View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by NASA Technical Reports Server

N89-16314 167059 11P.

The Impact of Common APSE Interface Set Specifications on Space Station Information Systems.

by

Jorge L. Diaz-Herrera and Edgar H. Sibley
George Mason University, Fairfax, VA

ABSTRACT

Certain types of software facilities are needed in a Space Station Information Systems Environment; the Common APSE Interface Set (CAIS) has been proposed as a means of satisfying them. This paper discusses how reasonable this may be by examining the current CAIS, considering the changes due to the latest Requirements and Criteria (RAC) document, and postulating the effects on the new CAIS 2.0. Finally a few additional comments are made on the problems inherent in the Ada (*) language itself, especially on its deficiencies when used for implementing large distributed processing and database applications.

1. INTRODUCTION

Certain types of software facilities are needed in a Space Station Information System Environment (SSISE). Not the least of these are:

a. the distribution of the target and host facilities for both the run-time and development environment,

b. the absolute need for good configuration management methodology to control the development and use of the many versions of the software and tools,

c. the need to develop and modify systems within distributed environments using sophisticated terminal interfaces,

d. a consequent need for good interfaces and standards, abstract data typing in a distributed system (including development and run-time bindings),

e. a real-time distributed software development methodology, and corresponding language support and operating environment and tool constructs,

f. good human to human and machine to machine communication techniques.

----

* Ada is a Registered Trademark of the U.S. Government, Ada Joint Program Office

ORIGINAL PAGE IS OF POOR QUALITY

Because SSISE development will use Ada as its implementation language, it would be extremely unfortunate if its needs were not addressed in the Ada environments now under specification and development: the Common APSE Interface Set (CAIS). This paper is structured around the following three major aspects:

1. How well are these needs addressed within the current CAIS specification? Indeed, would a poor fit have a bad effect on the Space Station software?
2. What improvement can be expected due to changes mandated by the latest Requirements and Criteria (RAC) document?
3. Will this truly affect the next CAIS (version 2.0)?

2. <u>SPACE STATION INFORMATION SYSTEMS ENVIRONMENT NEEDS AND THE CAIS</u>

The Space Station Software Working Group and NASA software specialists have recently defined their needs for support of space station software development [Dixon 85], and produced a definition of the space station software support environment requirements [Chevers 86] in early 1986. The major issues include aspects about generic elements of the environment, tool characteristics and consideration of the following major questions:

- Should a uniform NASA Software Development Environment for space station be defined and developed? Issues relative to this include:
  * Software development for the space station will be highly distributed, with no localized single development group.
  * Major software portions will be managed by various centers and not by a single NASA center.
  * Important functional differences exist between major software systems; these need completely separate software environments.
- How much of the space station software development environment should be furnished by NASA?
  * This had a major impact because NASA has never developed its own SDE.

2.1 THE SSISE AND ITS REQUIREMENTS

Despite the fact that the specification of a single standard environment may involve solving many problems, the working group felt that the potential advantages far outweigh the difficulties. There was therefore a recommendation for the definition of a well-defined development environments with capability for two classes of user:

ORIGINAL PAGE IS
OF POOR QUALITY

- SDE interfaces to support software developers and their managers. These were to consist of:
  * Mail and Telecommunication support (e.g., editors, file systems, communications aids, etc.)
  * Technical management/control aids (e.g., cost models, project management and planning systems)
  * Data base support (e.g., file management, retrieval, control, etc.)
  * Modeling/simulation aids (e.g., Architecture models, testing aids, etc.)
  * Prototyping aids (e.g, requirements, specs, man/machine interface, etc.)
  * Document preparation aids
  * Requirements specification validation and analysis aids
  * Design specification aids (e.g, PDL analyzers, data dictionary, etc.)
  * Code construction and control aids (compilers, linkers, configuration managers, etc.)
  * Program analysis/testing and integration aids (path coverage/test generators, symbolic executors, etc.)
  * Metrics (quality, complexity, cost and reliability measures)
  * Man-machine interface support (interface and use of the environment, help, tutorial, etc.)
- An SDE interface to support NASA software managers responsible for software requirements/acquisition/acceptance; this required essentially the same capabilities as those above, with changes in emphasis or tailoring the relative importance, complexity of function and response needs. Thus the management controls should be more heavily directed toward schedules, planning, project management, and PERT, while the modeling, prototyping, and simulation aids would be minimal or unnecessary.

