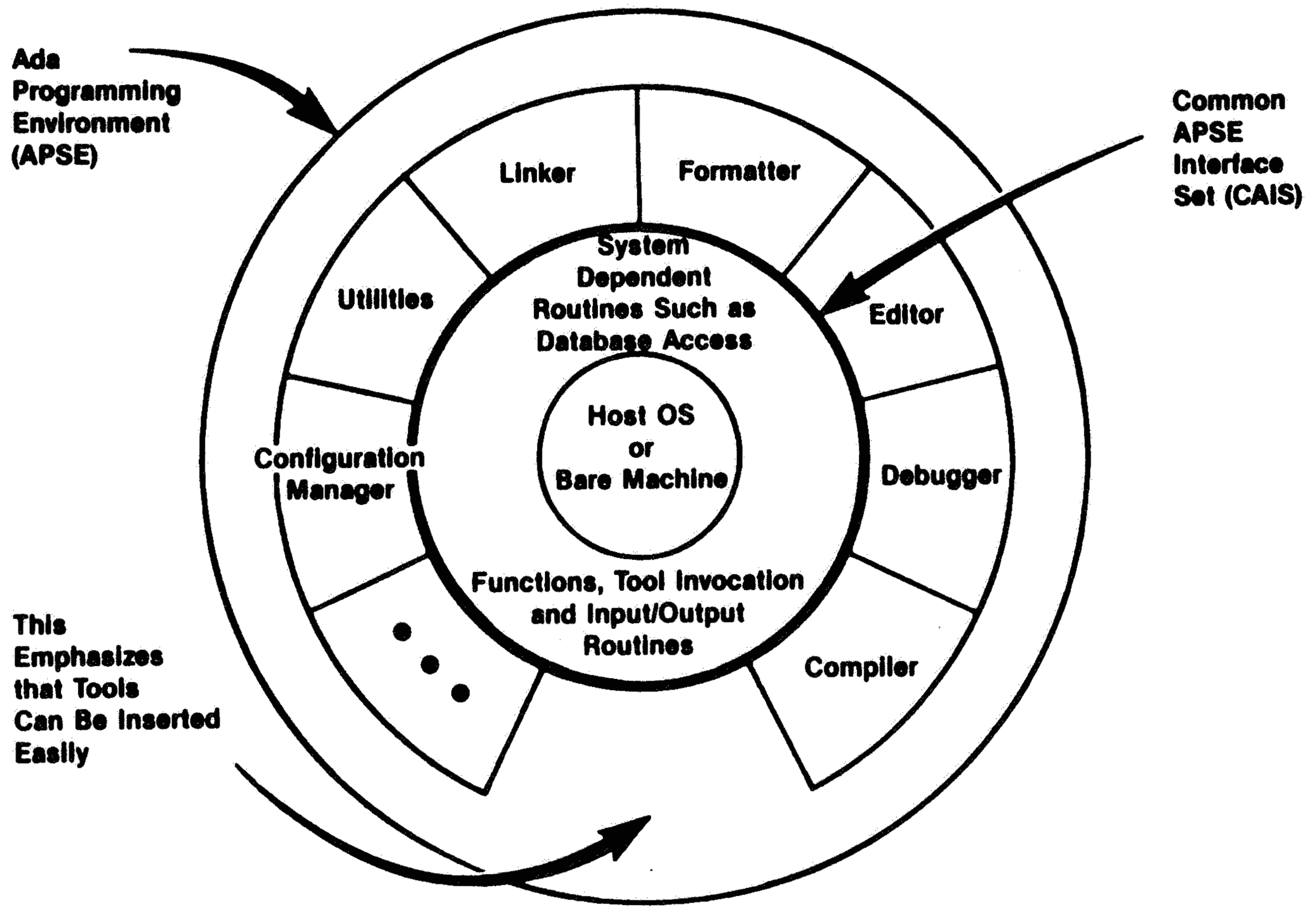
# MITRE CAIS PROTOTYPE

MITRE Corporation,
W33 Ada Software Eng.
IR&D

## Background

- ☛ Air Force Was Lead Service For Ada PSE (APSE) Development: The Ada Integrated Environment (AIE)
- ☛ Army Was To Develop A Few Critical Tools And Take Lead In Education
  - — Compiler And Tools Intended To Act As A Bridge Between Existing Environments And The AIE: The Ada Language System (ALS)
- ☛ Ultimate Goal Was The Development Of A DoD Standard APSE To Complement Ada

