Extending the Granularity of Representation and Control for the MIL-STD CAIS 1.0 Node Model

Kathy L. Rogers
Rockwell International
Space Station Systems Division

Introduction

The Common APSE (Ada[1] Program Support Environment) Interface Set (CAIS) [DoD85] node model provides an excellent baseline for interfaces in a single-host development environment (see Figure 1). To encompass the entire spectrum of computing, however, the CAIS model should be extended in four areas. It should provide the interface between the engineering workstation and the host system throughout the entire lifecycle of the system. It should provide a basis for communication and integration functions needed by distributed host environments. It should provide common interfaces for communication mechanisms to and among target processors. It should provide facilities for integration, validation, and verification of test beds extending to distributed systems on geographically separate processors with heterogeneous instruction set architectures (ISAs). This paper proposes additions to the PROCESS NODE model to extend the CAIS into these four areas.[2]

Rationale

The intent of the CAIS is to promote transportability and interoperability. The user interface should provide the same view of the system for a remote workstation connected through a network as for a directly connected terminal. Accessibility and finer granularity of the PROCESS NODE and QUEUE file information could provide processor performance measures during the design phase of the project, debugging information during the coding phase, and assessments of hardware and software changes during the

---

[1] Ada is a registered trademark of the U.S. Government, Ada Joint Program Office (AJPO).

[2] It is the intent of this paper to discuss some of the topics which were explicitly deferred in MIL-STD CAIS 1.0.

Extending the Granularity of Representation
and Control for the MIL-STD CAIS 1.0 Node Model

