JOHNSON GRANT 13
IN-61 CR
3/2537

A Study of System Interface Sets(SIS) for the Host, Target and Integration Environments of the Space Station Program(SSP)
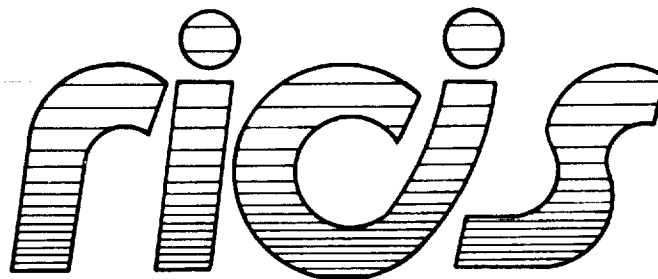
C. McKay

University of Houston - Clear Lake

D. Auty

SofTech, Inc.

K. Rogers

Rockwell International

June 30, 1987

rⁱcⁱs

T·E·C·H·N·I·C·A·L   R·E·P·O·R·T

The RICIS Concept

The University of Houston-Clear Lake established the Research Institute for Computing and Information systems in 1986 to encourage NASA Johnson Space Center and local industry to actively support research in the computing and information sciences. As part of this endeavor, UH-Clear Lake proposed a partnership with JSC to jointly define and manage an integrated program of research in advanced data processing technology needed for JSC's main missions, including administrative, engineering and science responsibilities. JSC agreed and entered into a three-year cooperative agreement with UH-Clear Lake beginning in May, 1986, to jointly plan and execute such research through RICIS. Additionally, under Cooperative Agreement NCC 9-16, computing and educational facilities are shared by the two institutions to conduct the research.

The mission of RICIS is to conduct, coordinate and disseminate research on computing and information systems among researchers, sponsors and users from UH-Clear Lake, NASA/JSC, and other research organizations. Within UH-Clear Lake, the mission is being implemented through interdisciplinary involvement of faculty and students from each of the four schools: Business, Education, Human Sciences and Humanities, and Natural and Applied Sciences.

Other research organizations are involved via the "gateway" concept. UH-Clear Lake establishes relationships with other universities and research organizations, having common research interests, to provide additional sources of expertise to conduct needed research.

A major role of RICIS is to find the best match of sponsors, researchers and research objectives to advance knowledge in the computing and information sciences. Working jointly with NASA/JSC, RICIS advises on research needs, recommends principals for conducting the research, provides technical and administrative support to coordinate the research, and integrates technical results into the cooperative goals of UH-Clear Lake and NASA/JSC.

Study of System Interface Sets (SIS) for the Host, Target, and Integration Environments of the Space Station Program (SSP)

Preface

This research was conducted under the auspices of the Research Institute for Computing and Information Systems by Charles McKay, Director of the Software Engineering Research Center (SERC) at the University of Houston-Clear Lake, David Auty of SofTech, Inc., and Kathy Rogers of Rockwell International

The views and conclusions contained in this report are those of the authors and should not be interpreted as representative of the official policies, either express or implied, of NASA or the United States Government.

University of Houston

Clear Lake

Director,
High Technologies Lab

Final Report

on

A Study of System Interface Sets (SIS) For the Host, Target, and Integration Environments of the Space Station Program (SSP)

Research Activity No. SE. 10

Contract # NCC 9-16

By

Charles McKay, Team Leader
Software Engineering Research Center

David Auty
Softech, Inc.

Kathy Rogers
Rockwell International

30 July 1987

CONTENTS

Relevant Quotations From the Authors of This Report and Related Memos:

1. "Although CAIS is believed to be necessary and extensible, it is certainly not sufficient for the SSE. However, it represents very good work by good, experienced people. The underlying conceptual model is sound. Any attempt to define an adequate SIS for the SSE must cause the designers to come to grips, eventually, with the same issues faced by the CAIS designers. It would be far too expensive, time consuming, and risky to ignore this body of work by 'reinventing this wheel'."

2. "Building-in and sustaining safety in large, complex, non-stop, distributed systems is not simple. Nor can this be guaranteed. Success requires depth-of-knowledge of a number of interrelated subjects..."

3. "For target subsystems which life and property depend upon, any attempt to piggy-back....on inappropriate models and paradigms of an obsolete operating system is too risky!!! (e.g., UNIX V.X., BSD4.X etc.)"

4. "The one apparent certainty determined by the SERC team and others is that the unit of software distribution supported by the PCEE must be below the program level and at least to the task level. Otherwise, the risk of being unable to sustain the SSP life cycle requirements for both mission and safety support is unacceptably amplified."

5. "Working Definition
   Safety: The probability that a system, including all hardware and software and human-machine subsystems, will provide appropriate protection against the effects of faults which, if not prevented or handled properly, could result in endangering lives, health, property or the environment. (CWM, July, 1987)"

6. "The terms 'secure UNIX' and 'safe UNIX' are temporal oxymorons unless:

    1)   all features of UNIX are hidden beneath the virtual interface set of a strongly typed language such as Ada <u>and</u>

    2)   application programmers are denied access to assemblers and other untyped languages."

Preface and Acknowledgements

- KIT and CAIS-A Design Team
- The ESPIRT/PCTE Design Team of the Commission of European Communities
- Mark V Systems, PCTE Ada* Specification Team
- MITRE, CAIS Research Team
- CAIS designers from: IBM FSD, Gould, Canadian Armed Forces, TRW, and others in the CAIS Implementors Working Group.

The following is a list of trademarks used throughout this document.

- *Ada is a registered trademark of the US Government, Ada Joint Program Office.
- *UNIX is a trademark of AT&T Bell Laboratories.

The focus of this report is on Systems Interface Sets (SIS) for large, complex, non-stop, distributed systems, such as the Space Station Program (SSP), which incrementally evolve over a long period of time and have an indefinite life cycle. This research team is convinced that the traditional division of issues into host environment versus target environment is inadequate to scale-up to meet the needs of the SSP. Instead, the subdivision must be expanded to encompass the issues integrated across three distinct sets of environmental activities and responsibilities:

- Host (Where software for the target environment is developed and sustained.)
- Target (Where the executable versions of the software developed in the host environments are to be deployed and operated.)
- Integration (Where the configuration of the current target environment baseline is controlled. This environment is responsible for the test and integration plans used to interactively advance the target environment baseline with approved changes in software emanating from the host environments. This environment is also responsible for controlling interactions with the target environment to maximize safety during emergencies.)

All three environments have requirements for User Interface Sets (UIS) and System Interface Sets (SIS) where the UIS refers to the human-system interfaces and the SIS refers to the interfaces of the application software and command language to the underlying system software and hardware resources. Although the requirements for the UIS and SIS have some key differences among the three environments, an integrated perspective reveals a strong core of commonality which can and should be exploited to enhance the life cycle management of SSP:

- complexity
- safety and quality for both systems and software
- cost effectiveness for both systems and software
- technology transfer into other applications to enhance productivity, sustain safety, improve quality, and improve cost effectiveness.

The SIS of the SSP was selected as the focus of this study because an appropriate virtual interface specification of the SIS is believed to have the most potential to free the project from four life cycle tyrannies which are rooted in a dependance on either a proprietary or particular instance of:

- Operating Systems
- Data Management Systems
- Communications Systems
- Instruction Set Architectures

The SIS allows tools, rules, application software, test software, command language scripts, etc. to be developed/acquired on a foundation of the virtual interface specifications rather than on the physical interface specifications of the underlying operating system, etc. Please note that this research team is convinced that the avoidance of these four tyrannies is an absolute requirement if the life cycle goals of the Space Station Program are to be accomplished under the currently anticipated constraints of: adequate availability of appropriately qualified personnel, support of intended missions, budgets, and schedules.

Functional requirements are the primary drivers in constructing host environments. That is, resources are configured and controlled to maximize the productivity of computer systems and software engineers, programmers, and management in the phases and activities of developing and sustaining software for the target environment. Nonfunctional requirements (i.e., constraints on the implementations of the functional requirements) are far less of a driver in the establishment of requirements for the host environment's UIS and SIS. For example, the host SIS typically exists in an earth-based environment without stringent requirements on tightly constrained real time operations; fault tolerance; limitations of electrical power; volume limitations; etc. By contrast, the SIS of the target environment must be strongly influenced by the nonfunctional requirements imposing constraints upon: real time deadlines, fault tolerance, power and volume availability, etc. The SIS of the integration environment is driven by a balance of functional and nonfunctional requirements which allow it to interact with the target environment for performance monitoring; reconfiguration; on-orbit integration to advance the target environment baseline; symbolic debugging and critical control support of safety during emergencies.

Although the UIS is not the principal focus of this study, a few observations may be useful. For example, the UIS of the host environment should emphasize functional requirements to enhance programmer productivity. This implies an emphasis on: syntax-directed, template-driven editors; window management; graphic design aides; and other enhancements for professionals who develop and sustain software. By contrast, the UIS of the target environment should be optimized not for software professionals but for users and mission specialists who desire a natural interface to their application domain. Similarly, the UIS of the integration environment provides needed support for command language querying and interaction with system components of the target environment under normal and emergency conditions.

To understand the issues of both the unique requirements and the commonality requirements of the SIS among the three environments, two macroscopic perspectives are useful. The first perspective is the "static" viewpoint which encompasses all host environment phases and activities from systems requirements analysis up to and including the preparation of the executable versions of the software which are to be deployed and operated. Tool builders/acquirers typically have this perspective. The SIS of the CAIS is strongly influenced by this perspective.

The second macroscopic perspective is the "dynamic" viewpoint of what happens during the execution of programs. This is where the subset of the SIS requirements described in CAIS should be augmented both by complementary extensions needed for the SSP and, of even more importance, by the dynamic requirements of a Portable Common Execution Environment (PCEE) capable of maximizing the ability to sustain safety while fulfilling mission requirements in a distributed system with non-stop components. For example, are there enough resources in the execution environment to handle peak workloads and safety critical emergency conditions? Can safety be sustained and meet mission requirements when certain classes of faults are encountered, etc.?

Please note that whereas static viewpoints pertain primarily to the tools and rules of the host environments, dynamic viewpoints must cross all three environments. That is, whereas the SIS-based tools and rules for developing and sustaining software are often uniquely located in the distributed host environments instead of target environments, execution issues cannot be unique to the target environment for at least two major reasons. First, verification issues including: testing, quality assessment, simulation, and emulation (all of which require execution in the host environment) must take place to an appropriate degree before code can be trusted to be transitioned through the integration environment into the target environment for deployment and operation. Second, new tools and rules are likely to be needed throughout the life cycle of the host environment. These tools must be developed and executed in the host environments to be effective. Thus, although tool builders and tool configuration managers are likely to emphasize a static viewpoint in describing the UIS and SIS of systems such as the SSP; mission administrators, specialists, operators, users, safety engineers, and quality managers are likely to emphasize the viewpoint of the dynamic aspects of the three environments.

This report recommends the adoption of CAIS as an extensible baseline for the SIS of the host and integration environments of the SSP. However, the reader should note that the SIS requirements of CAIS strongly reflect the predominantly static viewpoint of tool builders and tool administrators. As such, they are only an extensible subset of the overall SIS

requirements needed for SSP. This report further recommends that a definition and appropriate test bed support be established as soon as possible for a Portable Common Execution Environment (PCEE) that reflects the execution environment requirements of activities which are unique to the needs of a particular environment as well as those which are integrated across the three environments (i.e., the PCEE must be both tailorable and extensible.) In particular, recent SERC research which has produced the "Clear Lake Model for Life Cycle Support of Computer Systems and Software Safety in the Target and Integration Environment of the Space Station Program" has identified requirements for a "safety kernel" execution environment composed of a minimum of 12 highly interdependent models of key components underlying a PCEE such as: management of distributed, nested transactions; management of unique identifications for objects, transactions, and streams; dynamic, multilevel security; redundancy management; etc. The requirements for the high degree of interactions and interdependencies among these 12 component models which underlie the SIS of an execution environment are not incompatible with the less stringent requirements that emerge from the more static viewpoint of tool builders and tool administrators proposed in the SIS of CAIS. However, since these additional requirements are clearly issues-at-risk on the critical path of major SSP phases and components and since none of the test bed activities now underway are specifically focused on these issues, appropriate action is needed soon.

Chapter 1 of this report provides: background and overview information with a set of working definitions and explanations of key terms and concepts used throughout the report. Chapter 2 focuses primarily on the static perspective of CAIS and PCTE activities. Chapter 3 focuses primarily upon the dynamic perspective of a PCEE. Chapter 4 provides a summary and recommendations. Appendices and a bibliography are attached to further clarify issues addressed in the report.

1.0 Background

This report is based upon the lessons learned from:

- Reviewing and interacting with the development of the current CAIS (MIL-STD 1838)
- Interactions with other designers proposing distributed versions of the CAIS
- Interactions with the design team now evolving CAIS-A
- Interactions with the design team of the Portable Common Tool Environment (PCTE)
- Interactions and participation in the evolution of the Ada RunTime Environment Working Group (ARTEWG) Catalog of Interface Features and Options (CIFO)
- Development of the Clear Lake Model for Distributing Entities of Ada Programs
- Development of the "Clear Lake Model for Life Cycle Support for 'Computer Systems and Software Safety' in the Target and Integration Environments of the Space Station Program".

1.1 Intended Audience

This report should be of use to at least the following groups:
- All NASA offices concerned with computer automated systems. In particular, NASA SSP offices from Level A and Level A' down to the offices administrating SSP test beds.
- SSE contractors, management, and users
- TMIS contractors, management, and users
- DDT&E contractors, management, and users
- PSC contractors, management, and users
- SSP international partners, contractors, managers and users.

1.2 Overview of Key Terms and Concepts in a Hierarchical Development Order

1.2.1 Services and Resources:

Service: Work done or duty performed for another or others (Webster's Unabridged, 1987)

Resource: Something that lies ready for use or can be drawn upon for aid.
A supply of something to take care of a need, e.g., coal (Webster's Unabridged, 1987)

Discussion: As used in this report, the word "service" conveys a sense of "active" work performed whereas "resource" conveys a sense of being "passively" available for use.

1.2.2 Object:

Any logical or physical entity with an abstract specification which answers the following questions:
- what services and resources are provided or consumed by the object
- how well are they provided
- under what circumstances are they provided

(See Figure 1-1)

Discussion: This abstract specification can be considered to provide a "virtual interface" to the object. From the perspective of users, the abstract specification also provides the "one way in/one way out" for the object. Since objects are intended to communicate by messages, the virtual interface can be used as an inventory of the message requirements. Abstract specifications are always to be separately compilable from the implementation part of the object if an implementation part exists.

The implementation part, if it exists, hides all design decisions regarding the object. Trade-offs between: hardware and software, algorithms and data structures, and traditional versus AI design techniques are encapsulated inside this object implementation. Other information which may also be hidden inside the implementation part may include knowledge that this is a complex object which is composed of several other objects. For example, a component written in assembly language can be encapsulated inside an object body. Since users of the object can only see the abstract specification, access to the code segments and data structures of the assembly language routine would be hidden from the users and controlled inside the implementation part.

```
┌─────────────────┐              ┌─────────────────┐
│    abstract      │  ◄───       │    abstract      │
│  specification   │      ╲╱  ──► │  specification   │
└─────────────────┘       ╱       └─────────────────┘

                    Communicate
                    with messages
                    via abstract
                    specifications

┌─────────────────┐              ┌─────────────────┐
│                  │              │                  │
│ implementation   │              │ implementation   │
│                  │              │                  │
└─────────────────┘              └─────────────────┘

      object 1                          object 2
```

FIGURE 1-1 TWO COMMUNICATING OBJECTS

1.2.3 Entity Attribute/Relationship Attribute Models:

Represent domains of interest in terms of:
- the entities within each domain
- relevant attributes of these entities
- the relationships among the entities
- relevant attributes of these relationships

Discussion: Although the Entity Attribute/ Relationship Attribute (EA/RA) model was introduced by Dr. Peter Chen in 1976 as a data base modeling approach which captured additional semantic information (meta-data) beyond the techniques of hierarchical, CODASYL and relational data bases, the EA/RA approach was quickly recognized for its power and parsimony. Its use has now spread into almost the entire spectrum of computing.

When objects are mapped to entity representations, the approach is particularly powerful. Since each object embodies the software engineering principles of abstraction, modularity, information hiding, and localization, an EA/RA model can be used to passively represent a design structure depicting the relevant objects, their relationships, and the key attributes that represent the design. Analysis for completeness, consistency, etc. is greatly facilitated. Also of great importance is the ability to leverage the rich semantic information captured by the model to understand the potential effects of proposed changes to the baseline design. The model is also bringing discipline and order to what had previously been a collection of ad hoc approaches and guidelines (typically insufficient) for developing and sustaining on-line schemas, subschemas, and dictionaries.

Two more recent developments which hold great promise for future computing systems include:
- the use of on-line instances of EA/RA models to actively enforce strong typing, integrity controls, and multiple views for access and context control.

4

- the draft ISO (International
  Standards Organization) standard
  for IRDS (Information Resource
  Dictionary System). This standard
  permits on-line instances of EA/RA
  models to be used among
  heterogeneous computing systems of
  different vendors.

1.2.4 Stable Interface Sets:

Used to represent and sustain baseline control of an
interface set. They must satisfy three requirements:
1. From the perspective of a complete inventory,
   an SIS is a virtual interface set resulting
   from the union of all abstract specifications
   of the objects designed to be visible at this
   interface. (That is, the total collection of
   all the services and resources to be provided
   at the SIS, how well and under what
   circumstances.)
2. From the perspective of any given object
   within an SIS regarding its relationship to
   other objects in the SIS, there should exist
   a formal model in EA/RA form describing:
   - the "need-to-know" and "right-to-know"
     visibility among all objects of the SIS
     and
   - the grouping of objects into discreet
     sets of services and classes of services
     within each of the sets.
3. From the perspective of external objects
   outside the SIS, a formal model in EA/RA form
   is used to describe:
   - exactly what sets and classes of
     services are to be visible to the object
   - the protocols (necessary steps) and
     access control to be imposed upon the
     external object and its use of the
     services and resources provided at the
     SIS.
(See Figure 1-2.)

