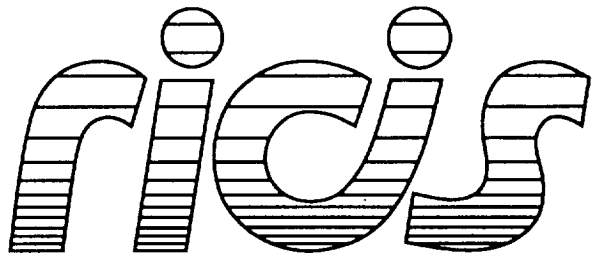
Ground Systems Development Environment (GSDE) Interface Requirements and Prototyping Plan

Victor E. Church
John Philips
Mitchell Bassman
C. Williams
Computer Sciences Corporation

September 1990

Cooperative Agreement NCC 9-16
Research Activity No. SE.34

NASA Johnson Space Center
Mission Operations Directorate
Space Station Ground Systems Division



Research Institute for Computing and Information Systems
University of Houston-Clear Lake

PRELIMINARY REVIEW REPORT

The RICIS Concept

The University of Houston-Clear Lake established the Research Institute for Computing and Information Systems (RICIS) in 1986 to encourage the NASA Johnson Space Center (JSC) and local industry to actively support research in the computing and information sciences. As part of this endeavor, UHCL 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 continuing cooperative agreement with UHCL 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 UHCL/RICIS mission is to conduct, coordinate, and disseminate research and professional level education in computing and information systems to serve the needs of the government, industry, community and academia. RICIS combines resources of UHCL and its gateway affiliates to research and develop materials, prototypes and publications on topics of mutual interest to its sponsors and researchers. Within UHCL, the mission is being implemented through interdisciplinary involvement of faculty and students from each of the four schools: Business and Public Administration, Education, Human Sciences and Humanities, and Natural and Applied Sciences. RICIS also collaborates with industry in a companion program. This program is focused on serving the research and advanced development needs of industry.

Moreover, UHCL established relationships with other universities and research organizations, having common research interests, to provide additional sources of expertise to conduct needed research. For example, UHCL has entered into a special partnership with Texas A&M University to help oversee RICIS research and education programs, while other research organizations are involved via the "gateway" concept.

A major role of RICIS then is to find the best match of sponsors, researchers and research objectives to advance knowledge in the computing and information sciences. RICIS, working jointly with its sponsors, 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 goals of UHCL, NASA/JSC and industry.

Ground Systems Development Environment (GSDE) Interface Requirements and Prototyping Plan

RICIS Preface

This research was conducted under auspices of the Research Institute for Computing and Information Systems by Computer Sciences Corporation in cooperation with the University of Houston-Clear Lake. The members of the research team for this report were: Victor E. Church, John Philips, Mitchell Bassman and C. Williams from CSC and Alfredo Perez-Davila from UHCL. Mr. Robert E. Coady was CSC program manager for this project during the initial phase. Later, Mr. Ray Hartenstein assumed the role of CSC program manager. Dr. Perez-Davila also served as RICIS research coordinator.

Funding was provided by the Mission Operations Directorate, NASA/JSC through Cooperative Agreement NCC 9-16 between the NASA Johnson Space Center and the University of Houston-Clear Lake. The NASA research coordinator for this activity was Thomas G. Price of the ADPE and Support Systems Office, Space Station Ground Systems Division, Mission Operations Directorate, NASA/JSC.