These two interfaces can thus result from a single CONFIGURABLE environment which is tailored to the specific needs of each work station and locale.

2.2 THE CURRENT CAIS

Several needs in the above list have not been addressed in the CAIS 1.4 specification. These issues have been discussed at length in KIT (KAPSE Interface Team) and KITIA (KIT Industry and Academia Support) group meetings but are, as yet, only partially resolved. In fact, many of these were deliberately excluded from discussion in the current CAIS document. They are:

ORIGINAL PAGE IS
OF POOR QUALITY

* A particular Configuration Management Methodology
* Sophisticated Device Control and Resource Management Capabilities
* Distributed Environments
* Inter-tool Interfaces
* Interoperability
* Typing Methodology
* Archiving

These and other issues are each discussed in the detailed sections below.

2.3 THE EFFECT OF THE RAC

Although the requirements of the first version of the CAIS were never explicitly defined, they were a mixture of the specification and partial implementation of the ALS provided by Softech and the AIE under design by Intermetrics. Thus, because these two efforts were already funded, they introduced several problems because the CAIS specification team were attempting to provide as much compatibility as possible with these two, somewhat different, architectures of an environment (with differences also in their scope). In general this attempt may have introduced problems of upward compatibility. Thus the future CAIS will either have to ignore the normal needs of a "standard" in dealing with a required "upward compatibility" or else admit to serious deficiencies and possible poor interfaces in future systems due to lack of adequate controls and functions.

The new requirements were written to allow more flexibility and better interfaces, with an attempt to have better functionality. Thus the Entity Management Support (section 4. of the RAC) requires a support that parodies the description of a normal database management system without specifically saying that it is needed. Some of the needs are quite specific and (though open to interpretation) quite encompassing; e.g., "impose a lattice structure on the types which includes inheritance of attributes, attribute value ranges (possibly restricted), relationships and allowed operations."

Another type of problem arises due to wish to allow the CAIS to be operable on almost any current commercial and experimental operating system: viz, "The CAIS specification shall be machine independent and implementation independent. The CAIS shall be implementable on bare machines and on machines with any of a

ORIGINAL PAGE IS
OF POOR QUALITY

variety of operating systems." This could restrict the design in many unfortunate ways.

2.4 THE NEXT CAIS

It is difficult to peer into the future, and thus the following predictions for the next CAIS may prove incorrect, however, the degree of effort and choice of contractor (Softech) allows us to make some early assumptions.

First, it seems unlikely that the contractor would make a new specification that would not allow the current ALS to be considered an "almost complier with" or "minimal fix away from" the new CAIS.

Second, the level of funding and staffing is not one that would be expected to allow anything but the narrowest extension of the current CAIS.

Third, it is somewhat doubtful whether the politics of the situation would allow a large diversion from the Army's ALS.

Fourth, the contractor has already suggested that divergence from some of the old CAIS specifications to go to the RAC statements would be difficult. The discussion of such issues at recent KIT/KITIA meetings has not been encouraging to a feeling of extension of the role of the CAIS.

3. SPECIFIC DEFICIENCIES

3.1 CONFIGURATION MANAGEMENT

The lack of a particular Configuration Management Methodology means that several vendors could provide incompatible but "standard" systems. These issues seem, primarily, to devolve on a need for a long time naming continuity and, in general, software configuration management. The first issue is that of providing "Unique Names" across geographic and time boundaries. The term "unique name" (UN) has been used to define an immutable name for an entity; e.g., a compiler should be uniquely identified by a UN, which neither changes nor is "recycled". Thus a UN is given out once to an entity and remains its name; if the entity is deleted/removed, then the UN will still identify the

ORIGINAL PAGE IS
OF POOR QUALITY

entity, but an attempt to retrieve it will result in a statement that it is no longer available.

There are two possible problems:

1. Is any sort of change allowed to an entity without its UN changing? Normally, the contents of the entity may be altered, but this could mean that it is no longer even similar to it previous "parent" entity. Certainly, it seems reasonable that a program may be debugged without changing its name for each error detected. This would suggest that the unique name was really a run-time UN, which could be said to remain constant during programming and debugging. However, if the UN were for a data entity, the effect of a change in any one of its values would be a new "version" of the entity, and this could be important enough to be considered a new "entity" though the normal way of dealing with this is to consider the data entity to be "time and data stamped" with an audit trail to allow the previous entity to be reconstructed (e.g., for roll back).
2. How are UN related for the same (but changed) entity? There must be a method for data entity reconstruction -- roll back from an audit trail, however, the data in a traditional database must not be called by physical location, but by "name pointers" or indexes or "logical" keys -- these might be considered the UN for data. On the other hand, the only "audit trail" for programs is normally provided by the configuration management system (CMS). In fact, the idea of version in a CMS is another way of looking at the unique name; i.e., the UN is logically equivalent to a user name concatenated with the version number (or equivalent).