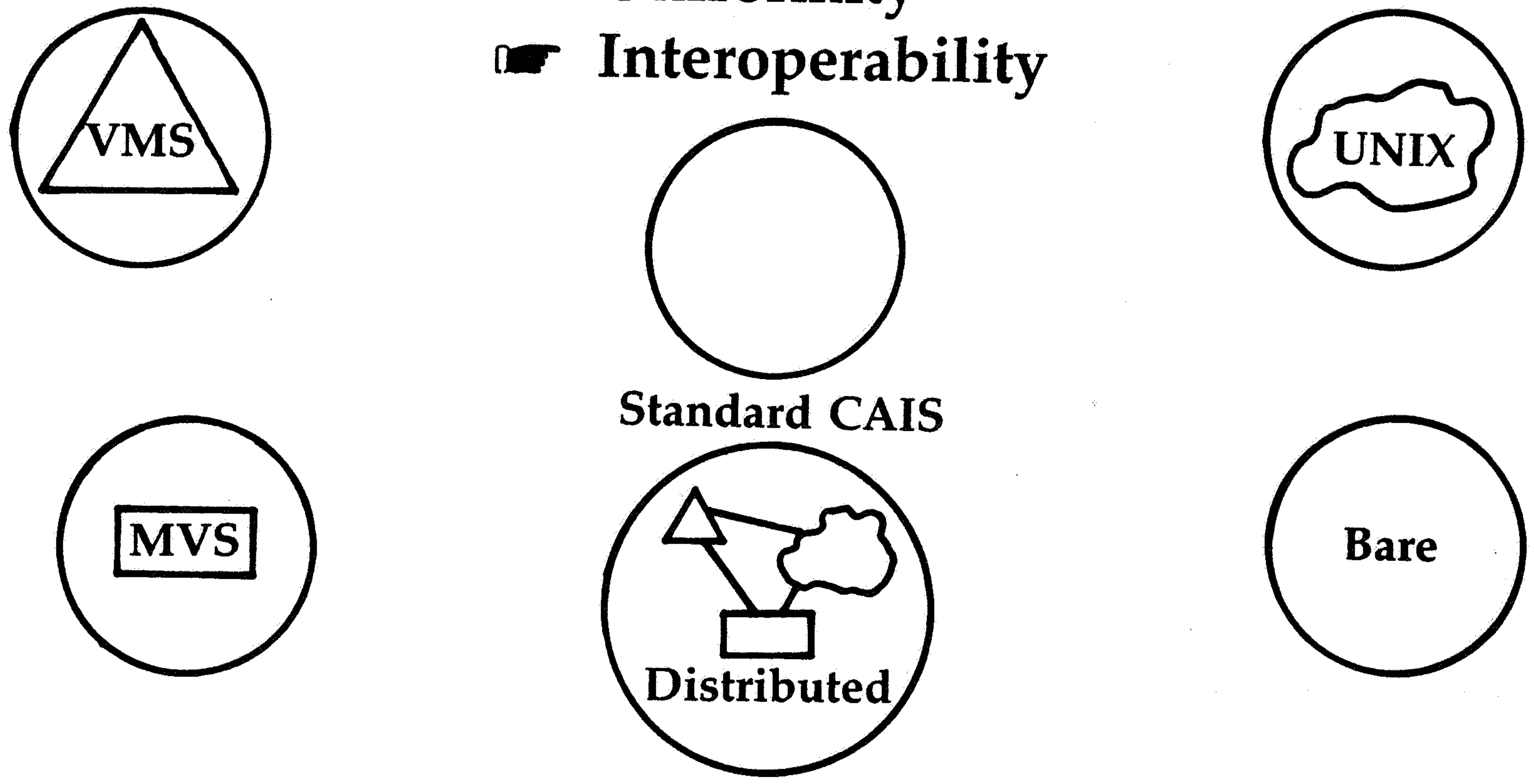
# APSE STANDARDIZATION



Ada Programming Environment (APSE)
Common APSE Interface Set (CAIS)
Linker
Formatter
Editor
Debugger
Compiler
Utilities
Configuration Manager
System Dependent Routines Such as Database Access
Host OS or Bare Machine
Functions, Tool Invocation and Input/Output Routines
This Emphasizes that Tools Can Be Inserted Easily

## CAIS Goals

- ☛ Portability
- ☛ Uniformity
- ☛ Interoperability



## Background

- ☛ Commercial Compilers Available On A Variety Of Systems
- ☛ WIS SDME
- ☛ NASA Space Station SEE
- ☛ STARS Software Engineering Environment (SEE)
- ☛ Several European APSE Efforts

## Background

- CAIS Working Group (CAISWG) Formed To Define Interfaces To Act As A "Bridge" Between ALS And AIE KAPSEs

- Goal Was To Support Tools Written In Ada That Use CAIS Services, Promoting Portability Between The ALS and AIE

## Overview of The CAIS

- ☛ CAIS Is Defined As A Set Of Ada Package Specifications And A Description Of Associated Semantics

- ☛ Underlying Model Is A Directed Graph With Attributes
  - — Nodes Can Be Files, Processes, Or "Directories"
  - — Graph Nodes And Edges Have Attributes