Ground Software Development Environment (GSDE) Interface Requirements and Prototyping Plan

Prepared for

The University of Houston-Clear Lake
Houston, Texas

by

Computer Sciences Corporation
System Sciences Division
Beltsville, Maryland
and
Special Projects Division
Falls Church, Virginia

under

Subcontract No. 075
RICIS Research Activity No. SE-34
NASA Cooperative Agreement NCC 9-16

September 1990

Prepared by:

Victor E Church 9/27/90

V. Church Date
M. Bassman
J. Philips
C. Williams

Quality Assurance:

M. Ellwood Date

Reviewed by:

Robert E. Coady 9/27/90

R. Coady Date

Approved by:

William Webb 9/27/90

W. Webb Date

Preface

This report is based partly on information provided by Ford Aerospace Corporation and by CAE-Link Corporation. It is also based on Computer Science Corporation's own understanding of the requirements placed on the Ground Software Development Environment (GSDE) for the development of Space Station Freedom ground software. As more information becomes available (i.e., as the ground system requirements are completed and the design efforts are begun), it will be factored into this report. Further information is particularly required from CAE-Link Corporation to complete this research effort.

This is a preliminary report on interfaces within the GSDE, together with a plan for prototyping software to support those interfaces. This report covers the major topics of interest, even though it is based on incomplete requirements information. An update to this report, based on further research, data collection, and prototype analysis, is planned for December 1990.

Abstract

This report describes the data collection and requirements analysis effort of the GSDE Interface Requirements study. It identifies potential problems in the interfaces among applications and processors in the heterogeneous systems that comprise the GSDE. It describes possible strategies for addressing those problems. It also identifies areas for further research and prototyping to demonstrate the capabilities and feasibility of those strategies and defines a plan for building the necessary software prototypes.

Table of Contents

Table of Contents (Continued)

List of Figures

List of Tables

Section 1 - Introduction

As part of the Space Station Freedom Program (SSFP), the Mission Operations Directorate (MOD) at the Johnson Space Center (JSC) is developing a Space Station Training Facility (SSTF) and a Space Station Control Center (SSCC). The software components of these systems will be developed in the Ground Software Development Environment (GSDE). The GSDE will serve as a common, high-productivity support environment for the development and configuration control of ground system software. It will make use of tools and procedures developed by the SSFP Software Support Environment (SSE) project. Both the SSTF and the SSCC will be developed using elements of this environment.

Computer Sciences Corporation (CSC) is studying ways to improve the effectiveness of the GSDE in supporting development for different target computer environments. This study is being performed for the Research Institute for Computing and Information Systems (RICIS) of the University of Houston-Clear Lake. The study includes identifying and documenting interface requirements and planning software prototypes to support those interfaces.

This report, *Ground Software Development Environment (GSDE) Interface Requirements and Prototyping Plan*, addresses the problems of constructing software in the GSDE for integration, test, and operation in the integration, verification, and test environments (IVTEs). It documents requirements for software to support the subject interfaces and describes a plan for prototyping that software.

1.1 Purpose of this Report

This report documents the data collection and problem analysis phases of the GSDE interface study. The interfaces of concern are those between the software development (or *host*) environment and the software execution (or *target*) environment. These interfaces include the following:

- Transfer of software from one environment to the other, including any necessary redevelopment (sometimes called rehosting, or porting)
- Communications of status information, test data, and test results between the two environments
- Configuration management of software across the boundary between the two environments.

These interfaces reflect the need for *cross-development* (i.e., development in one computer environment for execution in another) of ground system software within the GSDE. The

initial activity of this study is to identify obstacles to cross-development (interface problems) that are specific to the SSTF and SSCC projects. While many of the software construction details of those projects are unknown, due to the early state of the design of those two systems and to pending procurements, it is possible to identify and report on some of the anticipated problems. At the same time, the study team has identified approaches to resolving or reducing those problems.

This report also describes plans for the prototyping effort of the study. It describes procedures, operations, and interface problems that can be addressed and investigated with prototypes. The areas of investigation include software development operations in the complex of computers and workstations designated the Ground Systems/Software Production Facility (GS/SPF), GS/SPF to IVTE interfaces, and methods of achieving apparent functional equivalence between host and target systems.

This report will serve as input for the evolutionary development of the GSDE. It will identify requirements for moving software between the GS/SPF and the IVTE for compilation, testing, configuration management, and operations. It is intended to aid in developing the operational plans for use of the GSDE by the SSTF and SSCC contractors, as well as in acquiring the necessary tools and equipment for cost-effective software engineering.