5

VIEW 1:
An Inventory Perspective of the Total Visibility Available at the SIS

classes 1, 2, 3, of service set 2

classes 1, 2

SIS

class 2 of service set 1

class 1 service 1

objects not
seen at
interface

Figure 1-2a

6
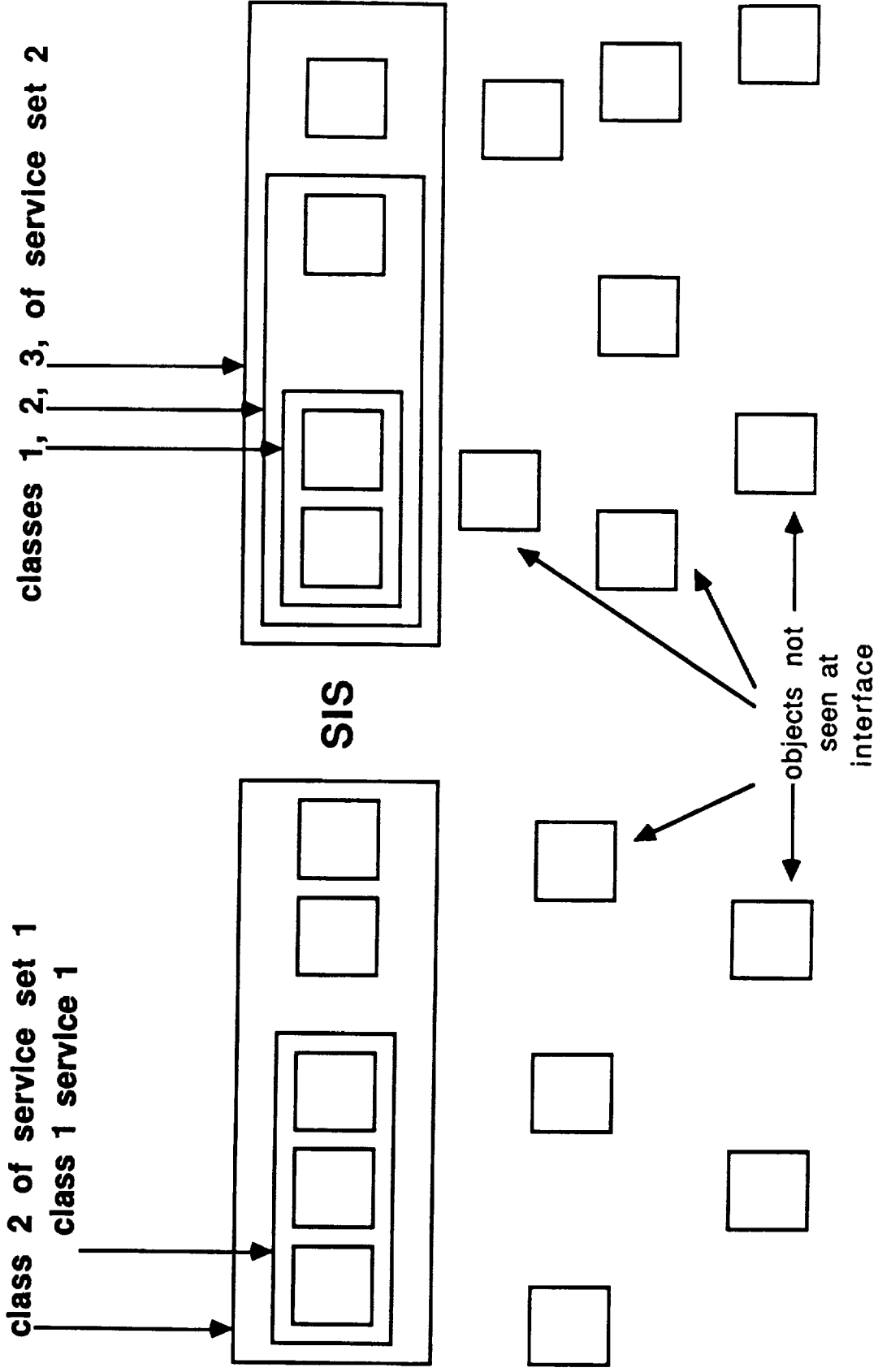
VIEW 2:
EA/RA Model of SIS Objcets Plus Required Supporting Objects Underneath

SIS

objects not
seen at
interface

7
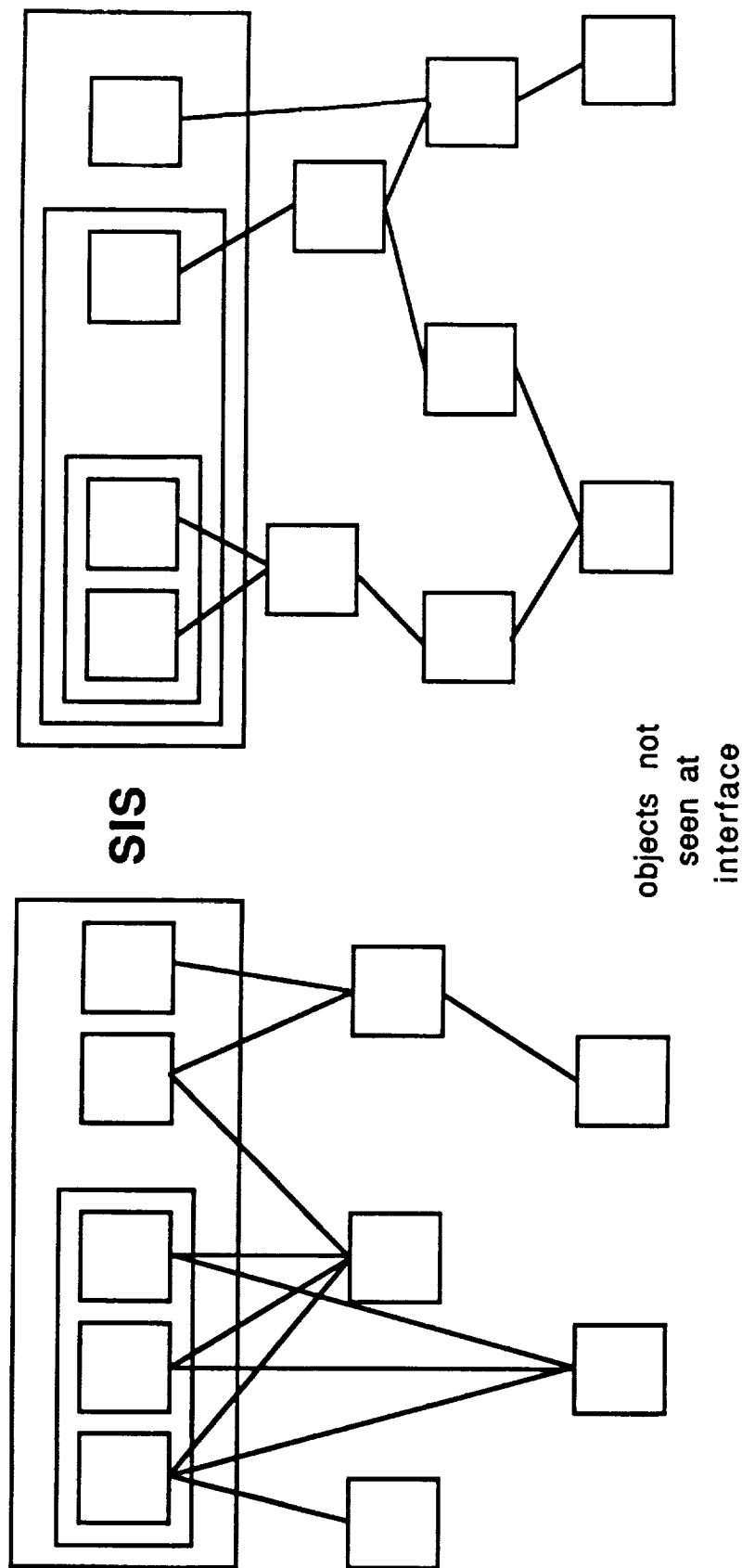
Figure 1-2b

VIEW 3:
EA/RA Model of Visibility and Access Control of Objects Outside SIS

Object external to SIS

Visibility of Class 2 of both Service Sets

Object external to SIS

Visibility of Class1, Service Set 1

SIS

objects not
seen at
interface

Service Set 2
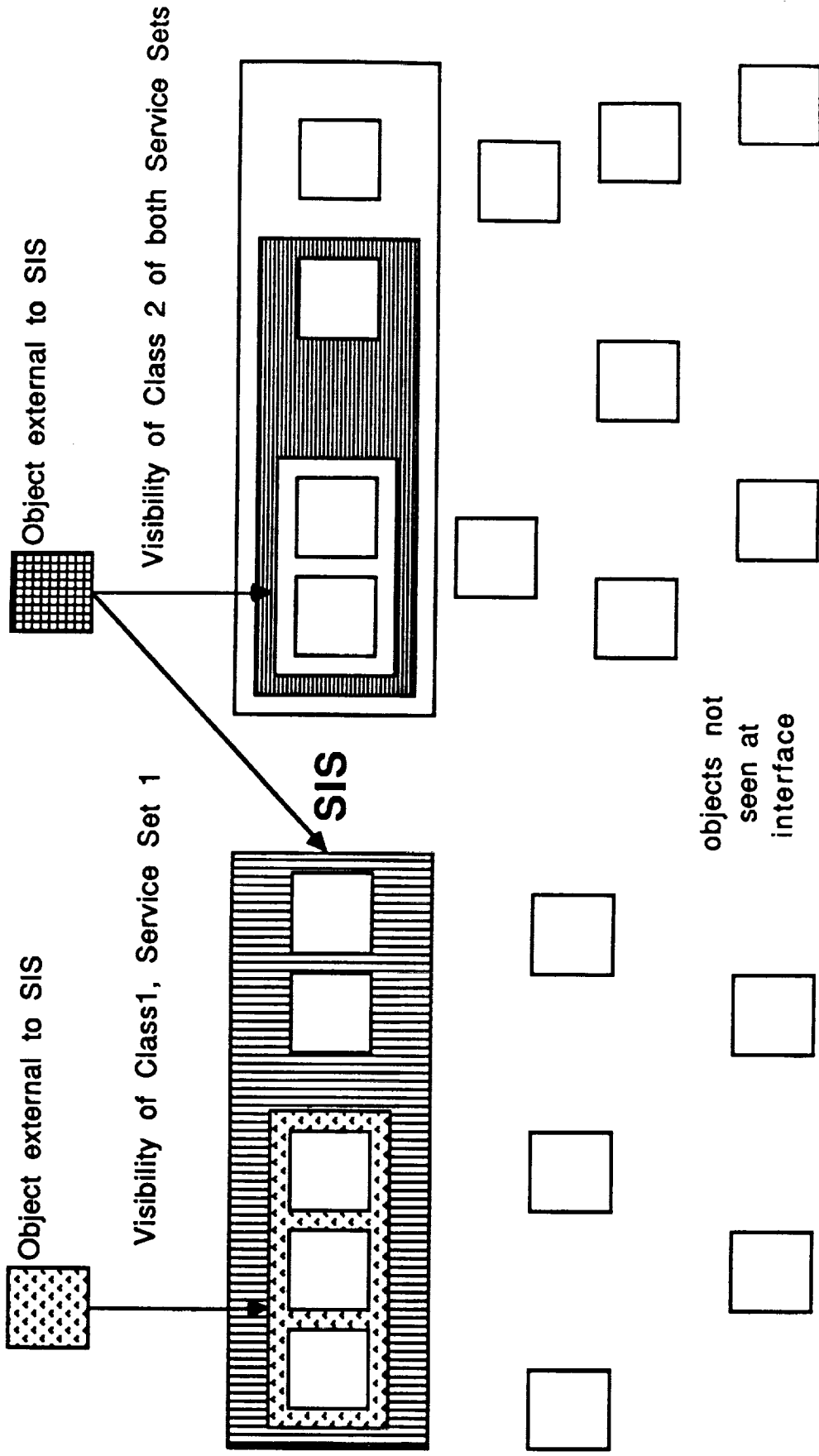
Service Set 1

Figure 1-2c

1.2.5 Layering:

Refers to a hierarchical structuring approach featuring enforced access to the services and resources of each layer through the layer's interface specifications. Thus an entity in layer "N" which is authorized to access a service of layer "N - 1" can only access the service by complying with the protocol requirements associated with the interface specification of the service. For baseline control, systems should be constructed in a hierarchy of layers where each layer is sustained as a Stable Interface Set. (See Figure 1-3.)

Two Layers of STABLE INTERFACE SETS

SIS Layer
N + 1

SIS Layer
N

Figure 1-3

10
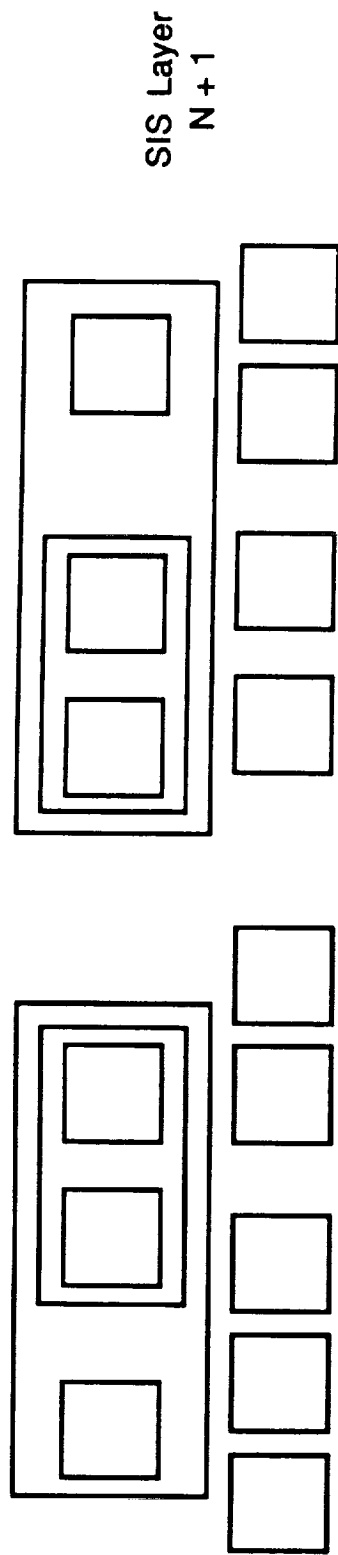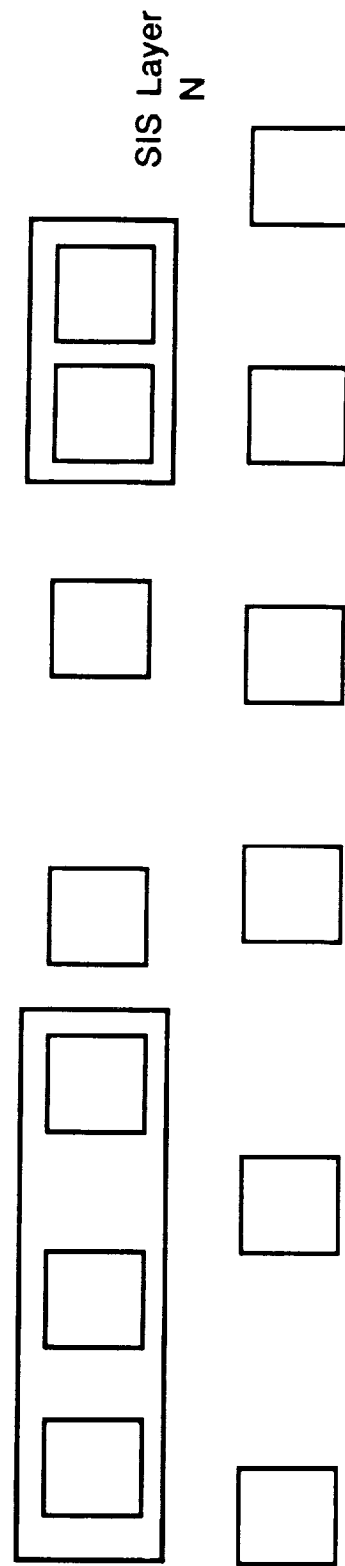
Discussion: Layers should not be confused with the structuring approach known as "Levels" (e.g., as used in UNIX). Levels also group services and resources into associated sets of functionality that are described by interface specifications. However, no automatic enforcement of access via the interfaces is typically available. Furthermore the protocols generally allow a service in level N to be accessed from anywhere (as opposed to restricting access to a few authorized entities in the adjacent layer). Because the interface specifications for levels are usually provided in an untyped language, any enforcement of structural integrity must rely strongly on off-line guidelines, practices, and shop standards.

Used properly, layering contributes to firewalling of faults that occur within a layer. Upper layers are often able to compensate for faults which occurred in lower layers. The successive layers of thrust explicitly reflect both the functional architecture of the system and the differing degrees of criticality associated with the functionality of the layer. Security, integrity, and reliability are three critical concerns that can now be addressed appropriately within each layer.

1.2.6   Stable Frameworks:

The three requirements for Stable Frameworks (SF) are:

1.   Within any layer a collection of closely related objects that should be regarded and maintained as a unit shall be identifiable by unique attributes.
2.   The collection of objects within a layer which are to be identified as a unit via the unique attributes can be treated as "strongly typed". That is, a complete determination can be made of legal values and legal operations for SF's of this type.
3.   Within any layer, a formal model in EA/RA form can be used to represent the relationships of the strongly typed SF's to other strongly typed SF's both within and external to the layer.
(See Figure 1-4.)