## What is in the CAIS

- ☛ Node Management
  - — Expanded File System
- ☛ Process Management
  - — Spawn/Invoke   Abort/Suspend/Resume
- ☛ I/O
  - — Text, Direct, Seq., Devices (Scroll, Page, ...)
  - — Import/Export
- ☛ List_Utilities
  - — Abstract Data Type
  - — Heterogeneous list of items

## What is not in the CAIS

- ☛ Support for Concurrency
- ☛ Memory Management
- ☛ Interrupts

} Ada

- ☛ Scheduling
- ☛ Paging/Segmentation
- ☛ Low Level I/O

} Operating System

- ☛ DBMS

## Background

☛ CAIS Has Evolved To A General System-Independent Operating System Interface

☛ Current Status of CAIS

— Controversy Over Perceived Premature Standardization

— Version 1 Is A Draft MIL-STD

— Explicitly for Prototyping Only

— Version 2 Contract Is Planned

- Funded Design Of CAIS

## Objectives of MITRE FY85 Work

- ☛ Select Subset Of CAIS And Implement Prototype
- ☛ Port Ada Tools To The CAIS Prototype
- ☛ Evaluate Implementability Of Specification
- ☛ Evaluate How Well The Interfaces Support Tools
- ☛ Provide Practical Input To CAIS Designers

## Technical Approach

- ☛ Ada/Ed Philosophy; Efficiency Is NOT A Priority
- ☛ Top Priority Is To Implement Subset Of CAIS
- ☛ Secondary Storage Is Used To Store Node Information
- ☛ Host (UNIX) Dependencies Are Isolated
- ☛ Using Verdix Ada Development System On ULTRIX

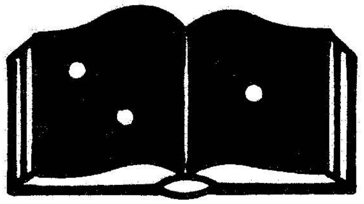
## General Comments

- Learning Curve For Using CAIS Will Be SIGNIFICANT
- Overall "Conceptual Consistency" Among Parts Is Good
- Implementation Is Uncovering Problems That Would Be Very Hard To Find By Inspection Of Specification
- CAIS Is A Good Vehicle For Continued Work In Standardized OS Interfaces And PSE/SEEs