1.2 Scope and Organization of this Report

The interface requirements addressed in this report include the following:

- Operational flow of software between elements of the GS/SPF and the appropriate IVTE, e.g., moving source code to an IVTE for compilation and test
- Distributed configuration management (CM), during implementation and after delivery
- Interface mechanisms (protocols) used by ground system software for communications within its execution environment (more specifically, data interoperability across disparate architectures)
- Rehost and test implications of differences between resources (e.g., specialized hardware components) available in the GS/SPF environment and in the IVTEs
- Requirements for specific tools and/or devices in the GSDE (to simulate or replicate elements of the IVTE in the GS/SPF).

This report addresses the interface problems that result from separating the development and execution functions on different computers. Those problems are, in effect,

requirements levied on the GSDE for specific elements of support. The primary focus of this analysis is the requirements placed on the GSDE by the SSTF and SSCC projects.

Following this overview section, Section 2 provides a detailed analysis of the problem, with references to similar types of problems encountered in other National Aeronautics and Space Administration (NASA) ground system development efforts.

Section 3 describes the development process suggested for software engineering in the GSDE. It discusses the requirements for CM, software porting and remote compilation, and integration and test.

Section 4 discusses ways that the host environment can be made functionally similar to the target environment. Requirements for a virtual environment are discussed, as are tools and devices used to simulate the target.

Section 5 describes the prototyping effort that is planned to demonstrate the workability of software to support the GSDE interfaces and to assist in further requirements clarification. Section 6 describes resources necessary for the performance of the prototyping effort. Section 7 presents the recommendations from this phase of the research effort.

1.3 Statement of the Problem

The Mission Operations Directorate at JSC is responsible for the development of ground support computer systems, the SSTF and the SSCC, for the Space Station Freedom Program. The software in these systems is being developed in the Ground Software Development Environment, on a complex of computers and workstations designated the GS/SPF. The GS/SPF provides resources that are part of the SSFP Software Support Environment (SSE). The GS/SPF includes an Amdahl mainframe, several Rational R1000 Model 300S Ada development computers, and a local area network (LAN) with various workstations (Apollo, MS DOS-compatible, and Apple Macintosh at a minimum) and some special-purpose devices attached. This is referred to as the *host* environment.

Note

The terms *host computer* and *host environment* in this report refer to the computers on which development is hosted. All of the computers in the GS/SPF--not only the mainframes--are considered host computers.

The *target* environments for this ground system software will be composed of computers, workstations, and special-purpose devices that differ from the corresponding elements in

the GS/SPF environment. Software will be developed in the host environment and transferred to the target for integration and system testing. The differences between host and target will force some transformation and even redevelopment of code. The separation of functions will also require mechanisms for communications and integration between the host and target environments. The fact that both environments will be heterogeneous, distributed systems further complicates the problem.

This study task (and the related prototyping effort) focuses on the *interfaces* between the host and target environments. Those interfaces include communications between host and target, actual transfer of files and command lists, and testing on the target that is orchestrated from the host. The goal of the effort is to find or develop mechanisms of GS/SPF-to-IVTE interfacing that will minimize the cost of rehosting software developed in the GS/SPF.

1.4 Related Documents and References

Campbell, I., "Standardization, Availability and Use of PCTE", *Information and Software Technology*, Vol. 29:8, October 1987

Campbell, I., "Emeraude Portable Common Tool Environment", *Information and Software Technology*, Vol. 30:4, May 1988

Federal Information Processing Standards Publication 151, *Portable Operating System Interface Standards (POSIX)*

Gallo, F., R. Minot, and I. Thomas, "The Object Management System of PCTE as a Software Engineering Database Management System", *ACM SIGPLAN Notices*, Vol. 22:1, January 1987

Johnson Space Center/T. Price, Ground Software Development Environment, April 1990 (briefing)

Liu, L-C. and E. Horowitz, "Object Database Support for a Software Project Management System", *ACM SIGPLAN Notices*, Vol. 24:2, February 1989