11

Stable Framework within a Layer

Stable Framework Among Three Layers

SIS Layer N+2

SIS Layer N+1

SIS Layer N
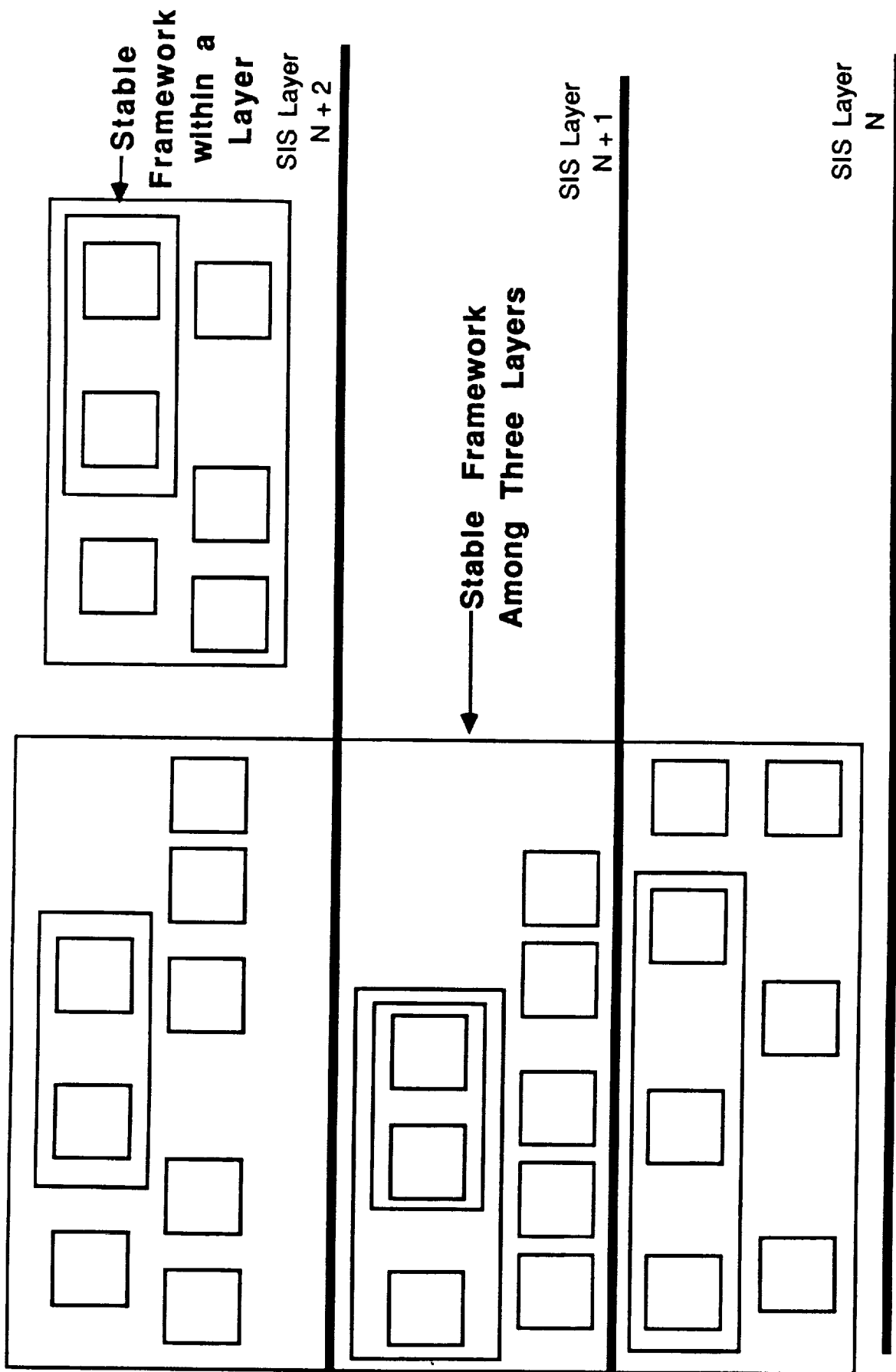
Figure 1-4   Stable Frameworks

12

1.2.7 Conceptual Models:

At a level of abstraction appropriate to facilitate understanding and communication among the intended audience, the conceptual model is an invaluable aid to explain the level-of-detail relevant to a level of decomposition of the major constituents of the solution or problem space under review. The current level's constituent entities, attributes, relationships to other constituent entities in the decomposition, and the attributes of these relationships are made visible. The transformations that have mapped this level from the previous level of decomposition are traceable. A foundation is also laid which will render the transformation to the subsequent level traceable. (Figures 1-5 and 1-6 could be augmented with appropriate documentation to serve as examples.)

TWO SCENARIOS FOR
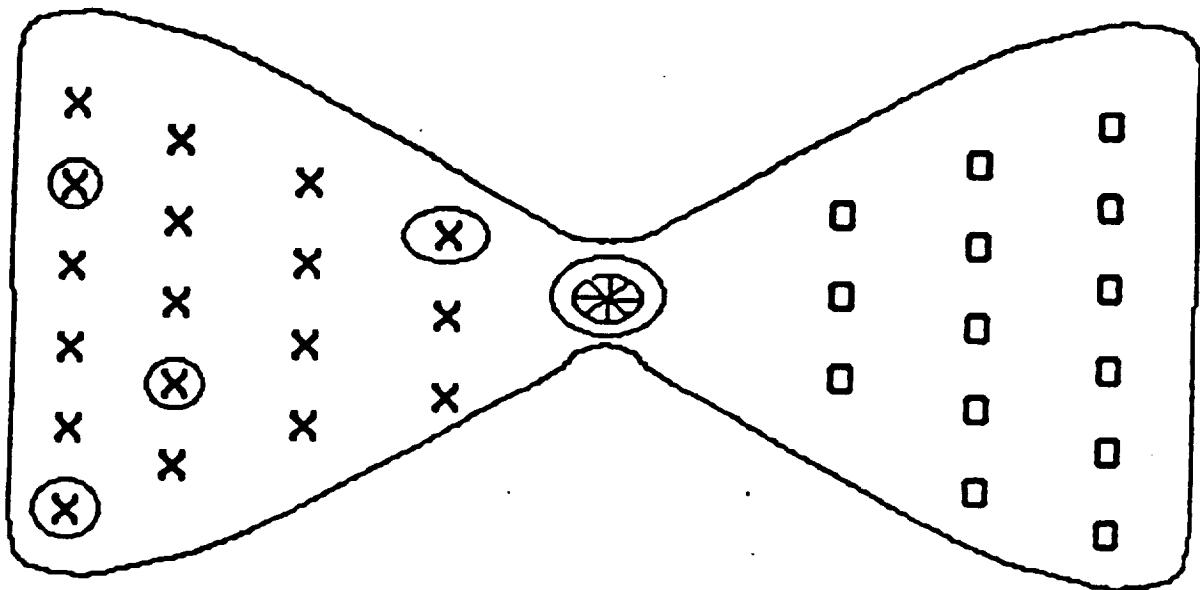
SSP ENVIRONMENT

IN 2000+ A.D.

HOST ENVIRONMENTS:	INTEGRATION ENVIRONMENT:	TARGET ENVIRONMENTS:
• DEVELOP • SUSTAIN	• CONTROL OF TGT. ENVIR. BASELINE • INTEGRATION V&V FOR NEXT BASELINE AND TEST & INTEGRATION PLANS	• DEPLOY • OPERATE

FIGURE 1-5 THREE SOFTWARE ENVIRONMENTS

FIGURE 1-6

A Model for Supporting a 'Bare Machine' Philosophy for 'Safety Kernels' of Ada Runtime Support Environments (Ada RTSE's)

Application Program Perspective

Note : Explicit Visibility

Note : Transparency

Applications Library

XRTL

RTL

Tool Set for Library & Object Base Mgmt.

Ada Source

compiler

C

T

Persistent Object Base

> 2%?

Object Code & C.L. IF

SIS

RTL | XRTL

RTK

Hardware

Host Environment

Integration Environment

Target Environment
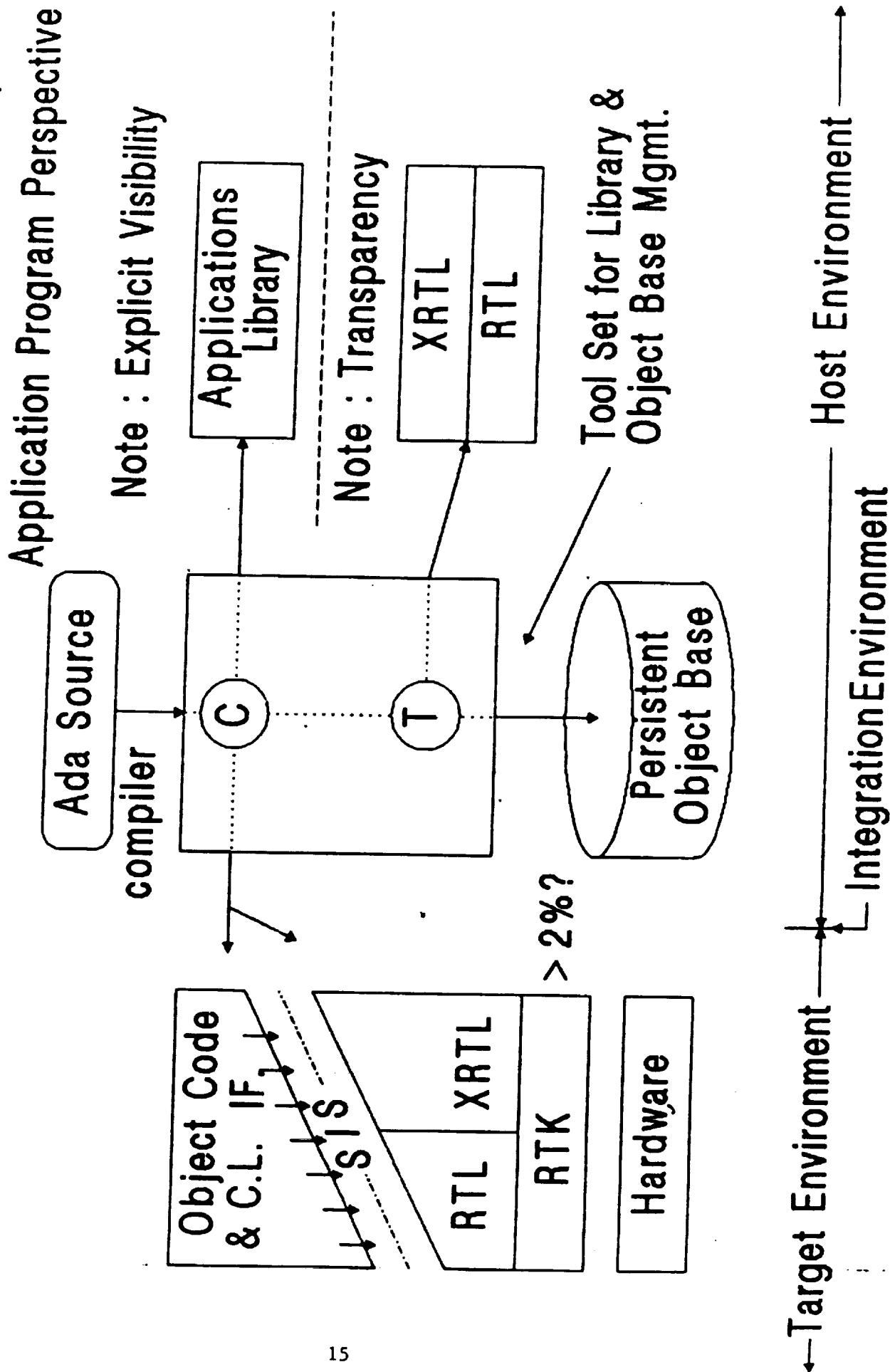
1.2.8 Environments: Host, Target, and Integration

Host Environment

- Software Life Cycle Only: Software for the Target Environment is developed and sustained in this environment. Deliverables for the Target Environment are provided to the Integration Environment which is responsible for installation, integration, and operation.

- System Life Cycle: Software, hardware, and human-system interfaces are developed and sustained in this environment. Deliverables for the Target Environment are provided to the Integration Environment which is responsible for installation, integration, and operation.

  Note that the Software Support Environment (SSE) of the SSP has system life cycle responsibilities in spite of its use of "software" instead of "systems" in the title.

Target Environment

- Software Life Cycle Only: Software developed in the Host Environment and commands emanating from the Integration Environment are deployed and operated here.

- System Life Cycle: (A similar distinction should be made as in the discussion of the Host Environment.)

Integration Environment

- Software Life Cycle Only:
  - Accepts software from the Host Environment and performs any final verification, validation, and quality assessment activities prior to installation and deployment in the Target Environment.

  - Responsible for administration of test and integration plans to interactively advance the Target Environment baseline.

  - Controls and monitors operations in simultaneous support of both mission requirements and safety requirements.

  - Controls and monitors the current software

16

baseline in the Target Environment during both normal and emergency operations.

- System Life Cycle: (A similar distinction should be made as in the discussion of the Host Environment.)

Discussion: The authors are convinced that, for large, complex, non-stop, distributed systems such as the SSP,
- the traditional division of issues into host and target is insufficient. The division into the three sets of environments described above is critical for the life cycle.

- Both the Host (i.e., SSE) and Integration Environments must support the systems resources of the Target Environment rather than just the software issues. Specifically, the traceability and management of all software, hardware, and human-system interfaces throughout the life cycle must depend upon an integrated approach to the on-line representation and control of these baselines in the target environment.

1.2.9 Environment Perspectives: Static and Dynamic

Static: Encompasses all Host Environment phases and activities from systems requirements analysis up to and including the preparation of the executable versions of the software which are to be deployed and operated in the Target Environment.

Dynamic: Encompasses all that happens during the execution of commands and executable versions of programs.

Discussion: Tool builders and acquirers typically emphasize a static perspective of the functional requirements of the Host Environment. By contrast, the simultaneous support in the Target Environment of both mission requirements and safety requirements is impossible to develop and sustain without a dynamic perspective that considers the balance of functional and nonfunctional requirements.

17

In this report, the authors recommend a Portable Common Execution Environment (PCEE) be defined as a complement to the more statically influenced standards for System Interface Sets such as CAIS (Common APSE (Ada Programming Support Environment) Interface Set) and PCTE (Portable Common Tool Environment). (See Figure 1-5 and the following discussion in Section 1.2.10.)

1.2.10   Host Environment Stable Interface Sets: CAIS, CAIS-A, ARTEWG CIFO, PCTE, PCEE

- CAIS: (MIL-STD 1838) Common APSE Interface Set. Provides a standard for the Systems Interface Set of host environments that is intended to promote transportability and interoperability. A primary focus is to facilitate this promotion by providing appropriate support at the SIS for tool builders.

- CAIS-A: Now being defined as an intended successor to CAIS. Specifically, it is intended to address many of the deferred subjects in CAIS such as distribution.

- ARTEWG CIFO: Ada RunTime Environment Working Group Catalog of Interface Features and Options. The ARTEWG is recognized and partially supported by the Ada Joint Program Office with a charter to address Ada runtime environment issues. Subgroup 3 (currently chaired by one of the authors) is responsible for evolving CIFO.

- PCTE: Portable Common Tool Environment. This work was sponsored by the Commission of European Communities. Although the work has many goals in common with the CAIS work, there are many notable differences in both goals and results. (Several of them are described in Chapters 2 and 3 of this report.)

- PCEE   Portable Common Execution Environment. This is a SERC Team proposal to define a standard for the SSP reflecting the execution environment requirements which are unique to each of the three

18

environments (host, target, and integration) as well as those that are integrated across the three.

Discussion: The standard should be in the form of Ada specifications in a Catalog of Interface Features and Options similar to the slowly evolving ARTEWG CIFO. The System Interface Set of the PCEE would be complementary to CAIS and CIFO but would have a distinct emphasis on dynamic issues to maximize the simultaneous support of both mission and safety requirements in the Target Environment of SSP. An essential component of the PCEE not now addressed in either CAIS or ARTEWG CIFO would be the command language requirements to support the Target and Integration Environment interactions as described in this report.

1.2.11   Bare Machine Philosophy:

As used by the SERC Team, this concept is based upon two principal tenets:

- Application software should be developed by professionals who focus on Ada implementations of solution models where independent application problems in the environment require independent solution models and therefore independent application programs. The application software professionals depend upon the underlying systems software for the runtime management of all services and resources which are intended to be sharable among independent application programs, either initially or in the future. In turn, the application programs are responsible for the management of any services and resources which are specifically intended to be non-sharable with other programs.

- Systems software should be based upon Ada implementations which:
  - manage sharable services and resources provided to authorized application programs and command language instructions
  - manage the simultaneous support of both

19

mission and safety requirements particularly in the Target and Integration Environments. Note that if the mission(s) involve multiple, independent application problems, then the systems software should support multiple, independent application programs (i.e., multi-programming) where each may have its own balance of functional versus nonfunctional requirements. However, the systems software must simultaneously sustain a level of mission and safety requirements at each site that can never allow a lower priority or less stringent application program to compromise the mission and safety requirements either assigned to the site by configuration control or that is associated with more demanding application programs with execution requirements dependent upon that site.

The two integrated tenets of this concept led to the term "bare machine philosophy" because both the application software and the systems software are built entirely in Ada (to the maximum extent possible). Furthermore, the same compiler is used for both. As shown in Figure 1-6, the individual application program components that are to be assigned to a site can be compiled and an inventory of their calls for runtime environment services and resources can be made. (Note that these calls come from the compilation of both application source code components and their explicitly imported application library modules.) Then, transparent to the application developers, this inventory of required runtime support is augmented by consulting the configuration object base to determine what additional services and resources may be required at this site. If, for example, "B3 class, multilevel security" and "no single point of systems software failure can cause..." are system level configuration requirements from this site, then this information is used in conjunction with the above inventory to select the runtime support library components that are to be compiled to support the application program components assigned to this site. (Note that such system level configuration requirements can and should be transparent to the application software developers.)

Concluding Discussion: The component models and paradigms of the Ada language, such as multitasking and exception handling within independent application programs, do not map easily or well to the older-if-

20

not-obsolete component models and paradigms of most
current state-of-the-practice operating systems. This
is particulary true for parallel and distributed target
environments. Therefore many compiler producers with
perceived requirements to execute on top of a
conventional operating system have had to devote
incredible talent, energy, and other resources to
produce Ada runtime support that is a compensatory
mechanism for the inadequacies of the underlying OS.
(Attempting to graft silk to a sow's ear comes to
mind.) This practice is not only inefficient resulting
in larger code segments and much slower execution than
necessary, it is also a much greater risk in the long
term for reliability, security, and the much more
demanding aspects of safety. (More will be discussed
about this in Section 1.2.12.) These and other
attributes of the executable code and command language
components can never be any better than the combination
of the:

-   compiler,
-   runtime library, and
-   execution environment

can support. The authors of this report, the SERC
team, and many other researchers of computer systems
and software engineering with Ada believe strongly in
the Bare Machine Philosophy as the best chance for
simultaneously supporting both mission and safety
requirements in large, complex, non-stop, distributed
environments which evolve incrementally over a long
development period and must be continuously sustained
over an even longer life cycle.

1.2.12   Safety: A Clear Lake Model for Integrating Twelve
           Underlying Component Models to Support Computer Systems
           and Software Safety

           Safety: "The probability that a system, including all
           hardware, software, and human-machine subsystems, will
           provide appropriate protection against the effects of
           faults which, if not prevented or handled properly,
           could result in endangering lives, health, the
           environment, or property." (CWM July '87)

           Discussion: The simultaneous support of safety and
                       fulfillment of mission requirements is an
                       "end", not a "means". For the Space
                       Station Program, such component models as:
                       dynamic, multilevel security; tailorable
                       runtime support environments developed in
                       Ada; resource pools; distributed, nested

21

transactions; command language interface; redundancy management; and six others are regarded by the authors as "means" of improving the probability of success of achieving these goals in the target and integration environments of the SSP. These twelve component models are highly interdependent and interactive in their support of mission and safety requirements for a Portable Common Execution Environment (PCEE) as recommended in this report. Readers desiring additional information should consult Appendix B of this report plus the relevant SERC Memo listed in the Bibliography.

2.0 Commonality Perspective of CAIS and PCTE

Both CAIS and PCTE are intended to support the execution of software engineering tools in a host environment. Their goal is to promote the portability of tools from one software engineering facility to another and thereby reduce the redundant effort currently required to develop programming support tool sets. The source of this redundant effort and the specific goal of both CAIS and PCTE is the significant variation in the organization and structure of required system services with which tools must interact. These variations are so extreme that the effort to move a tool from one system to another is a significant barrier. There are enough aspects which require complete revision of the software that often only a low percentage of the code can be used as originally designed.