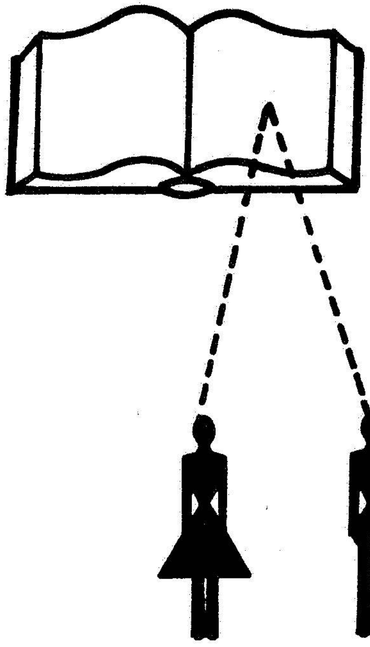
## Resolution of Ambiguities

☛ MIL-STD-CAIS needs to be Written with More Rigor

Completeness

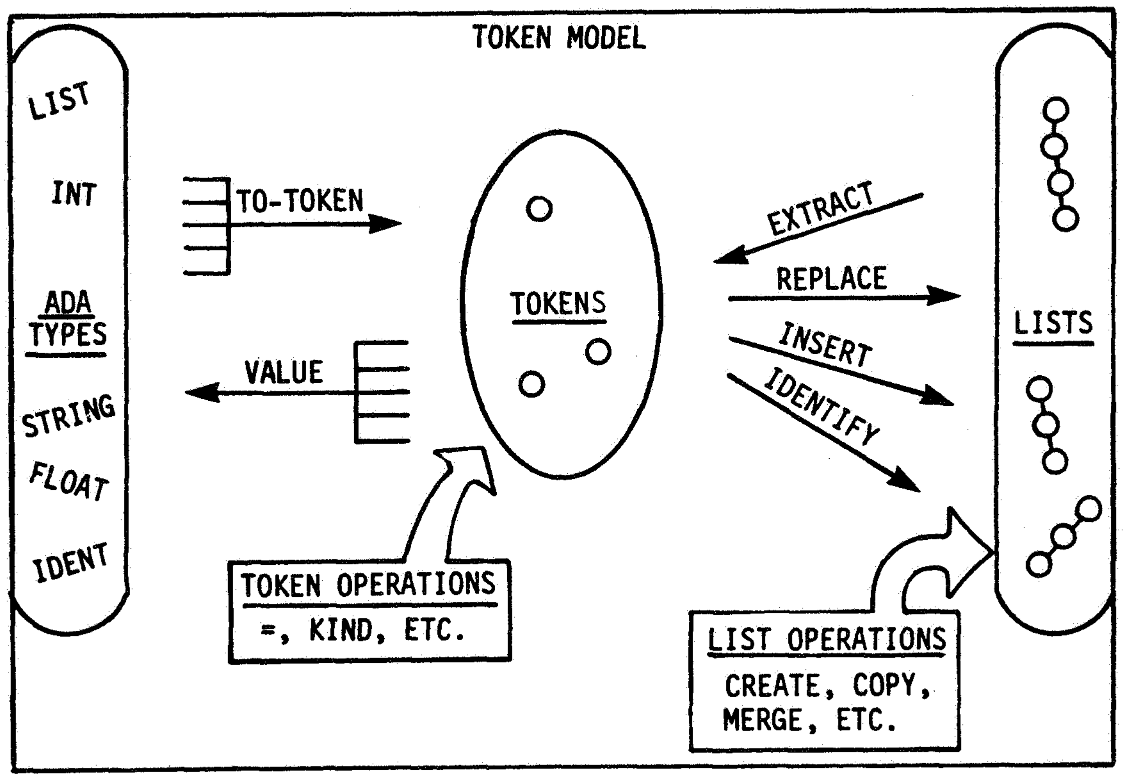


Precision



## List_Utilities Refinement

- The Current Model is Cumbersome and Inconsistent
- List_Types should Represent Lists of Tokens



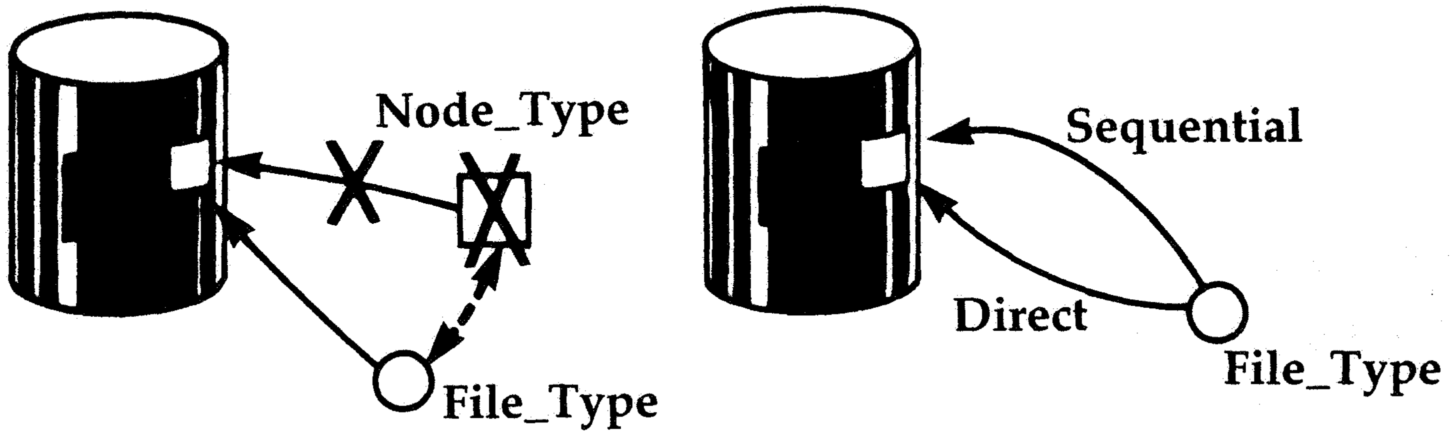
## Central Exception Model

- ☛ Currently, Addressed Procedure-By-Procedure
  - — Affects Clarity of MIL-STD-CAIS
  - — Affects User's Interface to CAIS

- ☛ The Set of CAIS Exceptions should be Centralized
  - — Declared Together
  - — Renamed in Each Package where they are used
  - — Names Refined to Establish a Clear Meaning

## Clarification of I/O Model

- ☛ File_Type Differs from Cais's Node_Type
  - — Affects Access Control
- ☛ File_Type Differs from Ada's multiple File_Type
  - — Affects Mixed Operations



## CAIS and the Host Operating System

- All Host Facilities won't be compatible with CAIS
- Access to Nodes and to Host Files must be Addressed
  - Expose / Escape Sequence for Host File Names

## Objectives for Rest Of FY86

- ☛ Extend Prototype
- ☛ Continue To Port Ada Tools to CAIS
- ☛ Port Prototype To VMS, Sun Workstations
- ☛ Port CAIS/UNIX Tools To CAIS/VMS
- ☛ Add Support For Distributed Processing
- ☛ Make The Source Widely Available