McKay, C., "Portable Common Execution Environment (PCEE)", UHCL Report

Penedo, M., "Prototyping a Master Database for Software Engineering Environments", *ACM SIGPLAN Notices*, Vol. 22:1, January 1987

Schantz, R., *et al*, "Resource Management in the Cronus Distributed Operating System" (abstract and bibliography), *ACM Computer Communications Review*, Vol. 17:5, August 1987

Stumm, M., "Strategies for Decentralized Resource Management", *ACM Computer Communications Review*, Vol. 17:5, August 1987

Thomas, I., "The PCTE Initiative and the PACT Project", ACM *Software Engineering Notes*, Vol. 13:4, October 1988

Vinter, S. "Integrated Distributed Computing Using Heterogeneous Systems", *Signal*, June 1989

1.5 Glossary

AAS	Advanced Automation System
A/D	analog to digital
AADS	Automated Attitude Determination System
Ada	Ada programming language; Ada is a registered trademark of the US Government, Ada Joint Program Office
APSE	Ada Programming Support Environment
CAIS-A	Common Ada Interface Set-A
CM	configuration management
CMVC	Component Management and Version Control system
COTS	commercial, off-the-shelf (i.e., commercially available hardware or software products not requiring SSFP-specific development
Cronus	distributed network operating system, developed at Rome Air Development Center
CSC	Computer Sciences Corporation
D/A	digital to analog
DBMS	database management system
DEC	Digital Equipment Corporation
DMS	Data Management System

DSDM	Digital Systems Development Methodology, a trademark of the Computer Sciences Corporation
DTIA	Distributed Tool Integration Architecture
FAA	Federal Aviation Administration
FAC	Ford Aerospace Corporation
GERM	Generalized Entity-Relationship Model
GESS	Graphics Executive Support System
GFE	government furnished equipment
GS/SPF	Ground Systems Software Production Facility
GSDE	Ground Software Development Environment
GSFC	Goddard Space Flight Center
IBM	International Business Machines
IVTE	Integration, Verification, Test Environment
JSC	Lyndon B. Johnson Space Center
LAN	local area network
MCC	Mission Control Center
MIPS	millions of instructions per second
MOD	Mission Operations Directorate
MSC	Mission Support Contract
NASA	National Aeronautics and Space Administration
NDI	non-developed item
OS	operating system
PCEE	Portable Common Execution Environment
PCIS	Portable Common Interface Set
PCTE	Portable Common Tool Environment

POSIX	Portable Operating System Interface (standard)
QA	quality assurance
RICIS	Research Institute for Computing and Information Systems
RXI	Rational X-Windows Interface
SDP	Standard Data Processor
SIB	Simulation Interface Buffer
SMM	Solar Maximum Mission
SPF	Software Production Facility
SSCC	Space Station Control Center
SSE	Software Support Environment
SSFP	Space Station Freedom Program
SSTF	Space Station Training Facility
TBU	Target Build Utility
TCP/IP	Transmission Control Protocol/Internet Protocol
TSC	Training Support Contract
UHCL	University of Houston-Clear Lake
VME	virtual machine environment
WAN	wide area network

Section 2 - Analysis of the Problem

As noted in Section 1.3, the problem addressed here is the development-to-execution interfaces within the GSDE. The analysis presented in this section characterizes the interfaces involved and identifies strategies for supporting those interfaces. The resolution strategies and derived requirements for GSDE interface support are investigated in Sections 3 and 4.

2.1 Ground Software Development Context

There are two major obstacles to the use of a single environment throughout the software life cycle. First, the target systems (the SSTF and the SSCC) will include computers that are not represented in the GS/SPF. Second, the target systems will include a significant amount of code that is reused from previous systems, for which the standard SPF does not provide compilers and tools. Because of these two obstacles, integration and testing of the ground software will require the use of target facilities that are distinct from the GS/SPF. There will be one such facility, called an IVTE, for each of the ground systems. Figure 2-1 shows this configuration.