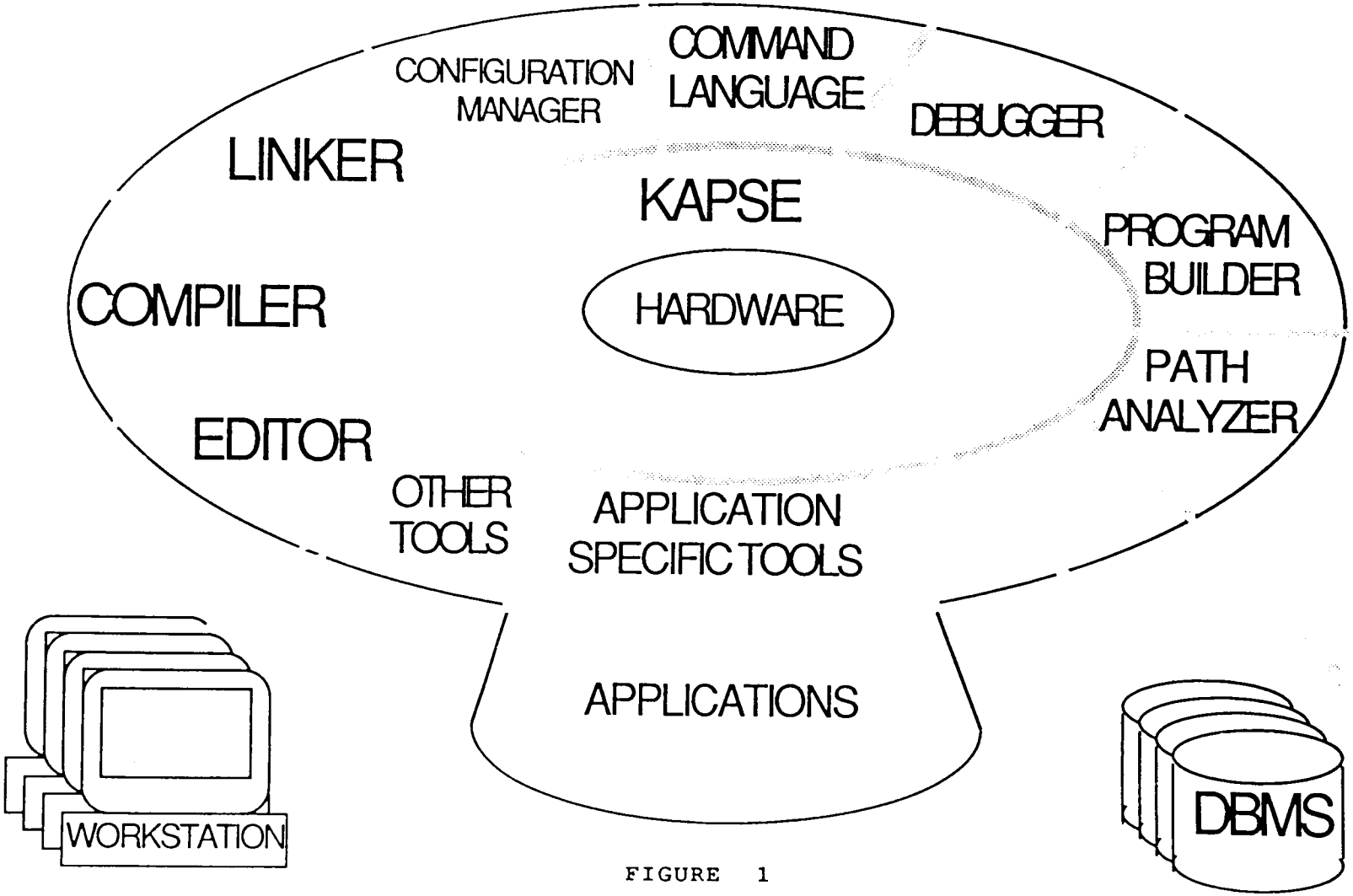

FIGURE 1

maintenance phase.

CAIS-provided code and data sharing could provide services (and entities) in a cost-effective manner to more than one application or user. To implement sharing, the node model must be able to manage data for dictionary driven processes, maintain version and revision information for library units, and provide security to maintain the integrity of the system. Data management requires information such as location, format, and access control of sharable resources. It might also extend to "knowledge" regarding the use of data, so that data may be relocated to facilitate convenient access. PROCESS NODES should be able to take advantage of code (such as common packages) that can be shared.[3]

The PROCESS NODE model should accommodate the communications necessary in a distributed environment. Five types of communication interfaces should be added to the current model: communication between parts of a process executing on separate processors, between processors (extending to processors with different ISAs), between the CAIS and the PROCESS (in both the host and the target environment), between different CAIS implementations, and among PROCESS NODES. In order to satisfy the Ada Language Reference Manual [LRM83] requirement that "several physical processors (may) implement a single logical processor"[4] effective information interchange is vital. Information must be communicated in an understandable format between heterogeneous ISAs. "Hooks" should be established so that individual elements of a test bed, as well as the integrated test bed, can be monitored. The CAIS should be extended to interact with other CAIS

---

[3] Multiple copies of packages, such as TEXT_IO, would be eliminated in favor of all processors at a site accessing the same copy. In a heterogeneous distributed environment, this can extend to shared copies of SYSTEM packages and STANDARD packages, if a common data representation scheme is used.

[4] Ada Language Reference Manual, Chapter 9, paragraph 5.

implementations.[5] When processes executing under the auspices of two different CAIS implementations interact and require CAIS services, a standard method should be used to determine which CAIS should be called. Interfacing to communication mechanisms, especially in a geographically separate system, is an important aspect of the CAIS.

Annotations for "non-functional"[6] directives could be handled by the PROCESS NODE model. These directives include desired degree of fault-tolerance, scheduling priority, desired level of status information, recovery processes, performance measures, special hardware requirements, and/or amount (and detail) of information to be promoted. Fault tolerance could be supported to ensure that sufficient resources are utilized to maintain the level of integrity required by the process. Scheduling of processes according to priorities should be considered; algorithms for serving processes according to their priorities could be provided in a straightforward manner. Directives stating the granularity of information required for a PROCESS (which determines the amount of overhead incurred) should be flexible.[7] Directives should also provide error recovery and rollback to the last "safe" state at a level of overhead which is appropriate for the PROCESS. Performance measures should be provided, especially for "time critical" processes which may need to be routed to a processor based on the speed and level of services available. The need to know the execution efficiency of processes on target processors is a major reason the CAIS services should be available in the target environment. In some configurations,

---

[5] Oberndorf, Patricia, Prototyping CAIS [Obern86].