Thus both PCTE and CAIS aim to define what is typically considered the operating system interface. This interface must be provided in one form or another on each system which is to support the common tools, but the effort to provide this interface is less significant than the redevelopment of the tools. Even if this were not the case, there are significant advantages to the commonality of the interface and tools in terms of the familiarity to the users, and ease of movement among software developers from one development facility to another.

In the process of defining these System Interface Sets, it was recognized that to do so requires an implicit model for information management and conventions on the organization of system resources, e.g.:

- the organization and naming of files and devices,
- how to store and reference file and device characteristics, and
- whether and how to support explicit identification of relationships between files, etc.

In fact, it was uniformly decided that this issue demanded a new perspective beyond what has been traditionally provided. Both PCTE and CAIS adopted an information management approach referred to as Entity Attribute/Relationship Attribute or EA/RA modeling. Developed originally for data base management, EA/RA is a generalized approach to information management which was chosen for its potential in providing management control over complex and evolving systems. It can be used not only to organize and provide control over files, but over other resources, such as devices, network facilities, users and user groups. As discussed in Chapter 1, it can also be used to support stable interfaces, layers, and stable frameworks.

23

It is quite significant that a common approach is being adopted and made a central part of the services provided. Equally significant, although beyond the scope of this discussion, is the approach to and support for the evolution of information management conventions. This holds promise for allowing technology development without outdating existing systems.

In addition to addressing information management, it was necessary to define some significant aspects of program execution in the host environment. Although PCTE and CAIS differ, it is equally a part of their specification to address this issue. It is in this area that PCTE and CAIS fall short of providing a Portable Common Execution Environment that meets the needs of the Space Station Information System. It is their limited focus on tool support and the necessity to remain open to a wide range of commercial systems that dictated their approaches. The consequence is limited application of their process management approaches to the widely distributed, highly reliable applications present in the space station.

2.1 Desired Characteristics of a Common Systems Services Interface Set

Both CAIS and PCTE consider a number of desired characteristics, from a number of different perspectives, in the design of their tools interface sets. One such perspective is the tool writer, i.e., what does the tool writer need, beyond the Ada language, to write portable tools. A different but equally important perspective is that of the project information manager, i.e., what management controls should be provided, for users through the tools, for use by tools, and built into common tool services. The last perspective is that of the system administrator, taking a systems-level perspective of collections of software development facilities and of collections of resources within a development facility. In this case the concerns are those of interaction between systems, the exchange of data, and of coordination of resources, as well as how to allow for distribution and manage such distribution. These different perspectives will be considered in turn, prior to the discussion of the contents of the two interface sets.

24

2.1.1 The Tool Writer's Perspective:

The writer of a tool is the first source of requirements, and perhaps the easiest to satisfy. The requirements are in the form of services and resources necessary for the tool writer to accomplish the task at hand and provide the user with a "friendly" interface. These services are by now quite familiar. UNIX is an excellent example of focusing on these requirements and providing a flexible and powerful set of services. (Only the introduction of high-resolution, bit-mapped screens, pointing devices for input, and the surrounding windows-oriented user-interface demand new attention). For the following discussion the required services are partitioned into two areas of concern: process control and external interactions.

Process control is a concern for two reasons. First, the tool writer needs to know about the execution context of a program, what are the predefined attributes and how these might be changed. Both PCTE and CAIS equate unit process execution, with associated scheduling and resource control, with Ada program execution. Thus, the model for process control provides the framework for answering the context of program execution.

Of further interest, however, is how to start up other processes and to coordinate their execution; how to communicate with other processes if necessary; and how to synchronize with the execution of another process. These services are needed principally to support multi-user systems and to support background activities and contexts for single users. They are used more by the command language processor or multi-user executive than the individual tools, but still must be a part of the common system services.

Neither PCTE nor CAIS are looking at process control facilities to provide parallel execution on multi-processor configurations for the purpose of improving the performance of tools. In this and other respects their process control facilities are very traditional and limited in their ability to support complex systems of cooperating processes.

The second area of concern is that of external interactions. A tool needs to know about its coordination with the user, in terms of inputs, messages, options in tool execution, user break-in possibilities, logs, and listings. A similar concern is its coordination with the data base (files) of design information, source, and object code. A common interface set must provide a common set of services and resources to support these interactions.

Altogether, the tools writer's needs are generally met by any standard operating system. The requirements are not new and can be easily met in a basic form. The nature of their basic

definition and interaction, however, depends on the other requirements of a system interface set covered in the following sections.

2.1.2 Information Management Concerns:

Most significant in altering the common conception of system interface sets is a recent focus on providing more powerful information management facilities. The typical concerns here are those of configuration control, ensuring data base integrity, information security, and access control. As a preface to this discussion, it is important to note that the whole concept of information management has taken on new meaning with the introduction of the Entity Attribute/Relationship Attribute (EA/RA) model. Now information management is frequently conceived of and understood in terms of entities, relationships and attributes. Thus while these concerns have been recognized as significant in the past, there is now a common mechanism to support their definition and raise the common perception of the need for their support. The draft ISO standard IRDS (Information Resource Dictionary System) provides a standard way of describing EA/RA models through schemas, subschemas, and dictionaries that can be shared among heterogeneous computing systems.

Configuration control of systems is the management of strict naming of components, baselining of stable versions, and control of updates. The requirements on a system interface set are to support these management activities and to provide the means to enforce management conventions. The EA/RA model offers a way to explicitly identify and trace relationships between components and to identify the distinguishing characteristics of components and their relationships. These have been recognized as significant, if not invaluable to configuration control. In comparison with existing file systems, in an EA/RA data base more semantic information about the files (i.e., their relationships and attributes) is explicitly available. In addition, the type and extent of information stored would be tailorable on a per-project basis.

Integrity control is the assurance of a stable and coordinated data base despite procedural (human and software) and hardware failures. With project data bases growing in size and complexity, this is becoming a significant issue. It is considered a significant weakness of UNIX. The techniques used consist of ensuring sufficient information has been saved and marked before an operation has started to enable returning to that information should the operation fail. For a common interface set the requirement is for a very localized version of this, commonly identified as transaction services. Various levels of complexity can be supported up to the level of distributed, hierarchically-nested transactions. PCTE provides

26

this full extent, while CAIS-A appears to be aiming at only basic transaction services.

Information security and access control are further requirements on defining a common interface set. There are two perspectives which may be imposed depending on the application of the tool set. The most common is to provide for group and individual ownership and access rights to files, thus providing partitions within a common data base. The principle purpose of such partitions is to protect against accidental destruction and casual search and to provide ownership responsibility of files which must be maintained, archived, and limited due to storage and other restrictions. The second perspective consists of formal security, logical if not physical isolation of information, and protection from active attempts at unauthorized access. This latter perspective is only now being considered feasible within a typical multi-user computer system.

2.1.3 System Administrator's Perspective:

The final source of design requirements is the system administrator. There are two major concerns which the system administrator recognizes: interaction between systems and distribution of resources within each system. In each case there is a wide range of issues which have been raised, with a corresponding range of complexity in addressing them in the definition of the system interface set. Unlike the tool writer's perspective, which is well understood, and even the manager's perspective, which is more recent and less understood but with a certain current consensus, the system administrator's perspective is not fully appreciated and the related issues are without full consensus.

The principle issues in providing for interaction between systems are of exchanging files and exchanging project data bases between systems which may not be compatible in terms of their host computer, operating system or other factors. A related concern is for supporting backup and archival of data and ensuring that a future system upgrade or replacement will not invalidate the data which has been archived. A typical resolution involves at least a common external form for data storage and for representing the relationships and attributes within a project data base.

The CAIS-A project team has clarified that support in this area is a middle ground between the application level and media levels. CAIS-A will not attempt to define application standards for representing application data systems, nor will it require that an unknowing program be able to open any given file and interpret its contents. At the other extreme, CAIS-A will not define media standards for storage devices. The support left in

27

the middle is the definition of a common external form, which all CAIS implementations will recognize, for the "meta-data" (relationships and attributes of the data base) and for the basic representation of data within a file.

To explain this situation further, the issues here are closely related to the issues of interoperability between tools whether they are co-located, concurrently executing, or even different generations of the same tool. Interoperability refers to the ability to exchange information, where information is not only the data involved, but also the proper interpretation of that data. The difficulty is that information is interpreted at many levels in its lifetime. Useful exchange of information requires standards of representation at all levels. Consider the following:

- Within the program, the design dictates how the information which is to be worked upon will be reflected as a collection of certain data declarations. These alternatives in representation are at the highest level and are dependent on the application domain. As an example, the definition of an intermediate representation for an Ada program is an important design characteristic of Ada compilers which differs substantially in current implementations.

- In its representation as data objects in the program, the information is interpreted by the compiler into a particular representation in the target hardware which is to execute the program.

- Beyond this, for information exchange, it must be written out using the standard Ada I/O services. One noted issue in interoperability is that current standards for external representations do not provide any information about the data structures which have been written out (non-self-descriptive file formats). The information cannot be retrieved without implicit knowledge of these data structures and the application which created them.

- Even without consideration for a self-descriptive file format, at this point, the compiler, operating system and physical media all define individual aspects of representation. These individualistic aspects generally prevent any other combination of compiler, operating system, and device from accessing the information, even when given the original program.

It is recognized that true exchange of information requires a staggering amount of standardization, and that a common system interface set cannot solve all of the problems. The system interface set should, however, define the middle ground between the application standards and implementation details, such that

28

tools developed at different sites and at different times will agree on the writing and reading of the basic data structures in the Ada language.

The other aspect of the system administrator's concerns is that of managing a distributed set of resources within one system. At one extreme, this concern has simply been dismissed as being entirely an implementation concern. That is, there is no need to consider these problems because distribution can be provided fully transparent to the tool writer; there is no impact on the definition of the interface, only its implementation. The alternative is to recognize the possibility of distributed resources and provide some support to the implementation of the interface set.

From this latter perspective, the most important feature is the incorporation of standard attributes and relationships which can be used to provide an information management model of the resource distribution. This can be used to allow tracking of resources (e.g., what data is where, what processes are where, etc.), network administration and fault recovery. Similarly, attributes and relationships can support distributed data management with replication as needed and coordinated updates.

2.2   CAIS and PCTE: The Definitions of System Interface Sets

The following sections will discuss in greater detail the contents of CAIS and PCTE. For our purposes, a system interface set will be considered to consist of three parts. We first look at the foundation, the model for information management. In both cases this is EA/RA. The bulk of a system interface set definition will be actual services provided, which are discussed next. The services are organized in terms of EA/RA management, process control, and external interaction services. The next section considers the conventions and predefined aspects which all implementations must provide. The final consideration is of general design issues in defining a common interface set in Ada. These last considerations are primarily drawn from the effort to develop an Ada specification for PCTE.

2.2.1 The Model for Information Management:

This is the area of the most technical innovation for system services, and one for which PCTE and CAIS have chosen the same approach. The basic system supports the establishment of a logical network of "entities" connected by "relationships". The most fundamental application of the model for information management is as a replacement for the traditional file system. This serves as a good example for comparing differences and similarities with an existing model.

29

Traditionally, file systems have been organized as a simple "flat" partitioning of the available file storage space, or as a more flexible hierarchical system. In the case of a hierarchical system, files are grouped into directories, similar to the partitions in a flat system, but which may also contain sub-directories. Sub-directories contain other files as well as other sub-directories and so on in a hierarchical fashion. Note that because of the hierarchical organization, access to a file is no longer a matter of specifying the one partition which contains the file. In this case, a "root-directory" (or top-level directory) as well as the nested set of sub-directories must be specified. This is known as a file's pathname.

In an EA/RA data base, a file is one type of entity. Relationships are used to group and connect them. A directory would be a different type of entity used solely to connect several related files. In this system, a file can have several relationships to other files and have several relationships connecting to it. The system is no longer a simple hierarchy, but a highly interconnected nest of files and relationships (see Figure 2-1). Each such nest would have a unique starting point, corresponding to the root directory. For the EA/RA data base, the root would have specific attributes associated with a particular user or group using the file-system.

KEY
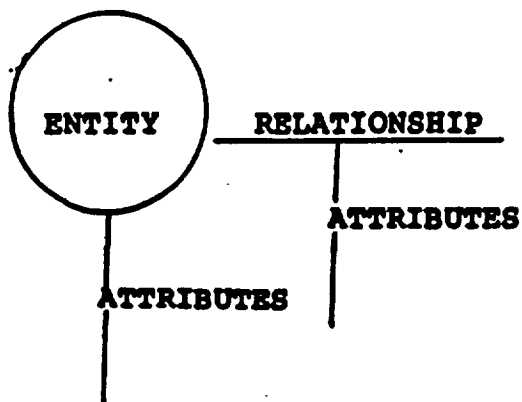
ENTITIES


```
   _____
  /        \      "STRUCTURAL NODE"
 <          >     SIMILAR TO DIRECTORY
  _____/
```

ENTITY   RELATIONSHIP

ATTRIBUTES

ATTRIBUTES

```
 _____
(      )           FILE NODE
 ‾‾‾‾‾‾
```

```
                              <  SYSTEM  >
                  USER              |
                                    |
                  < JONES >
                   /        \ DOT
              DOT /          |
                 /           | DOT
         <PROJECT         <PROJECT
            A  >              C  >
          / |  \
     DOT /  |DOT \ DOT
        /   |     \
     (   ) (    ) (    )
   SOURCE  OBJECT  TEST
   FILE    MODULE  EXECUTABLE
```
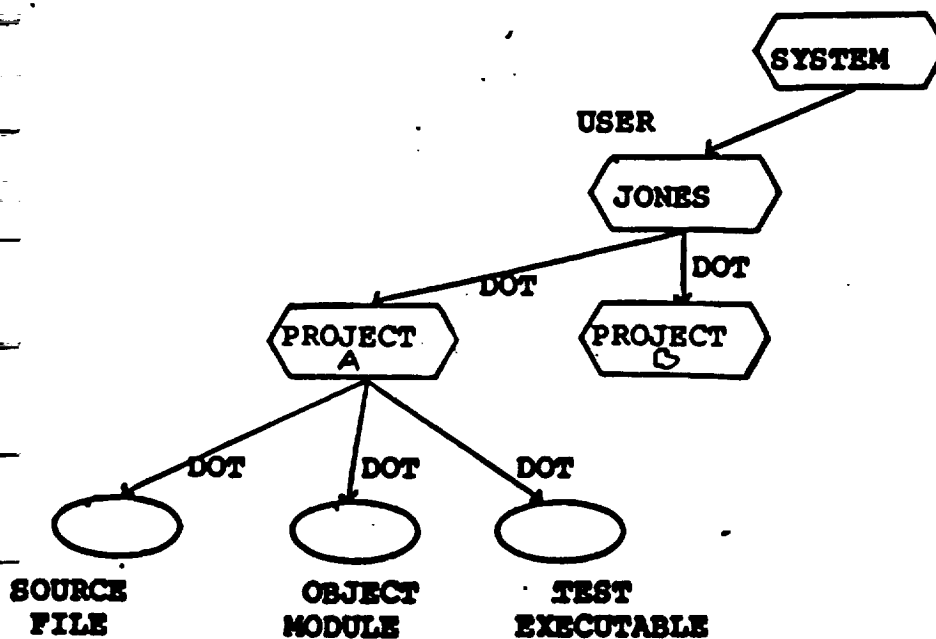
USER AND DOT ARE PREDEFINED RELATIONSHIPS.
DOT SIMPLY CONNECTS ONE ENTITY WITH A "PARENT" ENTITY.
AN ENTITY MAY HAVE SEVERAL DOT RELATIONSHIPS CONNECTED FROM IT,
BUT ONLY ONE CONNECTED TO IT, IN TYPICAL HIERARCHICAL FORM.

FIGURE 2-1a
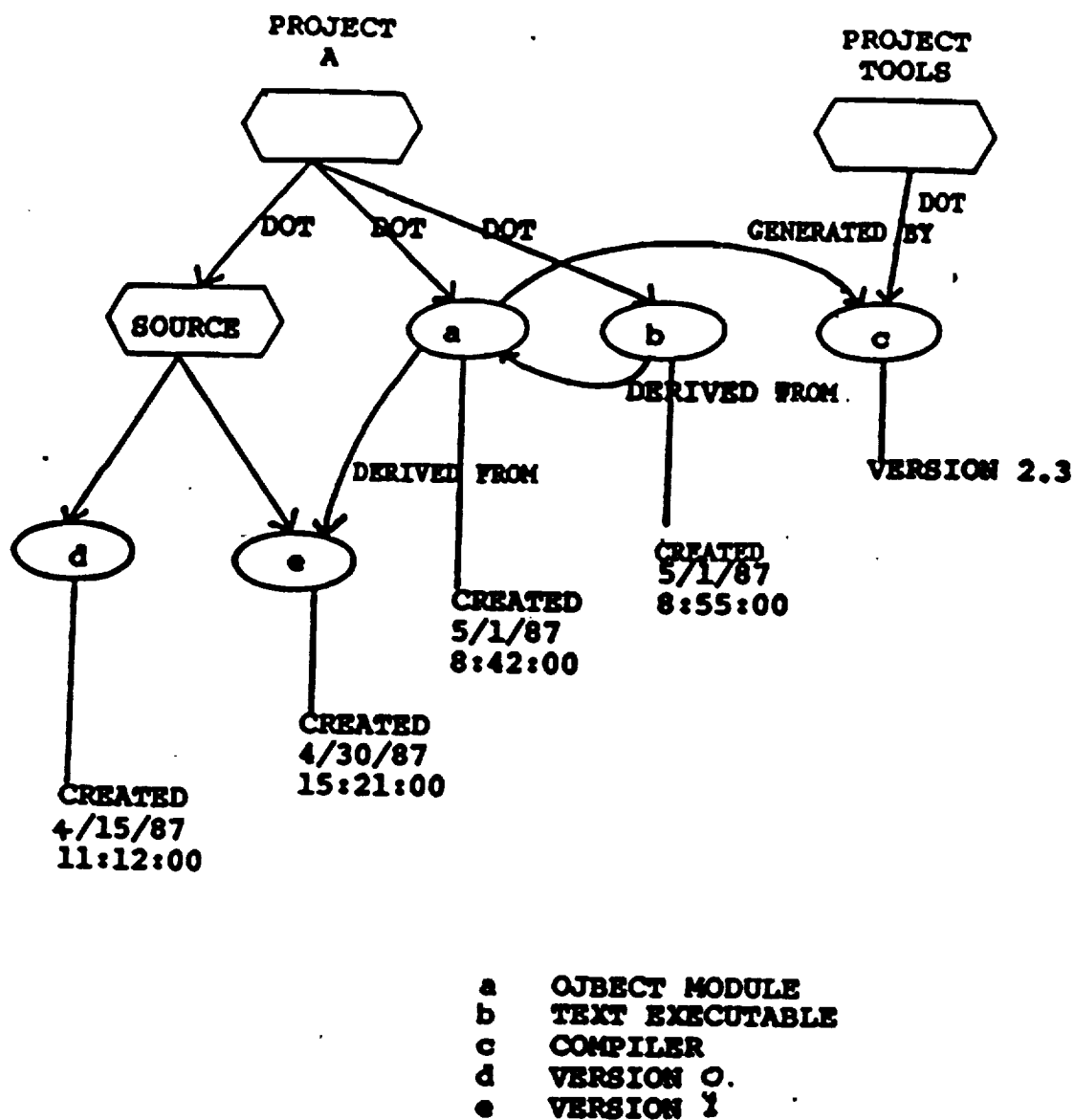EXAMPLE OF EA/RA CONCEPTS

FIGURE 2-1b
EXAMPLE OF EA/RA CONCEPTS

While it may appear to lead to enormous complexity, properly managed, the file system will reflect the logical relationships which exist among the files, and these relationships will have been made explicit and visible. The system does not change the requirements for accessing a file. As for the hierarchical system, a "root" starting point, and pathname for traversing the various relationships are all that is required. If appropriate or necessary, a hierarchical system can be directly represented as a subset of the more general EA/RA capabilities.