Because of execution, interface, and performance requirements, SSTF and SSCC ground software will operate on different types of computers from those used for development. In particular, the major development platform will be Rational R1000 computers, which are not suitable for operations. (The Rational computer systems are optimized for Ada code development and are not cost-effective for general data processing operations). Other differences will be defined as system designs and hardware procurements are completed.

Rehosting generally requires changes and modifications that increase the cost of ownership of the software. In extreme cases, a substantial amount of development (or redevelopment) occurs in the target environment. This reduces the cost effectiveness of the entire process. The host environment (e.g., the GS/SPF) is typically far more productive (due to factors like availability, power, and tools complement) than the target.

Integration and testing, on the other hand, can be very expensive if performed on the host because of the cost of simulating or emulating the target environment in the host environment. An example is the use of Data Management System (DMS) kits and the Simulation Interface Buffer (SIB) for development of SSFP flight software.

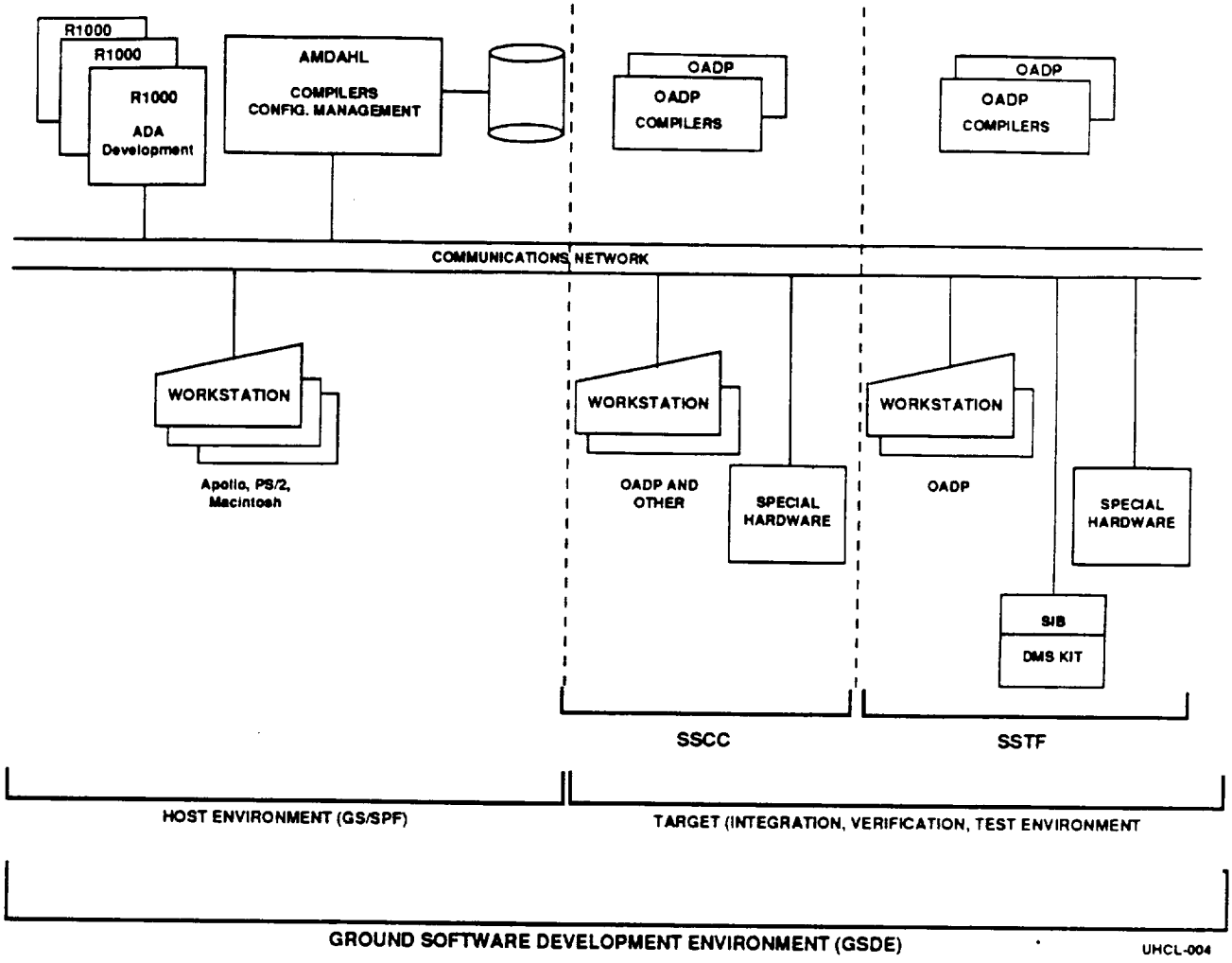


Figure 2-1. Ground Software Development Environment

The basic problem is to find a cost-effective balance between using the high-fidelity target environment and the high-productivity host environment. The study described in this report involves identifying strategies and techniques for optimizing the use of the GS/SPF in developing software for the SSTF and the SSCC. By supporting test and integration on the host and reducing the use of the target environments, the overall cost of development can be reduced.