What has been suggested above about a UN for both program and data could also hold for control structures.

3.2 SOPHISTICATED DEVICE CONTROL

Some of the biggest problem are undoubtedly going to be the introduction of more sophisticated input/output and other special device dependent interfaces (e.g., for a mouse). This will be a problem when there are unusual but sophisticated interfaces to devices and sensors. Unfortunately, this issue will require too much discussion to fit here and requires a paper of its own.

ORIGINAL PAGE IS
OF POOR QUALITY

3.3 DISTRIBUTED ENVIRONMENTS

The development of Space Station Information Systems is bound to be highly distributed with no single group solely responsible for the required software systems. This could result in difficulties when looking at large and complex development and run-time environments. Discussions on space station software development must address Distributed Environments (Host and Target) and particular ways to distribute data as well as control. The Ada Programming Support Environment (APSE), however, does recognize such a need, and states that additional software tools are necessary in order to allow "independent" programs to communicate with each other dynamically, in a "natural" and controlled way. The RAC states, however, that: "CAIS program execution facilities shall be designed to require no additional functionality in the Ada Run-Time System (RTS) from that provided by Ada semantics. Consequently, the implementation of the Ada RTS shall be independent of the CAIS"...

There are some problems here with Ada itself. A distributed system can be designed and implemented in Ada from two different points of view, namely as a single program or as a collection of cooperating programs. The first of these alternatives, single program, is particularly useful when considering tightly coupled multiprocessor systems. Inter-processor communication and synchronization can then be naturally achieved by using rendezvous. The second alternative is to design the system as several independent programs (one per processor). The Ada language, however, does not support the idea of independent programs dynamically cooperating with one another (i.e., no constructs are provided for inter-program communication).

Both approaches require further support from the environment. For example, Specific target-oriented tools (e.g., loaders) are needed, to assist in the actual implementation on the distributed architecture. An Ada solution to these problems may be in the form of a set of inter-program communication primitives provided at the APSE level in the program library. In general, the design and implementation of a multi-processing system as a collection of independent programs present a number of inconveniences resulting in the following issues:

- Creation of "linguistic" facilities to enable interprogram communication
- Provision of a methodology for designing Distributed Systems using these higher-level primitives.

ORIGINAL PAGE IS
OF POOR QUALITY

3.4 INTER-TOOL INTERFACES, INTEROPERABILITY, AND TYPING METHODOLOGY

These three issues represent the generic problem of the tool builder. When several tools must interchange data, they must either do it via the standard interfaces or else be designed as a suite of tools with total knowledge of the data requirements and functionality of the other tools in the suite. In general, there are problems in defining inter-tool interfaces, because a change to one tool may cause a ripple effect. However, reliance on interoperability interfaces entails passage of abstract data types across tool interfaces. This could have serious security and integrity repercussions.

Interoperability also has severe impact on distributed systems, where the passage of abstract types may be essential for accurate and reliable data interchange between the various nodes. Without a good typing methodology, it is obviously impossible to provide such features or to deal effectively with data base management and similar issues. The alternative to such methodology is of course straight ASCII interchange, with negligable checking. Again, these topics deserve a paper of their own.

3.5 ARCHIVING

This is an important issue in any configuration, but more so in a distributed environment of the kind mentioned here. However, for the purposes of this paper, it will be left as another undiscussed issue.

3.6 CENTRALIZATION AND DECENTRALIZATION ISSUES

The really tough problems of unique names of any of the types of entities occurs when they are (in some way or another) decentralized. As an example, when a compiler is moved to a new node, does its UN change? And whether it does or not, which node controls or restricts the change? Obviously, the answer to such questions involves policy and method of control. It is important that the controlled use of a distributed environment be effected through distributed kernels operating locally. It is conceivable that one or more nodes would be designated as decision making kernel(s), while other nodes will be merely servers. This seems to provide a reasonable compromise between centralized (high communication costs and high vulnerability) and decentralized (with its unnecessary control burden on every node).