Both PCTE and CAIS-A will provide an enhanced capability in which the various entities and relationships are strictly classified (also called strongly typed, but this differs somewhat from the strong typing of programming languages). For these systems, the different entity types will be distinct, ensuring that only proper operations are performed. The different types of entities which may be managed by the system are open to user extension. For example, sets of objects and their relationships along with their respective attributes can be assigned a common attribute suffix to form a Stable Framework (SF) as discussed in Chapter 1. Now the SF can be addressed as a strongly typed object and, in turn, can exchange messages with other SF's just as if they were single instances of a strongly typed object. Both CAIS-A and PCTE will allow for system evolution as new entity types and relations are derived from existing ones and designed to co-exist.

CAIS and PCTE diverge in their application of the EA/RA modeling. PCTE provides these services as a replacement for the typical UNIX file system, with entity typing used as a way to distinguish between different types of files in the system. CAIS takes the modeling further, applying it to process management and access control, and in the case of CAIS-A, to peripherals and distributed resources within a system.

2.2.2 The Set of Services and Resources:

With the EA/RA model as a basis for information management, a number of common operations, which would be duplicated for each type of entity in previous systems, can be consolidated under the heading of EA/RA management. These services provide for the creation and deletion of entities (called nodes in CAIS and objects in PCTE), as well as connecting the entities into the system through relationships. For each entity type and relationship type there are a number of required and optional attributes which may be specified at the time of entity creation and connection respectively. While PCTE and CAIS-A go about their EA/RA management in different ways, and PCTE has a more limited domain of entity types, the basic capabilities are similar between the two.

32

A second group of services are provided in the area of process control. Here the standard services of process creation and termination, including abort, suspend and resume are provided. Each system has mechanisms for returning results and for waiting for a process to complete and return results. CAIS-A has noted the need for separating process creation from actual startup. Interprocess communication is provided for through message queues. PCTE provides messageless non-waiting signals. These services are not significantly different from each other, each providing a basic multi-process capability. This multi-processing capability is traditional, however, and as was indicated earlier, is not appropriate for complex multi-processor and multi-programming applications. In particular, to represent an Ada program with one process control block, regardless of how many separate threads-of-control have been spawned within the program, is regarded by this research team as a very damaging deficiency of both CAIS and PCTE. (The reader should remember that Ada provides dynamic support for firewalling properly designed tasks. Therefore, the failure of one task need not cause the program to abort or other tasks to be corrupted. Designed properly, the failed task can be replaced or a safe work-around can be effected in a non-stop execution environment. However by not representing each thread-of-control within the program with its own process control block in an accountable, hierarchical relationship to the main program's process control block, such issues as: interactive symbolic debugging, reconfiguration, performance monitoring, and interactive safety support provided by the integration environment to the target environment become far too difficult to deal with safely.)

The final area of system services is broadly classified as external interactions. External in this case refers to being external to the program (tool) which is executing. We have already discussed interaction with another process. The other two aspects are data base (file system) interactions and user interaction.

Much of data base interaction is now subsumed by EA/RA management, but there are additional requirements to read and write the contents of the files. These are handled with traditional file I/O services. The differences between CAIS and PCTE are directly tied to their origins. CAIS is intended to serve principally the Ada language, and so its I/O capabilities are exactly those defined in the Ada language reference manual (LRM). PCTE, on the other hand, is explicitly derived from UNIX, and thus borrows the UNIX I/O services. The Ada specification for PCTE which is being developed will replace the UNIX services with Ada's, with slight modifications where necessary for compatibility. In any case, there is little surprising about these capabilities.

PCTE has taken, and CAIS-A intends to take, a significant

33

step beyond traditional system support in providing transaction services. Such services are necessary to ensure integrity of the data base with multiple and distributed processing. PCTE provides the full capability of distributed nested transactions, while CAIS-A currently is looking at only single-level transactions. The reader should note that distributed nested transactions is the most powerful and parsimonious mechanism known today for supporting software fault tolerance in parallel and distributed systems. Single-level transactions, by contrast, do not allow the full exploitation of parallelism and distribution in the system and do not support fault tolerant, non-stop operation.

User interaction is an area of considerable disparity between CAIS and PCTE, and also between CAIS and recommendations for CAIS-A. CAIS as it currently stands defines an extensive set of services for traditional devices, those which are similar to teletypes and line printers (called scrolling devices), those which allow direct cursor positioning as on a CRT screen (named page devices) and those which can be partitioned into fields, some for display text only and some for data entry (named form devices).

PCTE draws upon its focused charter to address networks of high-resolution, high-capability workstations. It establishes an elaborate model of screen and input device servers (processes) with which an active tool (separate process) may interact. Services are provided to establish interactive windows on the screen, and provide window management, mouse control and keyboard input support similar to that found on the Macintosh and other systems.

CAIS-A is looking to provide similarly expansive user interface services, with an eye towards what will be expected of future software engineering workstations and any suitable existing standards. Graphics standards which would support the drawing of software engineering diagrams and standards for windows, mouse and keyboard input are being considered.

2.2.3 Conventions for Processes, Files, Relationships and Attributes

In this organization for defining system interface sets we have covered the model for information management and the nature of services which are provided. Still needed for a useful system are standards for predefined node types, relationships and attributes upon which all tools can depend across all implementations from one system to the next.

The most common use for standards definitions is in the area of configuration control of the file system. For entities which

34

are files (of type File or derived from type File) required attributes and relationships might be creation date and time, tool which called for creation and input file which the tool used to derive the contents of the new file. This is only a small sample of standards and conventions which might be needed.

Much in these standards is embodied, in an EA/RA data base, in the typing of entities and relationships. Both PCTE and CAIS define an essential kernel of entity and relationship typing, but fully support user extension. As will be discussed shortly, CAIS-A has the additional capability for migrating such information directly between systems, along with the contents of the data base.

A second aspect of the EA/RA data base which must be standardized and, in some cases, built into the structure and implementation of the system services, is access control. In this area CAIS-A has gone significantly further than PCTE. PCTE provides only the first form of access discussed earlier, that of informal partitioning of the data base for ownership, and protection from accidental (unintended) or casual but undesired access.

CAIS-A attempts to provide the attributes and relationships necessary for a formal security system. In this case, for accreditation as formally secure, the implementation would have to be centered around a secure kernel which implemented the only access to the data base. Much of this accreditation would thus be based on the nature of the implementation. What CAIS-A provides, however, is the conventions and standards upon which such a system could be built.

A third area for conventions and standards is in support for distribution. PCTE, in recognition of its intended configuration of a Local Area Network (LAN) of workstations, provides some support in its definition for such an implementation. Its approach to distribution is basically to mandate the transparency of distribution to the user, plus augment its services and standards to accommodate distribution requirements. Note that the PCTE network is uniform and configurable, with no hierarchy of rights or responsibilities.

PCTE adds to its services and standards in the following ways:

- All identification of objects, processes, users, message queues, etc. is system wide and recognized throughout the network without reference to its location.
- Process start-up may optionally specify a location for execution, or may have default parameters which require start-up on certain workstations.

35

- The data base is maintained in separate volumes which are mounted on the various workstations in the system. There are no restrictions for relationships between objects on different systems.
- Volumes and workstations are modeled as objects in the data base with such modeling used to allow identification and management of the resources.
- There are a certain number of services which directly support network management.
- A standard model with kernel support services is provided for replication of data with coordinated updates. This is provided mostly to standardize implementations, i.e., it is not generally visible to the typical interface set user. It provides services, however, which aim to protect the system from resource unavailability and network failures.

CAIS-A in meeting its intentions to support distribution will necessarily define similar support for identification, resource control and management, although there is no indication that support for replication would be standardized as part of the interface set (an omission that will retard standard support for general redundancy management, unfortunately.) CAIS-A will, however, make visible a standard for information interchange (the common external form) which could apply to communications between processors in a distributed implementation.

The final area for conventions and standards is in support for system interaction. PCTE is essentially lacking in this area, while CAIS-A's approach (as discussed earlier) is limited to the definition of a common external form.

2.2.4 Design Issues in Defining a Common Interface Set in Ada

The final point of discussion on System Interface Set specifications is that of some general Ada style and design guidelines. This discussion is drawn from the recent effort to define an Ada specification for PCTE (PCTE-Ada) based upon the original specification written for C and UNIX (see PCTE Ada Conceptual Design Document). The design issues which were encountered can be grouped into issues in the use of the following Ada capabilities:

- private and limited private types in package definitions, handles, naming and modularity of package definitions, exceptions (a proper mechanism for error reporting/handling), generics, and tasking.

Some similarity can be found in how CAIS-A and the Ada binding of PCTE will be defined. CAIS-A will revise its error handling mechanism to support just error reporting without the raising of exceptions. While it is of academic interest to debate the appropriate Ada style, compatibility with the C version of PCTE is forcing this approach for PCTE-Ada. Also, a similar approach to task execution within processes is being adopted.

3.0 Common Environment Architecture

The collection of computing environments to be managed for future software systems will require capabilities to support heterogeneity, physical distribution, cooperative autonomy, safety and reliability in the execution environment. Heterogeneity is often unavoidable due to growth and evolution over the long operational life of a system. Physical distribution and cooperative autonomy will be necessary to model the structure of the disparate organizations involved in software system development, integration and operation. Safety and reliability are crucial to support critical systems upon which life and property depend. This section discusses features which are necessary to realize the entire set of dynamic capabilities as well as reports on the existing static host environment models that support a subset of these characteristics.

Three types of software environments, each with different functional and operational requirements, can be identified: host, target and integration. In the host environment, the primary concern is the development and maintenance of software, as well as associated documentation, requirements, specifications, design rationale, etc. The characteristic host environment will be resident on a general purpose operating system. In contrast, the target environment application software and runtime support modules will either reside on a general purpose operating system or a subset of an operating system specifically tailored to the supported application or, as discussed in Chapter 1, reside on a bare machine. The target environment is concerned with deployment and operation of executable code, the preservation of semantic integrity across disparate processors, and the realization of hard real time constraints, possibly in the presence of software, hardware, operational or environmentally induced faults. The functions of the integration environment are: control of the target environment baseline, including management of and promotion to further baselines, and the integration of software applications with hardware and operations.

Throughout the iterative, dynamic evolutionary life cycle of a system, the software (in different forms: multiple versions, different representations etc.) at one time or another, will reside in at least one of, or migrate among, the three environments. A Portable Common Execution Environment (PCEE) model that eases the software environment interaction and migration process among the three types of environments would provide a solution that increases productivity and the support of computer systems and software safety. A consolidated PCEE model can facilitate the management of the life cycle complexity of software systems in a similar manner across all three environments.

A PCEE is sustained at the stable interface set described by the Ada packages selected from the Runtime and Extended Runtime Libraries to be compiled into an appropriate RunTime Support Environment (RTSE) for each processor. The PCEE can be defined to consist of:

- a set of policies for the management of services and resources to be provided to the application program(s),
- the set of management modules to enforce the policies,
- and a set of rules for modification and extension.

Ideally, the PCEE should provide a common set of execution services and resources to application software and the command language that both hides and supports the use of differing instruction set architectures, data bases, data communications systems, bare machine implementations and operating systems without regard to their underlying implementations. We propose this common framework be modelled by a set of standard interfaces, a common object representation, and a supporting conceptual architecture.

3.1 The Model for Information and Process Management

Information is a fundamental resource and processing is a fundamental service of a software system. In order for a system to be most effective, the management scheme must accommodate the capture, organization and retrieval of all relevant information. Lessons learned through the development of successively more sophisticated data base management systems and, more recently, operating systems have shown that an entity/attribute relationship/attribute (EA/RA) model is very powerful for representing system level meta-data. An EA/RA model provides a basis for describing the system in a manner which can be uniform across host, target and integration environments. EA/RA supports a layered approach with stable interface sets and stable frameworks for organizing system resources. It can be used to represent and enforce precise, abstract interface specifications which are independent of the underlying manner in which they are implemented. The IRDS draft standard for representing instances of EA/RA models can be used to describe a formal semantic model for PCEE information and process management. IRDS provides a vendor independent, standard method for representing EA/RA models across heterogeneous computers. An important feature of the IRDS is the extensibility of the model.

3.2 Conceptual Architecture for the PCEE

The framework of the PCEE must be flexible enough to be represented in a number of ways in order to accommodate tailoring and extensibility in a large number of implementations. The PCEE should also support the differing operational requirements of the three environments. In the host environment, software is

39

typically supported by the services of a general purpose operating system. The target environment, in comparison, often cannot tolerate the overhead of a general purpose operating system. The PCEE can be considered a "virtual environment" which resides on a RunTime System (RTS) (which is supported by the Kernel). The Kernel "hides" features which are machine-dependent. The PCEE should "hide" underlying system software implementations ranging from a tailorable bare machine implementation to a tailored operating system to a general purpose operating system which co-exists with combinations of the first two.

A conceptual architecture for a PCEE is depicted in Figure 1-6. Reconfigurability and performance tuning capabilities are supported by the architecture for all three environments. The architecture is based on a combination of layered and virtual machine approaches: the lowest layer is the minimum functional subset that isolate the hardware dependency features. This minimal subset that supports the basic runtime support is called the kernel. Additional runtime environment services will be implemented via library modules and extended kernel functions. These include runtime services such as virtual memory services, file management, and atomic transactions.

Based upon the needs of the application code and the configuration requirements of the intended execution site, the runtime system is tailored by selections from a set of standard library routines. These modules provide a layered architecture on top of a virtual machine approach.

Subsetability and reconfigurability are accomplished by first identifying the lowest layer as the minimum functional subset that isolates the hardware dependency features. This minimal subset forms the foundation of an extended machine for the kernel that supports the basic Ada runtime support functions. Typically, this may include functions such as I/O services, memory management, process management, and interrupt handling. Additional runtime environment services will be implemented as incremental library extensions to the minimum kernel functions. The extensions will use the kernel services only, to ensure portability. These extended runtime environment services may include services such as the Ada tasking, file management, the dynamic memory management, and virtual memory services. Together the kernel and the runtime support library routines for the twelve component models discussed in Chapter 1 can be grouped into safety kernels underneath a PCEE to maximize protection for life and property.

To provide for selectability and configurability of services, the interdependency of the runtime library elements will be identified and minimized through the design process. The set of special service functions, such as fault management, configuration management, and system security can be provided as

40

independent packages. These packages should be designed to interface to the kernel functions only; there should be no direct interaction among the packages. All RTSE library routines execute as privileged code.

To support system tunability, certain functionalities can be provided with multiple implementations, each with different operational performance. For example, to support certain real-time control processing, a special set of scheduling algorithms and interrupt handling mechanisms will be required.

3.3 Additional Services of the PCEE

PCEE services required to support the collection of distributed heterogeneous computing environments, as well as the rationale behind the selection of particular services are discussed in this section.

3.3.1 Security

In any significant software system, security must be provided. A distributed system which supports a diverse user group is particulary vulnerable to problems which result from inappropriate access to information, processes, and devices. In the minimal case, protection must be provided against inadvertent access resulting from programmer error. Additionally, errors resulting from the incorrect functioning of programs should be prevented from causing catastrophic results by security built into the PCEE. (That is, even "error free" programs cannot always execute without error in a distributed and/or non-stop environment.) At the other extreme, limited access to life, property and other mission-dependent critical functions must be provided against the hacker, the terrorist, and the disgruntled employee. Secured processing may also be required. Context sensitivity may also be at issue: an access operation that would be allowed in a "secure" area might be prohibited depending on the location of the terminal being used to effect the transaction. Security should be provided according to the Trusted Computer System (TCS), to at least the Multi-level security (MLS) class B3.

From the discussion in Chapter 1, the reader is reminded that, at least for the SSP, security is a "means-to-an-end", whereas, safety is one of the "ends" which coexists with fulfillment of mission requirements. Specifically, security is one of at least 12 component models required as "means" to support both of these "ends". Obviously any compromise of the required component models underneath the PCEE (such as the dynamic, multilevel security model) can endanger both safety and mission. It is precisely for this reason that the PCEE interface set must be specified in Ada rather than in any untyped langauge such as C and the UNIX System V interfaces. The terms "secure

UNIX" and "safe UNIX" area temporal oxymorons unless:
1) all features of UNIX are hidden beneath the virtual interface set of a strongly typed language such as Ada and
2) application programmers are denied access to assemblers and other untyped languages.

The point is that the C compiler is an undefendable trap door to the underlying features of UNIX and it can be used to defeat the best efforts to support security, reliability, fault tolerance, and safety. Even if such "secure" or --far more difficult to develop -- "safe" UNIX implementations can be demonstrated at acceptance test/delivery time (and some "secure" ones have), mere mortals are unlikely to be able to sustain them over the life cycle without lots of Divine intervention. This is particulary true in distributed applications!

3.3.2 Cooperative Autonomy Among PCEEs

Not all of the potential interactions of a long-lived system can be determined during its initial design. In order to provide the most extensible basis for PCEEs, the capability for coordinating control among PCEEs that may need to interact should be provided. Conventions for determining the result or precedence when coordination is required should be established within the definition of the PCEE.

3.3.3 Process and Information Migration

Executing processes and information require the capability to migrate to different locations to accommodate load balancing, optimal access and/or fault tolerance. Load balancing is concerned with optimizing and/or ensuring processing throughout a system. Secondly, in order to expedite the transfer of frequently accessed data, movement of data (or procedures) may be necessary. A third, and potentially most important, reason for allowing processes to migrate to other processors is to provide critical functions in a fault-tolerant manner to support safety. In the event that a system recognizes the imminent failure of a processing resource, the most important functions and information would have the capability to move or be moved to other sites.

3.3.4 Heterogeneous Processors

During the existence of a long-lived system, the incorporation of new hardware technology may be beneficial to the system. New technology may be necessary to accommodate evolving needs or changing objectives. In order to accommodate new hardware, a PCEE may require support for a heterogeneous set of processors. To accomplish transactions between these differing processors, a common external format for representing data and data structures may be necessary (see the discussion in Chapter

2).   Common formats provide the mechanism for information exchange between machines whose underlying data representations are not compatible.