In general, several basic strategies can be used to achieve this balance of development between host and target. For example, the host environment can be enhanced, making it more attractive to developers. Tools (e.g., cross-compilers and target-machine simulators) can be used to perform simulated target-based testing. The target environment can be stripped of development tools (e.g., editors, debuggers) to make it less attractive. Virtual-machine interfaces (e.g., POSIX) can be installed on both target and host environments to minimize the differences. Some of these strategies are provided by the SSFP SSE and are already in place in the GSDE.

Software development for the SSTF and the SSCC will take place in the GSDE on the GS/SPF. The developers of these ground systems, the Mission Support Contractor (MSC) for the SSCC and the Training Support Contractor (TSC) for the SSTF, will each have components of the GS/SPF located within their facilities and dedicated to their use. Administration of the GS/SPF and CM of ground system software will be centralized at JSC. Figure 2-2 shows this basic configuration.

By using the GS/SPF, the ground systems software developers can take advantage of the tools and facilities that have been collected and created to boost software productivity. The GS/SPF provides tools and database support for many aspects of software development, including the following:

- Requirements development and tracking
- System and software design
- Schedule and performance management
- Configuration management
- Code development (for Ada code)
- Test and integration
- Documentation.

Using this environment, developers will be able to capitalize on the availability of software tools and procedures developed for all of the SSFP. Some of the tools, particularly Cadre Teamwork and Rational R1000 computers, have established excellent track records for improving the productivity of users and the quality of products. The GS/SPF will provide users with an extensive set of resources, including an Amdahl, several Rationals, many workstations of several types (Apollo, MS-DOS or OS/2 compatible, Apple Macintosh, possible others), and network support.