[6] "Non-functional" is used here to denote constraints on functionality beyond those which are explicitly written into the code.

[7] For example, information pertaining to the current/last instruction or procedure executing might be requested. In the same way, the status of entities ranging from register values to values of user variables might also be requested.

security will b^ important; security directives should be provided and enforceable [LeGra86]. The CAIS can be continually extended by providing additional handlers to accommodate future "non-functional" directives.

The PROCESS NODE model should include capabilities to query and to negotiate with other nodes. Negotiation may be required in the case of a remote procedure (subprogram) call where the size of the parameters exceeds the capabilities of the receiving processor. Query and negotiation procedures could detect this problem and establish a piecewise transmission of data. Processes executing on processors with different ISAs could negotiate a standard data format for transmitting data. Query capabilities are vital for processes which have very specific processor needs. Query and negotiation capabilities should be provided to determine the optimal processor configuration to execute a process. Library management, in a system containing heterogeneous ISAs and specialized processors, creates demand for information such as version/revision, intended ISA, special processing needs or priorities, and other required support.[8] Check out, with locking mechanisms, must be maintained for library units. Security for the items being managed is also a concern. The level of access required to read or update information must be established, including altering access requirements after updates. Creation and maintenance of multiple copies must be addressed with respect to update[9] procedures.

Recommendations

The current CAIS node model should be enhanced in four ways. First, the PROCESS NODE state information should be more descriptive. Second, there should be a PROCESS NODE representation of the status of each

---

[8] Other support may include speed, space, and/or security requirements, etc.

[9] Update is being used here to encompass all modification functions, addition, modification, deletion, etc.