3.3.5 Communication

Communication, whether the underlying processors are similar or different, must be managed in a manner that supports remote communications, asynchronous signals and the Ada rendezvous. Remote communications are best provided through the use of the Remote Procedure Call (RPC) which is necessary when a called subprogram resides on a remote processor. The need for asynchronous signals arises when a condition needs to be communicated, without the signalling processor relinquishing or slowing the execution of its thread of control. For both safe and --less demanding-- fault tolerant inter-task communication of application information or control synchronization, the concept of the Ada rendezvous is needed. Remote procedure calls must adhere to the semantics of the Ada language (i.e. behave as if the procedures were resident on a single processor). (see Rogers and McKay, 1986). Asynchronous signals, which do not wait for a response, provide communication services while allowing the signalling processor to continue its work. Additionally, the PCEE should support the concept of the Ada rendezvous across the network. The rendezvous provides a semantic model for two communicating autonomous threads of control with defined exception conditions in any of the three: calling task, called task, and communications link between the tasks. Beyond the semantics of communications, a model for implementing the actual transaction is provided by the International Standards Organization/Open Systems Interconnection (ISO/OSI) model. Within the model there are seven layers. Each layer provides a different portion of the required service, so that the amount of overhead incurred can be tailored depending on the complexity of the transmission. The ISO/OSI model is an emerging standard which will provide a uniform representation for asynchronous and RPC communications. However, for the SSP, the OSI model should be extended to include the Ada rendezvous semantics to maximize support for safety in a fault tolerant environment with distributed non-stop components.

3.3.6 Transparency of Distribution

Distribution of logical entities within the PCEE should be addressed at the most general level. Distribution across Remote Area Networks (RAN) of integrated Local Area Networks (LAN) requires solutions to the problems of unreliable communications, determination of an appropriate unit of distribution and unique naming (across networked interacting processors). The model should incorporate transparency of location, replication, parallelism and fault-tolerance and a "sufficiently precise" granularity of control. The amount of transparency provided

43

determines the amount of overhead incurred by the system. Ultimately, it also determines the safety and extensibility of the systems life cycle.

3.3.6.1 Unreliable Communications

In addition to addressing the types of communication required and the manner in which they should be implemented, the problems of dealing with "unreliable" communications links must be addressed. Unreliable communications are especially apparent in remotely connected systems. (This is exacerbated when the links are subject to the laws of orbital mechanics.) In such systems, especially when communication links are not direct hardware connections, problems can occur in establishing and maintaining transmissions. In order to assure "normally" correct communications mechanisms, the services provided by the ISO/OSI model may often be satisfactory. However, fault tolerant operations which the safety of life and property depend upon are far easier to sustain with support for a robust model of cooperating, autonomous threads of control which effect distributed, nested transactions. The authors believe the Ada rendezvous is the best available model for this purpose and strongly recommend its addition to the OSI model.

3.3.6.2 Unit of Distribution

The choice of a source-code level unit of distribution is concerned with source-level language visibility rules, which, in distributed RTE's, is a determining factor in feasibility of implementation. In Ada, the unit of distribution can range from a named entity to a compilation unit. Distribution of logical entities, at least to the task level, will be necessary in order to provide dynamic fault tolerance. The determination of the unit of distribution which would be appropriate for PCEE connections is beyond the scope of this paper, but is critical to the design of the PCEE. The most apparent distribution certainty determined by the SERC team and others is that the unit of software distribution supported by the PCEE must be below the program level and at least to the task level. Otherwise the risk of being unable to sustain the SSP life cycle requirements for both mission and safety support is unacceptably amplified.

3.3.6.3 Unique Identification

Operating systems support objects, such as files, directories, processes, services and I/O devices. A unique name, in a distributed environment, helps to protect the objects in the environment from incorrect manipulation. A method for the unique identification of objects, streams, and transactions, throughout a distributed environment, which complements the choice of unit of distribution, is required. Universally unique identifiers are

necessary for configuration management. They are also required to locate entities that may migrate throughout a vast network or to restore entities that existed previously. They are also extremely useful in the safest and fastest possible replacement of an entity that was just recognized as having failed in a non-stop, fault tolerant environment.

3.3.7 Transaction Management

Transaction management is concerned with the organization of actions into groups which can be monitored by the PCEE. Transactions should be able to be described, manipulated and controlled in much the same way that information within the PCEE is handled. Transactions management requires the capability to effect atomic actions, synchronization, inheritance of capabilities, and control stable storage. Transaction support may also require support across a distributed system, as discussed above. Atomic actions are perceived by the PCEE as one uninterruptable action to either advance to the next stable state or remain at the current one. Physical atomic actions can be combined with "vertical" atomic actions to provide sets called transactions. Nested, distributed transactions are the most parsimonious, efficient and robust of the known ways to support safe, distributed systems. If any component part of the action fails, either a recovery option is successful or the whole action is considered failed. Values associated with the transaction are not updated. Of course, use of expendable quantities, such as firing a rocket, or creation of by-products, such as heat generated by the running of a machine, must be taken into consideration when the initial state is "reinstated". Synchronization provides a mechanism for limiting access to an entity so that multiple actions do not produce indeterminate results. Inheritance allows "parent" processes to assimilate the knowledge of their subordinates once the subordinate terminates.

Transaction management is an integral part of providing a stable baseline. A stable baseline allows each process in the system (to the desired level of granularity) to interact with the system in a manner which provides "fire walls". That is, a process which does not complete successfully cannot impact an unrelated process. The stable baseline is advanced from one stable state to the next. A stable state is a description of the relevant portion of the system data base which can only be changed by correctly completed transactions. Semantic integrity across environments, including type protection, in terms of Ada semantics, is required throughout the persistent data base. That is, values for a transaction are not written until the entire transaction has successfully completed. In this way, each action terminates correctly or has (virtually) no effect on the system.

3.3.8 Granularity of Representation

45

The level at which a software processing entity is considered by the environment to be a "black box" is the lowest level at which a representation is supported. For instance, if the granularity of representation is an Ada program, then any task spawned by that program is not recognized by the environment. This has ramifications if the parent task becomes abnormal, especially in a distributed environment. Since the environment has no knowledge of the subordinate task, it cannot employ a mechanism for removing the task from the environment. If another copy of the program is started and spawns the task again, the continued functioning of the originally spawned task may lead to undesirable results. Granularity of control should be defined for performance measures (design phase), debugging (coding phase), process monitoring (operations phase) and assessment of hardware and software changes (operations/maintenance phase). The granularity of control will dictate the ability to manage multiple command streams. Each body of a distributed subprogram, task or package in a state of execution, while they possess a thread of control should be accessible through PCEE interfaces. Facilities for maintaining the transaction status for transaction management (as discussed above) should also be associated with each subprogram, task or main program. By providing this level of control, a fault-tolerant process is better able to assess its state of execution and determine the need for recovery procedures. Note that the single process control block representation of an entire program (as in UNIX, CAIS or PCTE) can not support these needs. However, a legal extension to CAIS that could support these needs is conceptually straight-forward. Simply have the process control block of the program represent aggregate views of all its tasks while allowing an expanded view of the program showing a control block structure for each thread-of-control's participation in the aggregate. (See Rogers, K., 1986.)

3.3.9 Interoperability

Interoperability in the minimal sense, is concerned with the capability to move source code among processors in the environment without changing the functionality of the software. Interoperability can be extended to include provisions for common external data formats which would allow tools to generate or manipulate information used or created by other tools. If interoperability among tools is provided, an Ada Program Support Environment (APSE) could be created from individual tools. In this way, the APSE could be tailored to the particular application under development. Interoperability of control streams implies that commands should have the same effect (within reasonable limits) anywhere in the environment.

Data, tools and control streams should be interoperable across the distributed network, extending to heterogeneous processors. Requirements for a common external form may be

46

necessary to implement these functions in an autonomous manner. In order to provide interoperability for individual tools, abstract data types (such as for a symbol table) will need to be defined. These types will have to be entered into the data base by the producing tool (for instance, a compiler) for later access by "consuming" tools (e.g., symbolic debugger). EA/RA models represented in IRDS standard form can be enormously useful in supporting such interoperability. (See the Chapter 2 discussion of Common External Form.)

3.3.10 Optimizing the PCEE Goals

The PCEE should balance the life cycle goals of safety, performance, portability, adaptability, cost effectiveness, and stable baselines across the host, target, and integration environments. Reasonable trade-offs among these goals are necessary to manage distributed, non-stop, large and complex systems. The PCEE should provide support for multiprocessing capabilities to provide for concurrent and non-stop operations. It should build on the guidelines established in the ARTEWG Catalog of Interface Features and Options (CIFO) to provide runtime support (see ARTEWG CIFO, July 1986).

3.4 Existing Models and Paradigms

If an existing set of models and paradigms is not exploited in defining the PCEE, a unique set will have to be created. The use of existing models and paradigms provides not only a demonstration of feasibility but also a supply of tools that may have application within the PCEE. Two existing environment models, the CAIS and the PCTE, were studied as possible foundations for the PCEE.

3.4.1 Common APSE Interface Set

The current CAIS, MIL-STD 1838, is an extensible baseline from which a complementary PCEE can be developed. The successor to the CAIS is referred to as CAIS-A (note that the descriptions of the possible contents of CAIS-A are the CAIS contractor's current thinking and are subject to change). The authors believe CAIS-A can and will further the baseline established by the CAIS toward the complementary goals of a PCEE.

Object management within the CAIS is modelled by nodes, and is based on the EA/RA model. Nodes have properties and attributes that can be read using CAIS interfaces. The node model is designed to execute processes in substantially the same way, except for timing. This fulfills the PCEE requirement for an EA/RA-based system. A further extension that would make the CAIS implementation more compatible with the desired features of the PCEE would be to use the IRDS standard for EA/RA.

47

The CAIS provides a virtual host environment system interface between the virtual operating system (represented by the Kernel APSE) and the environment tools; therefore, it can be implemented on either an existing general purpose operating system or a tailored operating system. In this manner, the CAIS provides a "virtual environment" which could be extended to support all three environments that comprise the PCEE.

Security and performance features will be enhanced (in CAIS-A) in order to provide at least Trusted Computer System class B3 multi-level security (MLS). (Formal verification is required for a higher classification.) The incorporation of this class of security meets the currently perceived need for dynamic security within the PCEE. Implementation of security facilities in a modular manner should allow the PCEE to utilize only the necessary portions of the CAIS-A implementation and only the necessary portions of the security class currently imposed.

CAIS-A will also address the facilities that will be required to coordinate actions among cooperating CAISes. The issues that are addressed by CAIS-A will provide at least a basis for determining the interactions among cooperating yet autonomous PCEE implementations.

In addition, CAIS-A will address interfaces that exist among multiple CAIS implementations (across RAN's and LAN's). One aspect of the network implementation will be interfaces which will allow processes to migrate to other processors. CAIS-A will allow support for heterogeneous processors. Some support will be given to a common external data format, to allow communication of data and information between differing processors. Unique names for objects will also be addressed, as will some support for at least single-level atomic transactions.

Within the PCEE definition, support for RAN's of LAN's will be required. The underlying assumptions for modeling RAN's can be used as a basis for PCEE LANs. The basis can be extended to include those assumptions to account for the distinctive features of LAN nodes. However, the reverse --i.e., scaling-up-- is not true. As an example, SQL's single level transactions are believed to be inadequate for supporting safety during RAN operations. Furthermore the life cycle cost effectiveness and performance of single level transactions is believed to be significantly worse than for PCEE support of distributed, nested transactions.

Neither the current CAIS nor the plans for CAIS-A include provision for the types of communication that are necessary for a PCEE. The CAIS also does not provide support for all three of the environments of the PCEE. The current goal of both CAIS and CAIS-A is to provide interfaces for the host environment. Additionally, support for a unit of distribution below the Ada

program level, interoperability among programs, and transaction management would be necessary extensions to CAIS-A, in order for it to provide the functionality needed within the PCEE.

3.4.2 Portable Common Tool Environment (PCTE)

Selected features of the PCTE should also be incorporated into the PCEE baseline. Although there are some features which are common between the PCTE and the CAIS, there are also differences. The PCTE provides some features which go beyond CAIS and CAIS-A toward providing facilities necessary for the PCEE. These PCTE features can be considered a further step toward a conceptual and physical framework for integrating the tools and rules for a PCEE. This framework is supported by the PCTE Ada Specification.

In the PCTE, the first four layers of the seven layer ISO/OSI model are implemented to support distribution at the LAN level. The lack of support for the upper three layers is a most unfortunate shortcoming of the current PCTE. LAN support also includes the concept of agents, servers and clients which is a significant strength. Messages and message queues allowing executing programs to exchange information directly are also provided by the PCTE interface specifications. The existing implementations of PCTE can be evaluated as a starting point for the PCEE implementation of RAN's of integrated LAN's.

The PCTE provides support for both distributed nested transactions and the implementation of replication facilities for entities in distributed environments. It provides mechanisms to synchronize data access. These features will be necessary within the PCEE, to provide transactions and redundancy management, in order to provide a stable baseline.

Example PCEE features needed for Space Station which the PCTE does not support include security, automatic information and process migration, granularity below the Ada program level, tolerance of unreliable communications and unique naming. See Figure 3-1 for a comparison of certain PCEE, CAIS, CAIS-A, PCTE features.

49

Feature	PCEE	CAIS	CAIS-A	PCTE
Status	Definition Stage	MIL-STD	In Progress	Completed
Validation Suite	Required	In Progress	In Progress	Completed (based on XPC)
Basis	ARTEWG CIFO and Clear Lake Model	unique	CAIS	UNIX SVID
Representation	Object	Node	Node	Object
Information Management	Extensible EA/RA (based on IRDS)	unique EA/RA	unique EA/RA	unique EA/RA
Kernel	"bare machine", operating system	"bare machine", operating system	"bare machine", operating system	operating system
Security	Full TCS "Puce Book"	minimal	TCS B3 class MLS	minimal
Cooperating Environments	required	not supported	supported	LAN only
Location	Migratable	Fixed	Migratable	LAN migratable
Processor Types	Heterogeneous	Homogeneous	Heterogeneous	Homogeneous
Common External Data Format	required	not supported	some support	minimal support
Communications Implementation	Full OSI	NA	TBD	Four layers of OSI
Distribution	RANs of Integrated LANs	single site	Some RAN and LAN support	LAN
Unique Names	Objects, Processes, Transactions, Relationships, and Attributes	Nodes, Relationships, and Attributes	Nodes, Relationships, and Attirbutes	Objects, Relationships, and Attributes
Transaction Management	Distributed Nested	NA	Single Level	Distributed Nested
Data Access	Synchronized	NA	NA	Synchronized
Stable Storage	required	NA	NA	NA
Granularity of Representation	Each thread of control for each program	program	program	program
Interoperability	data, tools, control	data	data	data
Goals	portability, performance stable baseline & safety across all environments	portability, performance	portability, performance	portability, performance, stable base
Support for Multiprocessors	required	NA	NA	NA
Support for nonfunctional requirements	ARTEWG CIFO and Clear Lake Model	NA	NA	NA
Environments	Host, Target and Integration	Host	Host	Host (some Target)
I/O	graphics, windows and other devices	character-oriented terminals	graphics and windows	graphics and windows

Figure 3-1 Comparison of Features for a PCEE

50

3.5 Considerations for PCEE Guidelines

Inclusion of the capabilities discussed above into a standard for a PCEE impose constraints on both the RTSE as well as tools within the environment. Requirements for recognition of individual threads of control levies requirements on the RTSE not only to provide "hooks" into that type of information (which extends to dynamic task creation and retirement) but also creates a requirement for a "standard" manner of providing that information (across different ISA's). One solution might be to include this as an optional EA/RA represented feature within an extended runtime library (XRTL).

The choice of an appropriate unit of distribution also impacts the amount of information that must be provided by the compiler to the RTSE. The smaller the unit of distribution after a certain point, the greater the amount of information and, therefore, overhead is required. The amount of overhead incurred must not be so high as to negatively impact the performance of software in the target environment. On the other hand, the larger the unit of distribution, after a certain point, the less opportunity to exploit parallel and distributed processors which also begins to negatively affect performance.

3.6 Conclusions

The PCEE should provide a flexible, extensible model which addresses the three types of program execution environments. The features of the PCEE should be defined in a modular manner, so that when an environment is "scaled down", the overhead of a full set of capabilities is not imposed. Useful existing and emerging standards, such as IRDS and OSI should be employed and extended wherever necessary to eliminate duplication of effort and derive benefits from the corresponding development of conforming tools. The PCEE model must be robust enough to provide the required services, but at the same time must be easily implemented, tailorable, and extensible (extending to the capability to be implemented in a number of different ways). Finally, the choices for features, such as the granularity of control, should be made in a manner that provides for a technically feasible and economically reasonable solution.

4.0 Recommendations and Summary Discussions

The findings of this research team lead to five suggested actions:

1. Adopt CAIS as a necessary and extensible subset of the System Interface Set (SIS) and User Interface Set (UIS) of the distributed host environments (i.e., the SSE) and the integration environment (yet to be fully defined).

2. Publicly declare NASA's intention to help shape, develop, and utilize an upward compatible CAIS version which will combine the best features of the current CAIS and PCTE, and also meet specific needs of the Space Station Program including the dynamic perspective of the Portable Common Execution Environment (PCEE).

Discussion:

There are currently seven working prototypes of CAIS that have been reported in the public domain. Several more are on the way. Although they are not production quality, the reports clearly indicate that CAIS does indeed facilitate both developing and acquiring portable tools. By contrast, there exists one reported prototype implementation and one production quality implementation of PCTE. Unfortunately, the production quality version currently uses C language interfaces and therefore would be completely unacceptable for the life cycle of the Space Station Program. However, the recent conversion of these C specifications to an Ada specification set holds great promise for a subsequent production quality implementation featuring the advantages of building and importing tools which adhere to the Ada interfaces.

From the perspective of stable frameworks to host tools and rules, the current PCTE has many desirable features not found in the current CAIS. Unfortunately, the pragmatic considerations that led to the early availability of a production quality implementation of these desirable features are entirely too dependent upon obsolete models and paradigms within the underlying C language and UNIX System V. Therefore, CAIS-A has the unique opportunity of benefiting, in the long term, from the valuable lessons learned from working with both earlier versions of the CAIS and PCTE.

3. Publicly acknowledge that the requirements to simultaneously support:

    *   building-in and sustaining safety while
    *   meeting mission requirements and
    *   meeting future requirements for extensibility and adaptability

52

are issues-at-high-risk in the critical path of the Space Station Program. Furthermore, recognize that none of the current test bed activities now underway are specifically focused on these issues.