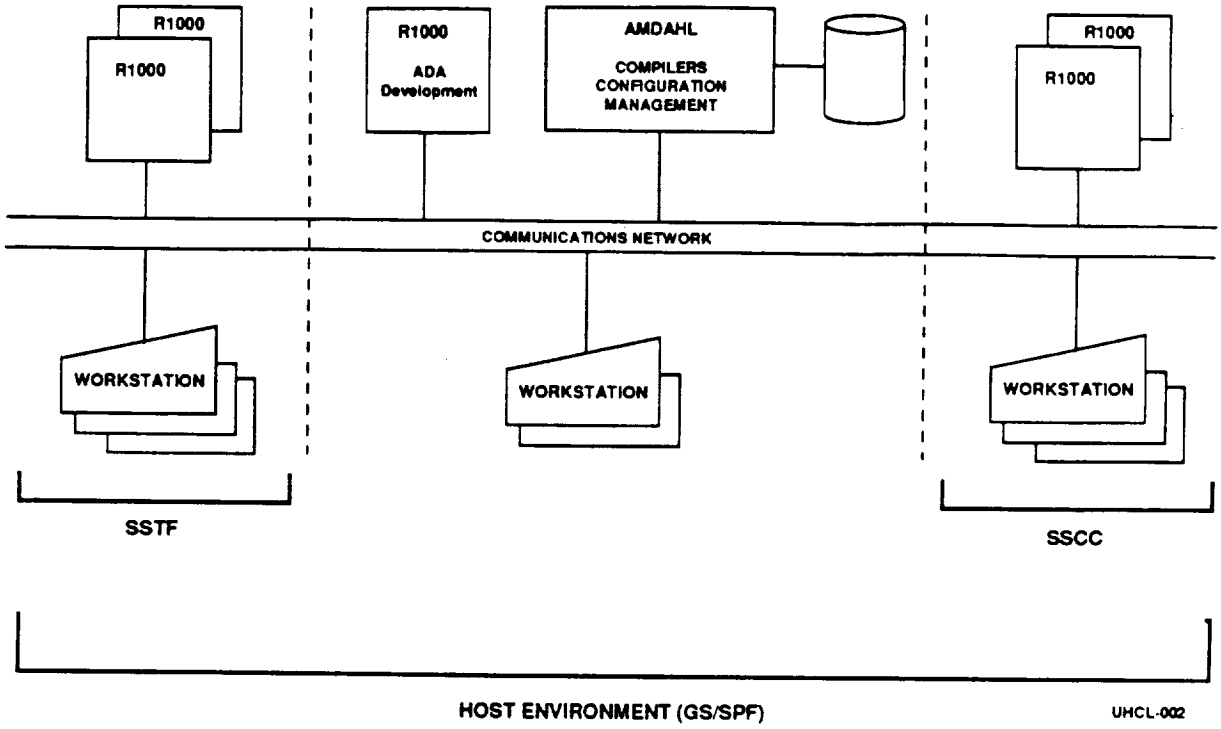


Figure 2-2. Ground Systems/Software Production Facility

2.2 Host-to-Target Development Requirements

The most common strategy for developing software is to use the same computer(s) for development and for operations. (This strategy, called *self-targeting*, also includes using compatible computers, such as different models of VAX computers). This strategy requires that the target computer provide adequate development tools such as compilers, linkers, and debuggers. There are few problems of incompatibility between host and target machines because they are the same, or at least functionally equivalent. (Even with this strategy, differences in processing speed or in peripheral complement could cause problems, but these are generally minor and easily solved.)

2.2.1 Cross-Development Justification

There are several reasons why functionally different machines will be used for development and operations. The requirements of the two different uses are generally quite different: the target computers are selected to meet operational requirements (such as fast processing, real-time response, small size, or flight qualification), while the host computers are chosen to minimize the cost of development. For SSFP ground software, it is not possible or practical to find one system that is optimal for both sets of requirements.

Additional development costs are incurred whenever software must be ported from one computer to a different one. In this instance, the difference in requirements (between development and operations) is large enough to justify the cost of transitioning from host to target. Accordingly, different computers have been selected.

The GS/SPF is based on the SSFP SSE and includes a very extensive set of procedures, tools, and standards for software development. The support provided by the SPF extends from requirements definition and analysis to configuration management of developed code. The SPF includes documentation and project management support, an extensive training and support system, and ongoing development of new tools and capabilities. This level of support would be impossible to provide for each target environment.

The GS/SPF also includes Rational development systems, which are highly effective Ada development tools that are not suited for use as targets in real-time applications. It includes other tools and capabilities (such as Teamwork and Interleaf) that are not suited to either the SSTF or the SSCC target environment. Both target environments require real-time distributed processing and involve specialized hardware that is not part of the standard SPF complement.

To support integration and test, the special-purpose hardware and target system computers will be replicated in the IVTEs. For both cost and productivity reasons, the IVTEs will be restricted to use only for integration verification and test. All

development will be performed in the GS/SPF, and developed software will be ported to the IVTEs.