ORIGINAL PAGE IS
OF POOR QUALITY

4. ADA LANGUAGE ISSUES

As discussed earlier, there are some severe problems in using Ada in multiprocessing and distributed systems. From Ada's point of view, a multiprocessor system which uses a common memory can be viewed as a "uniprocessor system which implements multitasking in a more efficient manner." In this case, the entire system is designed and built as a single Ada program with certain procedural abstractions implemented as tasks. Each of these tasks represents the work of one logical processor, and may eventually run on a dedicated physical processor. Inter-processor communication and synchronization can then be naturally achieved by using rendezvous. However, before the program is run on the target multi-processor environment, the different tasks need to be "assigned" to their corresponding processors. And this is not explicitly supported by the language. The use of PRAGMAS has been suggested here. On the other hand, a distributed system may be supported by Ada as a collection of Ada programs communicating through intermediaries. One way would be to provide library packages to maintain "mailboxes" and whose "procedures" (which could be implemented as tasks) can be called from several programs. In any case a standard protocol is needed.

An Ada solution to these problems may be in the form of a set of inter-program communication primitives provided at the APSE level in the program library. Basically, what we are talking about here is a general facility by which programs can communicate and synchronize their activities. These facilities must be designed in such a way that they could be applied in a number of situations using different programs. Thus, the specification must be general enough as to hide the identification of the programs involved, and yet provide ways to identify a particular situation. Ada's generic units provide the answer. They are general at the definition level, and particular at the instantiation level.

Unfortunately, the use of generics here presents a number of inconveniences since the identity of the actual programs using the tools is not known at the time of writing the tool, these tools cannot be tasks themselves. The Ada tasking model defines an asymmetric inter-task communication mechanism in which the identity of the callee must be known to the caller. In other words, to have true library tasks (where the identity of the callers/callees is not revealed),

ORIGINAL PAGE IS
OF POOR QUALITY

we need to introduce extra programs. For example, if we want to connect two library programs and run them in parallel, we have to do so through a third intermediary program. This is feasible because the identity of this third program is known to the other two. The fact that these units run in parallel is an implementation decision, which is best hidden inside the unit body (an added benefit).

The first alternative seems more effective, since we could use the full power of the language at compile time (e.g., type checking) and at run-time (at least the system can be tested on a uniprocessor environment), and it does not require any "special" features from the programming language (in fact, most available implementations will not even support multi-processor targets directly). The second alternative, however, may be more convenient and elegant, reflecting the real world situation (i.e., independent parallel programs each running on its own CPU), but requires a well-defined STANDARD distributed systems methodology.

5. CONCLUSIONS

Accommodating heterogeneity in a software development environment requires that the system be written for a number of different machines and be able to support numerous software packages associated with various operating and run-time systems. It is postulated that control of such system must be effected through **distributed kernels** operating on a local basis. The run-time system is best organized following the layered model provided that we are able to highlight:

- the relationship between the distributed and local operating systems
- the relationship between the different types of decisions made by the juxtaposition of the two control domains (i.e., local and global)
- the visibility necessary to effect the various implementation issues

Obviously, the APSE approach is the way to go, but perhaps it will need to be modified to resolve distributed computing issues such as:

- network transparency at the user level
- interprogram (internode) communication mechanism
- exception handling mechanisms encompassing distributed characteristics
- awareness of application objectives

ORIGINAL PAGE IS
OF POOR QUALITY

- fault tolerance strategy over the placement and updates of back-up copies of information

What we need here therefore is an extra layer, the DAPSE, in between the MAPSE and KAPSE. This will provide a standard interface for such a system support environment.

6. REFERENCES

Chevers,E. "NASA Space Station Software Requirements" (JSP, Jan. 1986)

Dixon "Open Forum on Space Station Software Issues" (NASA, Jonhson Space Center Houston, Texas, Feb. 1985)

KIT/KITIA "DoD Requirements and Design Criteria for the Common APSE Interface Set (CAIS) September 1985

KIT/KITIA "Military Standard Common APSE Interface Set (CAIS) Version 1.4 October 1984.

7. ACKNOWLEDGEMENT

The authors would like to thank Dr. Ann E. Reedy of Planning Research Corporation for discussion of many of the Standards and Unique Naming issues.

ORIGINAL PAGE IS
OF POOR QUALITY