4. Establish a fully instrumented, highly reconfigurable PCEE test bed as soon as possible to support empirical verification and validation of the concepts and requirements of a PCEE intended to facilitate simultaneous support of:

- building in and sustaining safety while
- meeting mission requirements and
- meeting future requirements for extensibility and adaptability

in large, complex, non-stop, distributed systems such as the Space Station Program.

Discussion: From a logical perspective, the fully instrumented, highly reconfigurable PCEE test bed would be used to support:
- proof-of-concept demonstrations
- empirical evaluations
- and verification and validation

of:
- concepts
- principles

- models
- methods

and associated:
- standards
- guidelines

- practices
- policies

which are related to developing and sustaining a PCEE appropriate for the SSP life cycle.

From a physical perspective, the PCEE test bed would be used to support:
- proof-of-concept demonstrations
- empirical evaluations
- and verification and validation

of implementations of PCEE:
- services and resources
- stable interface sets
- layers
- stable frameworks
- reusable runtime library modules
- mappings from conceptual to implementation models
- mapping to-and-from the requirements and tools of the host and integration environments
- mappings to-and-from other NASA test beds and subsystems (e.g., Technical Management and Information System, TMIS).

53

Combing the benefits of these two perspectives, the PCEE test bed would be an invaluable resource to the SSP to facilitate:

- early emulation of an appropriate working environment
- early demonstration of the fault tolerance, security, and other components which support safety in the distributed target environment
- the study of integration issues in an end-to-end environment which is large, complex, non-stop, and distributed
- the study of one more aspects of the SSP life cycle management of:
    - complexity
    - safety and quality
    - cost effectiveness
    - technology transfer for the benefit of NASA and NASA's constituency.
- early evolvement and evaluation of the distribution of Ada entities among parallel and distributed resources
- other important studies which evaluate and lower risk in the execution environment of the SSP.

5. Because major SSP contracts such as TMIS and SSE have recently been awarded, publically declare the assignment of responsibility to an appropriate SSP office or working group to effect the integration of the System Interface Set (SIS) and User Interface Set (UIS) not only across the three environments (host, target, and integration) but also across the various SSP contracts (SSE, TMIS, DDT&E, etc.). This task should begin as soon as possible to reduce unnecessary risk, costs, and complexity.

Discussion:
Reports such as this one and the SERC memos on issues of transistioning from CAIS to CAIS-A (5 May 1987) and on computer systems and software safety (15 June 1987) could be used to stimulate discussion and planning of an initial agenda.

APPENDICES

A) McKay, C. "A Proposed Framework for the Tools and Rules to Support the Life Cycle of the Space Station Program", <u>COMPASS '87 Conference Proceedings</u>, IEEE, June 1987.

B) McKay, C. and P. Rogers. <u>Life Cycle Support for "Computer Systems and Software Safety" in the Target and Integration Environments of the Space Station Program</u>, SERC Set of Presentation Foils, June 1987.

APPENDIX A

A PROPOSED FRAMEWORK FOR THE TOOLS AND RULES TO SUPPORT THE LIFE CYCLE OF THE SPACE STATION PROGRAM

Charles W. McKay

Software Engineering Research Center
University of Houston Clear Lake

Abstract

In 1986, the author was the leader of a team commissioned to produce two reports for NASA concerning the requirements for a life cycle, Software Support Environment (SSE)for the Space Station Program (SSP). The interim report identified over 70 functional tools and seven standards or proposed standards that would be helpful in extending a Common APSE Interface Set (CAIS) conforming Minimal toolset of an Ada* Programming Support Environment (MAPSE) into an appropriate SSE for supporting the life cycle of the SSP. The final report described the requirements for an integrating foundation and framework to host the "tools and rules" identified in the interim report. This paper amplifies two vital boundary points considered by these reports: the proposed, principal goal and the proposed means for organizing a model of the SSE to effect this goal.

Introduction

This paper addresses two boundary points in environments which support the life cycle of large, complex, non-stop, distributed systems such as the Space Station Program (SSP). A useful model which depicts systems-level considerations in addressing these issues consists of three macroscopic, hierarchical levels: goals and objectives, strategies and tactics, and means.

The two focal points of this paper are the extremes of this hierarchical spectrum. The first is to state the goal for such applications and to identify four of it's most important components. The second identifies three of the lowest levels of appropriate means to achieve this goal: concepts, principles and models. Since the size of this paper cannot permit an adequate explanation of the many linkages and considerations necessary in bridging this spectrum, the reader should be aware

*Ada is a registered trademark of the US Government, Ada Joint Program Office

of the author's rationale for addressing these two widely separated sets of issues. The author is convinced that the "means" represented by conventional designs of most of the off-the-shelf: operating systems, data base management systems and data communications systems are inadequate to support the goals of the SSP regardless of how well stated are the objectives and how carefully chosen are the strategies and tactics to be used in organizing, developing and applying more traditional concepts, principles and models as the means to do the job. Therefore, a "bottom-up" presentation of proposed means believed to be adequate for the challenges will be described as follows: objects, Stable Interface Sets (SIS), layering, Stable Frameworks (SF), conceptual and implementation models of a computer systems and software support environment. These descriptions amplify related portions of the two reports referred to in the bibliography.

An Appropriate Goal

The team believes the major goal of the SSP should emphasize both process and product. The product should enable mankind to derive the benefits of colonizing and industrializing that portion of our solar system from the earth to the moon in preparation for future programs which extend into deeper space.

The process should enable distributed teams from government, industry and academia to learn how to incrementally develop and continuously sustain large, complex, non-stop, distributed systems which must be trusted to simultaneously satisfy a variety of critical requirements throughout the life cycle. Four important parts of this goal are believed to be:

1. To enable the successful life cycle management of the complexity of the computer systems and software integration and configuration management. (Ie, to enable the complexity to be controlled over the life cycle of a successful project.)

2. To support systems and software level safety and quality control throughout

the life cycle. (Ie, to continuously protect life and property.)

3. To support systems and software level cost effectiveness throughout the life cycle. (Ie, to enable mankind to afford deriving the benefits of the program.)

4. To transfer into practice those aspects of the systems and software engineering support environment which can increase the systems and software productivity, safety and lower life cycle costs of other projects throughout NASA and NASA's constituency.

Means

"The third rule was to commence my reflections with objects which were the simplest and easiest to understand, and rise thence, little by little, to knowledge of the most complex." (Descartes, _Discourse on Method,_ About 1620 AD

The discussion of means will begin with a description of the fundamental building block: the object. Although high level languages such as Ada can certainly be abused so that the desirable properties to be discussed are compromised, the remainder of this paper will assume that the facilities of the language will be properly used. Subsequent discussion of means will be based upon higher levels of object groupings to achieve the desired properties for the systems and software support environment.

1. Objects

As used in this paper, objects may refer to either logical or physical entities. All objects must have abstract specifications that answer three questions: What services and resources are to be provided by the object? How well are these services and resources to be provided? Under what circumstances are they to be provided? This abstract specification can be considered to provide a "virtual interface" to the object. From the perspective of users, the abstract specification also provides the "one way in/one way out" for the object. Since objects are intended to communicate by messages, the virtual interface can be used as an inventory of the message requirements. Abstract specifications are always to be separately compilable from the implementation part of the object if an implementation part exists.

The implementation part, if it exists, hides all design decisions regarding the object. Trade offs between: hardware and software, algorithms and data structures, and traditional versus AI design techniques are encapsulated inside this object implementation. Other information which may also be hidden inside the implementation part may include knowledge that this is a complex object which is composed of several other objects. For example, a component written in assembly language can be encapsulated inside an object body. Since users of the object can only see the abstract specification, access to the code segments and data structures of the assembly language routine would be hidden from the users and controlled inside the implementation part.

For objects implemented in Ada, the visibility and scoping rules are specified in the language reference manual. Details of authorization and enforcement of access control rights which are more restrictive than these visibility and scoping rules are systems level issues beyond the applications source code level. Entity-Attribute/Relationship-Attribute (EA/RA) models will be proposed in the next topic as the means to gain these additional and desirable controls.

2. Stable Interface Sets (SIS)

Stable Interface Sets (SIS) can be formed from a union of the abstract specifications of a group of objects which are designed to contribute to a common level of functionality and to offer the same level of visibility. Three SIS requirements listed below are best understood by considering the unique perspective listed for each.

1. From the perspective of a complete inventory, an SIS is a virtual interface set resulting from the union of all abstract specifications of the objects designed to be visible at this interface. (Ie, the total collection of all the services and resources to be provided at the SIS, how well and under what circumstances.)

2. From the perspective of any given object within an SIS regarding it's relationship to other objects in the SIS, there should exist a formal model in EA/RA form describing:

   - the "need-to-know" and "right-to-know" visibility among all objects of the SIS
   - the grouping of objects into discreet sets of services and classes of services within each of the sets.

The benefit of the formal model is not only to allow automated aids to assist in checks for completeness and consistency but also to provide rules for considering proposed extensions and modifications. The concept of classes of services within a given set of

services allows all external users of
the service set to have access based
upon their authorization and their
context of operation. (Ie, there are
things that must/should still be done
even when doing everything is no long-
er an option.)

3. From the perspective of external ob-
jects outside the SIS, a formal model
in EA/RA form is used to describe:

. exactly what sets and classes of
services are to be visible to the
object
. the protocols (necessary steps) and
access control to be imposed upon
the external object and it's use of
the services and resources provided
at the SIS.

3. Layering

Systems should be constructed in a
hierarchy of layers where each layer is a
Stable Interface Set. The layers
contribute to firewalling of faults that
occur within a layer. Upper layers are
often able to compensate for faults which
occurred in lower layers. The successive
layers of trust explicitly reflect both
the functional architecture of the system
and the differing degrees of criticality
associated with the functionality of the
layer. Similarly, faults and higher
layers are firewalled to prohibit
contamination in lower layers (although
the lower layers may not be able to
compensate for the higher level faults).
Such layering promotes isolation of
related units of functionality and
separates the concerns of "what" from
"how". Security, integrity and
reliability are three critical concerns
that can now be addressed appropriately
within each layer.

4. Stable Frameworks (SF)

The three requirements for Stable
Frameworks (SF) are:

1. Within any layer a collection of
closely related objects that should be
regarded and maintained as a unit
shall be identifiable by unique at-
tributes.
2. The collection of objects within a
layer which are to be identified as
a unit via the unique attributes can
be treated as "strongly typed". That
is, a complete determination can be
made of legal values and legal opera-
tions for SF's of this type.
3. Within any layer, a formal model in
EA/RA form can be used to represent
the relationships of the strongly
typed SF's to other strongly typed
SF's both within and external to the

layer.

As an example of the use of stable
frameworks, consider the systems level
requirements analysis for the Space
Station Program. One contractor might
be responsible for determining the re-
quirements for the Space Station it-
self while two other contractors are
respectively responsible for deter-
mining the requirements for the or-
bital transfer vehicle and the free
flying platforms. Therefore, in the
life cycle project object base, the
baselined requirements for the Space
Station will be uniquely identified by
at least the following attributes:
systems requirements analysis phase,
persistent object base layer (this
assumes they are approved and under
baseline control), Space Station re-
quirements. Similarly, the same lay-
er of the project object base might
contain a stable framework consisting
of all objects with the unique at-
tribute identifiers: systems re-
quirements phase, persistent object
base layer, orbital transfer vehicle.
These SF's may be regarded as strong-
ly typed with EA/RA models depicting
their relationships to other SF's plus
the access control that will protect
the products of one contractor from
accidental corruption by another.

5. The Conceptual Model

The top of the attached figure depicts a
conceptual model of the life cycle to be
addressed by a systems and software
support environment. The rectangles
labeled P 1 through P 6 represent phases
of the life cycle. The elongated S shaped
figure to the right of the ellipse is
marked P 7, maintenance and operation.
This icon represents successive iterations
through the first six phases. For the
purposes of this model, a phase may be
defined as: a discrete period of time
and activities delineated by a beginning
and an ending event for each iteration in
the incremental evolution of the life
cycle.

The first phase, P 1, represents system
requirements analysis. All subsequent
phases may involve three sets of staggered
activities in time. For example, P 2
begins with the translation of a section
of the system requirements into software
requirements. In turn, these two will
later contribute to a P 2 activity
referred to as hardware requirements
analysis. Finally, a third P 2 activity
will use these three sets of results for
operational requirements analysis.
Similarly, the resolution into software,
hardware, and operational concerns are
mapped in staggered time to P 3,

preliminary design, and the subsequent phases. The fourth phase is detailed design. This precedes P 5, coding and unit test, which exists in a staggered time relationship to ongoing activities of P 6, computer software component integration.

The closed pairs of parallel arcs represent documentation requirements tailored from DOD Standard 2167 to meet the needs of NASA. For example, the closed pair of parallel arcs separating the rectangles for P 1 and P 2 represents the systems requirements analysis documentation. This documentation set creates a vertical stable interface set separating each iteration of systems requirements analysis on the left from the beginning of a transformation into software requirements on the right. For example, the first iteration of systems requirements analysis, P 1, might satisfy a minimum treshold of requirements for a small, identifiable segment of the systems requirements documentation. When this threshold is reached, automatically a signal is triggered to freeze the attributes of that segment of the document and to signal the quality management team they can begin verification and validation (shown in the small circles). Upon the recommendation of the quality management team to project management, a configuration management decision (shown in the tab as "SDR", system design review) determines whether this portion of the document should be placed under configuration management. The decision is forwarded to the project object base which also triggers both a report to the team doing systems requirements analysis and a signal to initiate activities of the team who will take this portion of the systems requirements and begin transforming them into software requirements. Later, the software requirements analysis will create a corresponding segment of the software requirement analysis document shown as the closed pair of parallel arcs separating P 2 and P 3. When a threshold for this segment of the document is reached, automatically the system is triggered to freeze the attributes of this part of the work so that the results maybe evaluated by the quality management team. Other documents identified in the figure include the: software design specification document, software design documentation, software development documentation. As stated earlier, the circles represent verification and validation activities by members of the quality management team. These activities are in accord with a version of DOD Standard 2168 tailored to meet the needs of NASA.

The tabs represent configuration management decision points. The first one

shown on the left is the "SRR", system requirement review. This represents the decision of the client to award the contract for a particular portion of the automated system to a contractor. At this point, an instantiation of the "tools and rules" plus the environmental framework is established for the contractor and the contract becomes one of the first items in the life cycle project object base to enter configuration control. Subsequent configuration management decisions are identified from left to right as: system design review, software specification review, preliminary design review, critical design review, test readiness review, functional configuration audit, physical configuration audit, formal qualification review.

The ellipse which is to the left of the P 7 icon represents acceptance testing. This is a transition milestone from the acceptance of a developed baseline for the target environment to the maintenance and operation that sustains the baseline in the future. The life cycle project object base shown at the right hand side of the figure supports: systems engineering, software engineering, hardware engineering, operational engineering, and the management of people and logistics. (Please note: the icons and general organization of the conceptual model were adapted from McDermid and Ripken, 1984.)

6. An Implementation Model

In contrast to phases, activities have been defined as: the process of performing a series of actions or tasks. Thus, some activities take place within phases. Others, such as quality management, integration and configuration management, information and object management, document generation, and other forms of communication, are pervasive throughout the life cycle. Together, the concepts of phases, activities, a life cycle project object base, and required documentation as stable interface sets help to explain the mapping of the conceptual model to the implementation model at the bottom of the figure.

From the perspective of the users of the environment, the first stable interface set which is experienced is known as the "User Interface Set" (UIS). The UIS is defined as a part of the Common APSE Interface Set (CAIS). It provides an integrated view of the users access to the tools and other services and resources of the environment via their terminals or work stations.

Via the UIS, the user is probably connected to the services and resources provided by one of two stable frameworks:

either the technical toolset or the management toolset for the particular phase of interest. If, for example, the methodology chosen for the systems requirements analysis phase (P 1) is COntrolled Requirements Expression (CORE), then the technical tools in the stable framework for that phase are intended to reinforce the correct use of the methodology and to promote productivity of the requirements engineers. Similarly, the stable framework composed of the management tools for that phase are both complementary to the requirements for properly applying the methodology and for the needs of the project managers. If the methodology chosen to transform the systems level requirements into software requirements is Structured Analysis and Design Techniques (SADT), then the technical tool set that composes the stable framework for reinforcing the chosen methodology is shown under the corresponding phase in the conceptual model. Two points are worth noting. First, phase identification attributes will be a property of all objects, relationships and attributes created by the tools of a given phase. Thus, one can look at the figure and imagine a stable framework in the lowest layer of the first phase that represents the current baseline of system requirements for a free flying platform. These requirements had to be captured by authorized users applying the technical tools of the P 1 phase. Although the tools of the second phase may have read-only access to the stable framework so that the transformation from system to software requirements may take place, the stable framework which will compose the software requirements under baseline control in the bottom row of the second column of the figure could only have been developed by the technical tools for the software requirements analysis phase. Furthermore, the use of these identification attributes to impose strong typing and access control insures that only the authorized teams from the appropriate contractors using the appropriate phase specific tools can modify the contents of the stable frameworks. The second major point considers a possible decision to change the methodology employed in the P 2 phase. Since P 2 exists between the stable interfaces of the systems requirements analysis documentation (stored in the life cycle project object base) and the software requirements analysis documentation (also in the project object base), then the change procedure will be to acquire/develop new technical tools and management tools for that phase and to install them as the stable framework replacement for the older tools. Note that this is possible because the technical tools and management tools for

each phase are horizontally bounded as stable frameworks between the UIS and the SIS and they are vertically bounded by the stable interface sets which represent the documentation requirements for transitions between phases. (Actually, the phase and layer identification attributes accomplish this vertical bounding.)

The next stable interface set is the "System Interface Set" (SIS) of the CAIS. The CAIS UIS and SIS provide a framework for importing and/or developing new tool set capabilities. The reader should note that if any of the chosen tools were developed in a language other than Ada, the recommendation of the author is to encapsulate the tool inside the implementation part of an Ada object. In this manner, users who access the tools via the UIS can only see the services and resources of the abstract specification of the object containing the tool. Therefore, no life cycle dependencies upon any of the code segments or data structures of the foreign language can inhibit the subsequent evolvement of the environment. The reader should also note that the SIS of the CAIS is specified in far more detail than the UIS. The reason is to facilitate the development of truly transportable tools among Ada Programming Support Environments (APSE).

The next two layers consist of the Stable Interface Sets for quality management and integration and configuration management respectively. The justification of ordering flows naturally from the activities previously described in the conceptual model above. That is, when an iteration through a phase has produced sufficient documentation for a given segment of the document, a trigger freezes the attributes of the segment from further change so that the quality management team may examine it. This in turn is followed by reports to management and appropriate control boards which then results in a configuration management decision.