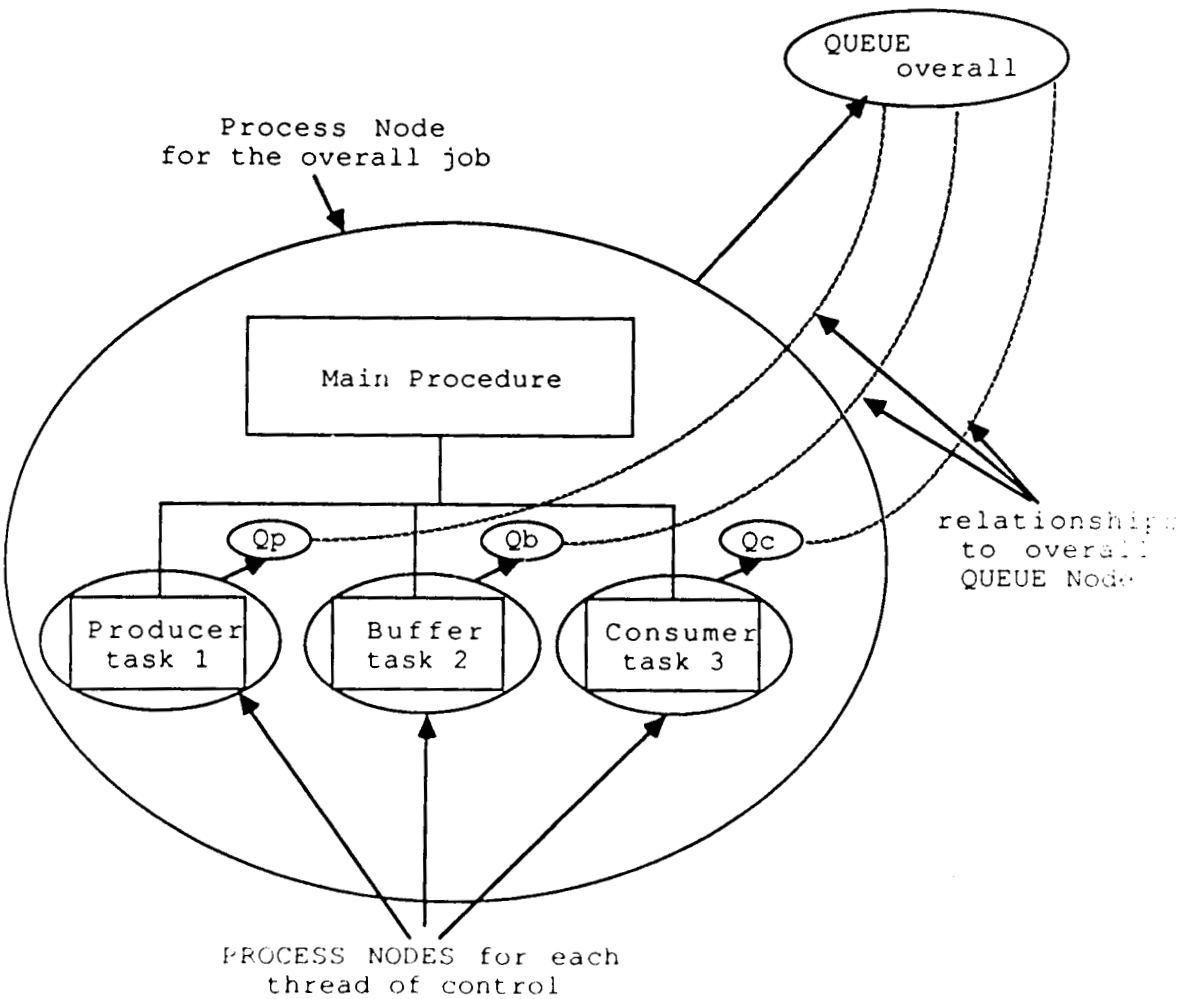

FIGURE 2
Snapshot of four PROCESS NODES

thread of control extending to any level of decomposition, and a QUEUE associated with each PROCESS NODE. Third, the QUEUE NODE[10] should be able to provide accessible status measures beyond those which are "hard coded" into the process. Finally, the QUEUE NODE model should provide capabilities to act on the information received.

As an example of the implications of the above recommendations, consider a PROCESS that spawns three subordinate tasks: a producer, a buffer, and a consumer. Figure 2 is a snapshot of the four PROCESS NODES; it represents the state of each thread of control currently executing on behalf of the "main" process. The overall job, as well as each subordinate task is depicted as a PROCESS NODE, with an associated QUEUE NODE. Each PROCESS NODE has several predefined attributes including: CURRENT_STATUS, PARAMETERS, and RESULTS. Other information, such as the logical name of the site where the process is executing, may also be available. Each QUEUE NODE representing one of the subordinate tasks has a relationship to the QUEUE NODE associated with the PROCESS NODE for the overall job. Note that when the subordinate tasks terminate, their respective PROCESS and QUEUE NODES cease to exist.

In order to augment the PROCESS NODE, the process states should consist of "meta-states" as well as "micro-states". In addition to the current "meta-states" READY, SUSPENDED, ABORTED, and TERMINATED, a new meta-state, RUNNING, should be added. The meta-states should also have micro-states to provide additional information. The READY meta-state should include the micro-states WAITING (for resources), COMPLETE (but not terminated), and BLOCKED (awaiting rendezvous). The TERMINATED meta-state should include the micro-states NORMAL and ABNORMAL.

To increase the granularity of the PROCESS model, the PROCESS NODE, which represents the overall job should also provide PROCESS NODES for each "thread of control". That is, a PROCESS NODE should be associated with every body of a subprogram, task, or package in a state of execution. All PROCESS NODES should be of the

---

[10] The term QUEUE NODE is used (rather than QUEUE FILE) in order to describe the QUEUE as an entity.

same form (complete with proposed extensions).[11] The PROCESS NODE for each thread of control should have an associated QUEUE NODE. Information from QUEUE NODES should be promotable upward to the QUEUE NODE representing the next higher level of decomposition, based on the amount of information required by the higher level PROCESS/QUEUE pair. In this way, the current CAIS PROCESS NODE is maintained on the job level, but is also decomposable to provide more specific information when needed.