The next layer is bounded by the stable interface set specifying the services and resources to be provided and consumed by information, library, and object management. The services and resources available at this virtual interface set hide implementation details from the user, tools and layers above. There are three macroscopic layers shown beneath the information, library and object management services layer, although each of these layers may in turn be subdivided depending upon the size and complexity of the project. The first layer is used to facilitate tool-to-tool communication. All tools should communicate through the project object base. When tools communicate directly to one another,

valuable audit trail information is lost and properties of recovery, security, integrity, and reliability are severely compromised. The reader should note that as tools and the tool set for a particular phase produce objects, relationships and attributes, the tool-to-tool communication that takes place in this layer had to pass through each of the preceding layers so that an appropriate history and management control is exerted upon the products at all time. When objects, their relationships, and associated attributes are proposed for consideration in the next baseline, they are advanced into the temporal level of the life cycle project object base. Again, attributes for the stable frameworks always identify the life cycle phase (Ie, column), the layer (ie, row), and the unique identification of the stable framework of related objects that are being considered. When the proposed changes are approved to roll the current baseline forward, and the integration and acceptance testing of the new baseline has been completed, a change of attributes installs the stable framework in the "persistent" object base layer representing stable frameworks under baseline control.

Summary

For large, complex, non-stop, distributed systems such as the Space Station Program which evolve incrementally while they are being continuously sustained and which must simultaneously satisfy a large collection of critical requirements, there are four tyrannies that must be avoided in the systems and software support environment. Specifically, these include life cycle dependencies upon any proprietary or particular: operating system, data base management system, data communications system, and instruction set architecture. Otherwise, the author believes strongly that a combination of all three requirements for: complexity management, safety management, and cost effectiveness will be unacceptably compromised. To avoid these dependencies, this paper amplifies a proposal to use a graduated series of concepts and principles to evolve a conceptual and an implementation model sufficient for the life cycle of such projects. These are: objects, stable interface sets, layers, stable frameworks and conceptual models.

Acknowledgments and Disclaimers

Bibliography

McDermid, J. & Ripken, K. Life Cycle Support in the Ada Environment, Cambridge University Press, 1984.
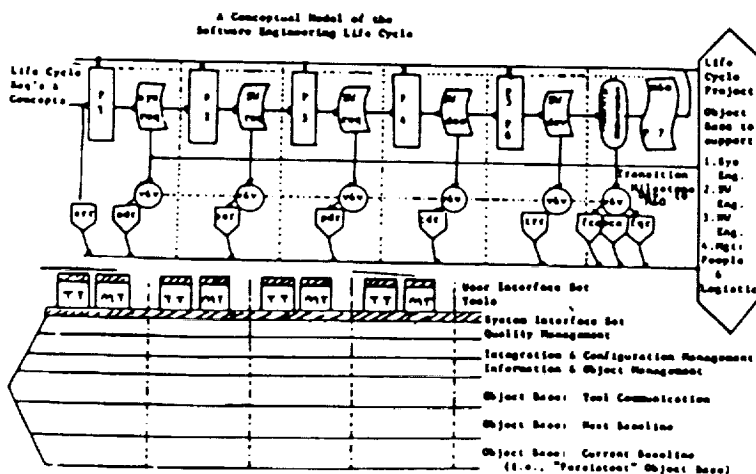
McKay, C., Charette, R., & Auty, D.,
. An Interim Progress Report on:
. A Final Progress Report on:
   A Study to Identify Tools Needed to Extend the Minimal Toolset of the Ada Programming Support Environment(MAPSE) to Support the Life Cycle of Large, Complex, Non-Stop, Distributed Systems Such as the Space Station Program. Task Order No. UH JSC 811, Contract No. NAS 9-17010, 1986.
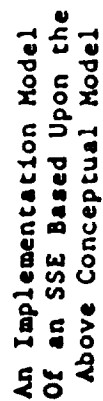
A Conceptual Model of the
Software Engineering Life Cycle

Life Cycle
Req's &
Concepts

Life Cycle Project Object Base to support:
1. Sys Eng.
2. SW Eng.
3. HW Eng.
4. Mgt: People & Logistic

Transition Milestone Day to H&O

m&o   P 7
operation & ...

SW dev   v&v   trr
P 5   P 6

SW des   v&v   fdr
F 4

SW spec   v&v   pdr
P 3

SW req   v&v   ssr
P 2

sys req   v&v   sdr
P 1   srr

v&v   fcabca fqt

User Interface Set
Tools

System Interface Set
Quality Management

Integration & Configuration Management
Information & Object Management

Object Base: Tool Communication

Object Base: Next Baseline

Object Base: Current Baseline
(i.e., "Persistent" Object Base)

TT MT   TT MT   TT MT   TT MT   TT MT   TT MT   TT MT

An Implementation Model
Of an SSE Based Upon the
Above Conceptual Model

APPENDIX B

Lifecycle Support
for
"Computer Systems & Software Safety"
in the
Target & Integration Environments
of the
Space Station Program

Charles McKay & Pat Rogers, Software Engineering Research Center
University of Houston – Clear Lake

Safety : the probability that a system, including all hardware
and software and human–machine subsystems, will
provide appropriate protection against the effects of faults,
which, if not prevented or handled properly, could result in
endangering lives and property (CWM, 1987)

Systems and Software Safety

Key Terms & Concepts

Fault

System Reliability

Fault Avoidance

Fault Tolerance

Security

Security Kernel

Safety

From Reason to Rationalization

"A requirement that cannot be feasibly met is not a requirement."

$\rightarrow$

"Until we know how to construct better execution environments, additional efforts beginning with requirements analysis are probably not justifiable."

$\rightarrow$

"Business – as – usual"

A Proposed

Clear Lake Model for Computer Systems and Software Safety

in a

Portable Common Execution Environment (PCEE)

A baseline from which subsequent progress in the target environments and integration environments may be made

An extensible model which can also scale down to improve safety in smaller, simpler applications

A "lessons implied / learned" stimulus and opportunity to develop host environment methodologies and tools which better address the lifecycle issues of safety

Clusters : Fundamental Units of Distributed Systems

Four classes of components:

— various types of processors with shared access to memory subsystems (eg, IOP's GDP's, etc.)

— memory subsystems which must include volatile and stable storage subsystems and will often include nonvolatile storage subsystems

— communication links to other clusters

— a sharable set of services and resources which are available to the various application programs

Clusters

Activities:

- communicate with other clusters only via messages

- be partially operational

- fail completely

- recover from either partial or complete failure

- be added at any time

- be removed at any time

- be reconfigured at any time

- have their functionality changed

Clusters

Classes of Tolerable Faults:

- communication failures (eg, lost or duplicated messages)

- abortion of application program components
  (eg, to preempt resources required for an emergency)

- crashes of one or more participating clusters
  (eg, total loss of power)

- lock cycles (eg, dead-locks, live-locks)

if

'safety' is adequately addressed in the host environment throughout development/acquisition

then

the support of this explicitly addressed set of safety specifications will be dependent upon a runtime environment built to:

1. Monitor the system and detect faults that enter the system state vectors ASAP

2. Firewall their propagation

3. Analyze their effects

4. Recover safely

end if ;

Application Level Visibility of the SIS

Required Visibility --->

- Ada Source Code
- + Objects
- + Nested Transactions
- + Messages for Distrib., Nested Transactions and Command Lang.

— Even here, "good" Ada code is built from objects

Class of Application Program Requirements:	Ada Source Code	+ Objects	+ Nested Transactions	+ Messages for Distrib., Nested Transactions and Command Lang.
Local to 1 Cluster, No Fault Tolerance	●			
Distributed Among Clusters, No Fault Tolerance	●	●		
Local with Fault Tolerance	●	●	●	
Distributed with Fault Tolerance	●	●	●	●

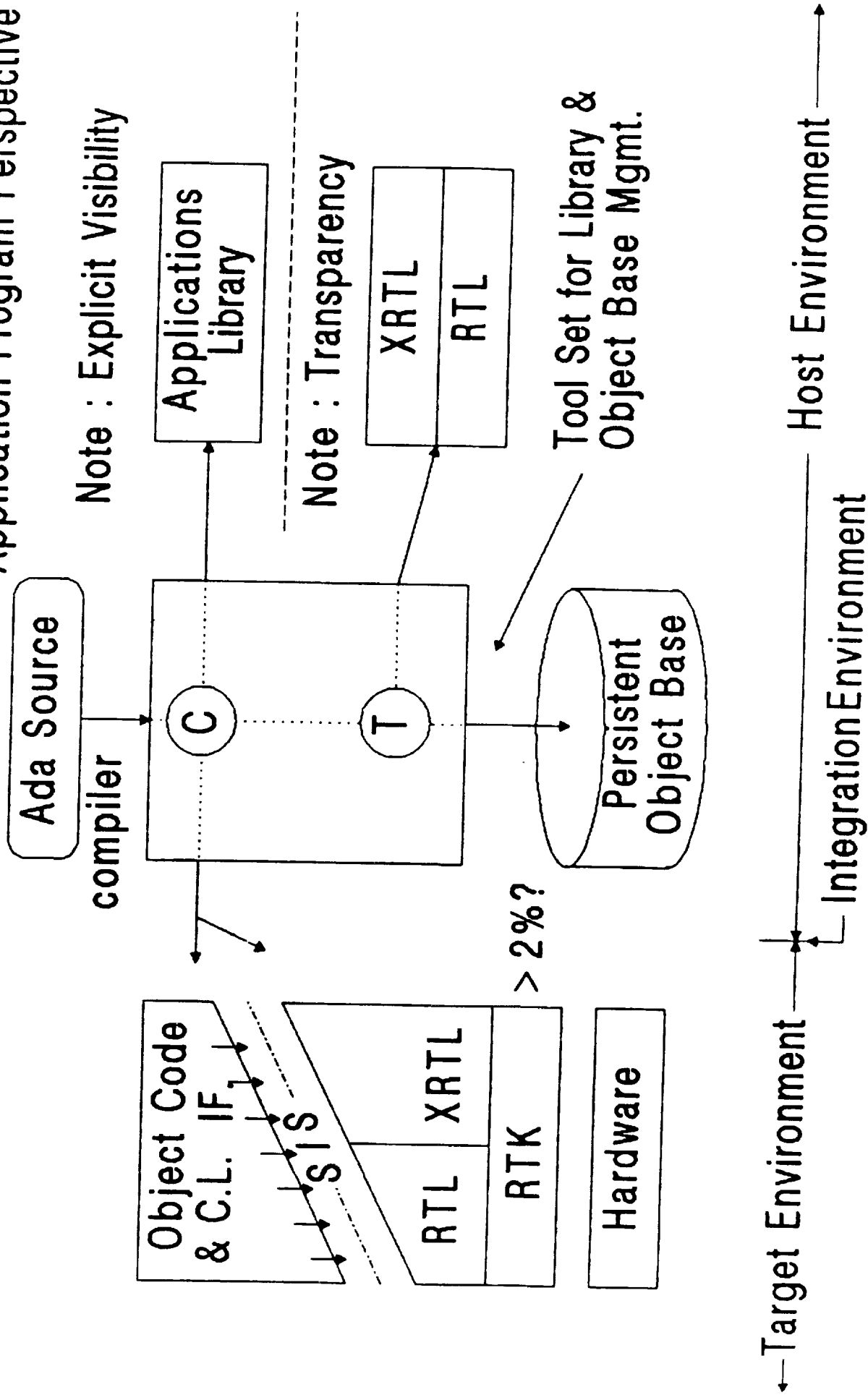# To support safety, the SIS should be built upon:

1. A tailorable RTSE developed & sustained in Ada

2. Software structuring which facilitates: firewalling, layered recovery capabilities, dynamic reconfiguration and extensibility

3. Pools of processes and processors capable of non-stop operation in a fault-tolerant environment

4. A command language interface between the SIS of the integration environment's PCEE and the SIS of the target environment's PCEE

5. System-wide, lifecycle-unique identification of all objects and transactions / subtransactions

6. Dynamic, multilevel security in the integration & target environments
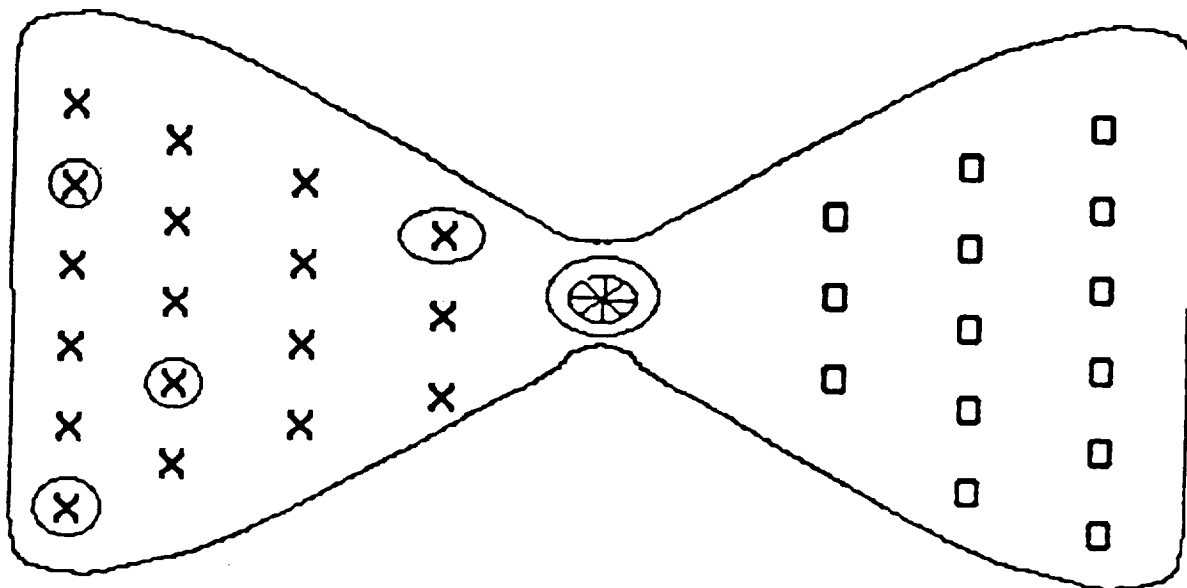
(more)

7. A message interface which supports three forms of communication among clusters: asynchronous send/receive with 'no waits', remote procedure call, Ada rendezvous

8. Hierarchical runtime structure of the threads-of-control

9. A redundancy management subsystem for services and resources which life and property depend upon

10. A stable storage subsystem for each cluster

11. A management subsystem for distributed, nested transactions

12. A multiversion, fault-tolerant programming capability with a granularity within any program which extends at least to the subtransaction level and explicitly identifies the recovery capabilities at that level

A Model for Supporting a 'Bare Machine' Philosophy for 'Safety Kernels' of Ada Runtime Support Environments (Ada RTSE's)

Application Program Perspective

Note : Explicit Visibility

Note : Transparency

Ada Source compiler

Applications Library

XRTL

RTL

Tool Set for Library & Object Base Mgmt.

C .... T

Persistent Object Base

> 2%?

Object Code & C.L. IF

SIS

RTL | XRTL

RTK

Hardware

Target Environment

Integration Environment

Host Environment

TWO SCENARIOS FOR SSP ENVIRONMENT IN 2000+ A.D.

HOST ENVIRONMENTS:	INTEGRATION ENVIRONMENT:	TARGET ENVIRONMENTS:
• DEVELOP • SUSTAIN	• CONTROL OF TGT. ENVIR. BASELINE • INTEGRATION V&V FOR NEXT BASELINE AND TEST & INTEGRATION PLANS	• DEPLOY • OPERATE

Bibliography

ACM SIGAda Ada RunTime Environment Working Group (ARTEWG), <u>A Catalog of Interface Features and Options for the Ada Runtime Environment</u>, ARTEWG Interfaces Subgroup3, Release 2, 23 July 1986.

Auty, David, <u>Ada and Operating Systems Practice and Experience in Targeting Ada</u>, presentation to NASA/JSC, April 1986.

CAIS Panel Discussion, First International Conference on Ada Programming Language, 2-5 June 1986, Session D.5.1.

Chen, C. "Conceptual Architecture for an Ada Run-Time Environment", Rockwell SSSD IR&D Progress Report 86567, (fall 1987).

Dolk, R.D. and R. A. Krisch II. "A Relational Information Resource Dictionary System", <u>Communications of the ACM</u>, Vol. 30, No.1 (January 1987).

Fisher, Herman, <u>PCTE Overview and CAIS Comparison Impressions</u>, 9 September 1985.

Fisher, Herman, <u>PCTE Ada Conceptual Design (PCD)</u>, Mark V Business Systems, Draft of 22 November 1986.

KAPSE Interface Team (KIT), <u>DoD Requirements and Design Criteria for the Common APSE Interface Set (CAIS)</u>, 13 September 1985.

KIT Meeting, presentation by CAIS-A contractor, April 1987.

Mark V Business Systems/Systems Designers PLC., <u>PCTE Ada Interface Requirements</u>, Version 1.1, 27 December 1986.

McKay, C. "Distributed Computer Systems and Software Safety", SERC Lecture Notes, April-June 1987.

McKay, C. <u>A Proposed Framework for the Tools and Rules to Support the Life Cycle of the Space Station Program</u>, COMPASS '87 Conference Proceedings, IEEE, June 1987.

McKay, C. <u>Life Cycle Support For "Computer Systems and Software Safety" in the Target And Integration Environments of the Space Station Program</u>, SERC Memo, 15 June 1987.

McKay, C. "CWM's Perspective of:
- Probable enhancements to transition CAIS to CAIS-A
- Potential implications for 3 environments of the Space Station Program (host, target, integration)", SERC Memo, May 5, 1987.

McKay, C., R. Charette, D. Auty <u>Final Report on: A Study to Identify Tools Needed to Extend the Minimal Toolset of the</u>

Ada Programming Support Environment (MAPSE) to Support the
Life Cycle of Large, Complex, Non-Stop, Distributed Systems,
SERC, July 1986.

Military Standard Common APSE Interface Set (CAIS), MIL-STD-1838
31 January 1985.

Space Station Software Support Environment Functional
Requirements Specification, National Aeronautics and Space
Administration, Johnson Space Center, JSC 30500, Draft 3.0,
(6 April 1987).

Notkin, D. et. al. Heterogeneous Computing Environments: Report
on the ACM SIGOPS Workshop on Accommodating Heterogeneity.

PCTE A Basis for a Portable Common Tool Environment, Project
Report, ESPRIT Technical Week 86.

PCTE A Basis for a Portable Common Tool Environment Ada
Functional Specification, First Edition, Volume 1.

Rogers, P. and C. McKay. "Distributed Program Entities in Ada",
Proceedings of the First International Conference on Ada
Programming Language, 2-5 June 1986, p B.3.4.1.

Rogers, K. "Extending the Granularity of Representation and
Control for CAIS Process Nodes", Proceedings of the First
International Conference on Ada Programming Language, 2-5
June 1986, p D.2.3.1.

Thall, R. and S. LeGrand. "The CAIS 2 Project", Proceedings of
the First International Conference on Ada Programming
Language Applications for the Space Station Program, 2-5
June 1986,
p D.2.6.1.