Status information provided to the QUEUE[12] should be usable by other processes. In the current model, data, procedures, or tasks in one process cannot be directly referenced from another process.[13] QUEUE files are currently used as holders of PROCESS information.[14] The level of detail for status messages and the amount of overhead incurred, should be able to be specified. Other specifiable information includes the amount of information that should be promoted from a QUEUE NODE at any level to a QUEUE NODE related to a PROCESS at a higher level. Extensibility of the QUEUE NODE model can be provided by viewing the node as a database which can be queried by applications (or engineers). Additional information could be added to the database in the future, which could be utilized by processes which are aware of the enhancements. Status information generated independent of the process (or processor) is necessary in a distributed system, in the event of process (or processor) failure.

The QUEUE should be more than a passive information receptacle. It should be capable of being used to initiate procedures, such as recovery upon

---

[11] The PROCESS NODES should extend to any level of decomposition necessary.

[12] CAIS Rationale and Criteria document.

[13] MIL-STD CAIS 1.0 p. 14.

[14] Three types of QUEUE files are defined. The QUEUE files can operate in SOLO (write append, destructive read), COPY (SOLO QUEUE with initial contents), and MIMIC (dependent upon another QUEUE file) modes.

used to initiate procedures, such as recovery upon detection of a fault in the system. Facilities such as those necessary to terminate processes which are not performing correctly could also be provided. Early warning regarding process failure (rather than fault detection upon request for service) provides the calling process with a potentially greater number of recovery possibilities. Action in the event of failing processes is essential in environments which require fault tolerance, especially in unattended systems or in those systems where life and property depend on continuous, correct functioning of hardware and software.

Conclusion

The potentially long lifetime and large number of host development environments and target processor configurations, using Ada, require a CAIS that promotes transportability, interoperability, communication, and extensibility. The CAIS should provide a constant view (at an appropriate level of detail) of the supporting hardware and the APSE tools. This view should be provided to an engineer at a workstation, as well as to a secure, fault-tolerant distributed process. The CAIS should be extended to provide query and negotiation capabilities among nodes. It should include mechanisms for handling "non-functional" directives (in order to address the spectrum of processing complexity). It should also accommodate sharing code and data, as well as communication interfaces. These enhancements are necessary to accommodate the potential changes that will occur throughout the lifecycle of Ada applications. Some extensions to the CAIS model are necessary. Recommendations include maintaining more descriptive PROCESS state information; viewing the current PROCESS NODE model as a description at the overall job level (and providing PROCESS nodes for subprograms, tasks, and packages, while they possess a thread of control); viewing the QUEUE as a resource NODE rather than a logging file, and enhancing the QUEUE NODE to make it responsive to processing requirements. The proposed extensions to the CAIS model maintain the job level view of the original CAIS design and enhance it by providing decomposition to a finer level of granularity.

Extending the Granularity of Representation and Control for the MIL-STD CAIS 1.0 Node Model

The author gratefully acknowledges the contributions of the Joint NASA-JSC/UH-CL APSE Beta Test Site Team members, especially the expertise and encouragement of the research team's technical director, Dr. Charles W. McKay.

References

[DoD85] Department of Defense, Military Standard Common APSE Interface Set (CAIS), 31 January 1985.

[LeGra86] LeGrand, Sue and Richard Thall, The CAIS 2 Project, Softech Incorporated, Proceedings of the First International Conference on Ada Programming Language Applications for the NASA Space Station, June 1986.

[LRM83] Reference Manual for the Ada Programming Language MIL-STD 1815A, 1983.

[McKay84] McKay, Charles PhD., University of Houston at Clear Lake lecture notes, Fall 1984.

[Obern85] Oberndorf, Patricia, Prototyping CAIS, presented by Hal Hart at the Los Angeles SIGAda, February 1986.

[Roger86] Rogers, Patrick and Dr. Charles McKay, Distributing Program Entities in Ada, University of Houston at Clear Lake, High Technologies Laboratory, Proceedings of the First International Conference on Ada Programming Language Applications for the NASA Space Station, June 1986.