2.2.2 Models of Host-Target Development

The cost of transitioning is dependent on the support provided to the developer for masking or accommodating the functional disparities. There are several models for supporting this transition, with varying implications and cost factors.

The simplest model (see figure 2-3), often used in avionics development, uses a bare machine for the target and performs all code implementation on the host system. Cross-compilers and loaders are provided by the host computer. Executable images are downloaded to the target; testing is often performed under control of the host system. The target computer may be simulated on the host system so that functional testing can be performed without using the actual target. Such simulators generally provide more instrumentation (e.g., symbolic debuggers and breakpoints) than the actual target computer provides.

This model will apply to special-purpose hardware such as communications processors, which be identified as the SSTF and SSCC design efforts proceed.

A more complex model (figure 2-4) involves a target processor with its own operating system and system-level applications (e.g., a DBMS or an inference engine). The host environment is used to create and unit-test code, but the target's compilers and loaders are used to create executable images for integration test and operations. Some of the *elements* of the target (such as external interfaces) may be simulated on the host to support unit-testing, but the entire target environment cannot be realistically simulated.

This model may be appropriate when the target is adequately supported and is powerful enough to support compilation and test. This model makes effective use of the host environment during early stages of implementation but may get cumbersome during later stages of the life cycle. Changes to applications (e.g., replacing stubs with actual system calls) are often needed when transitioning from host to target. Those changes make it difficult even to unit-test modifications on the host system.

The mainframes and workstations in the IVTEs fit this model. The specifics of those machines are dependent on the outcome of the OADP procurement.

A third model (figure 2-5) involves the use of "virtual machine" software to hide the differences between host and target. The virtual machine environment (VME) software is itself hardware specific. It resides on both host and target systems, masking the differences. This model requires that functionally equivalent compilers be available on both systems. As with other models, there may be some simulation of the target on the host, but the VME provides most of the interface.

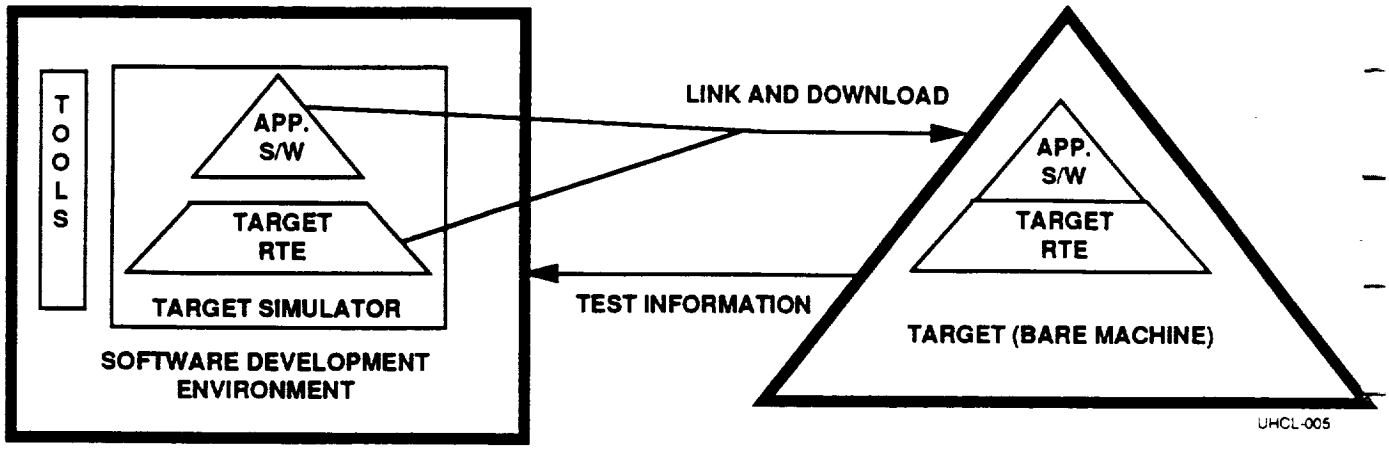
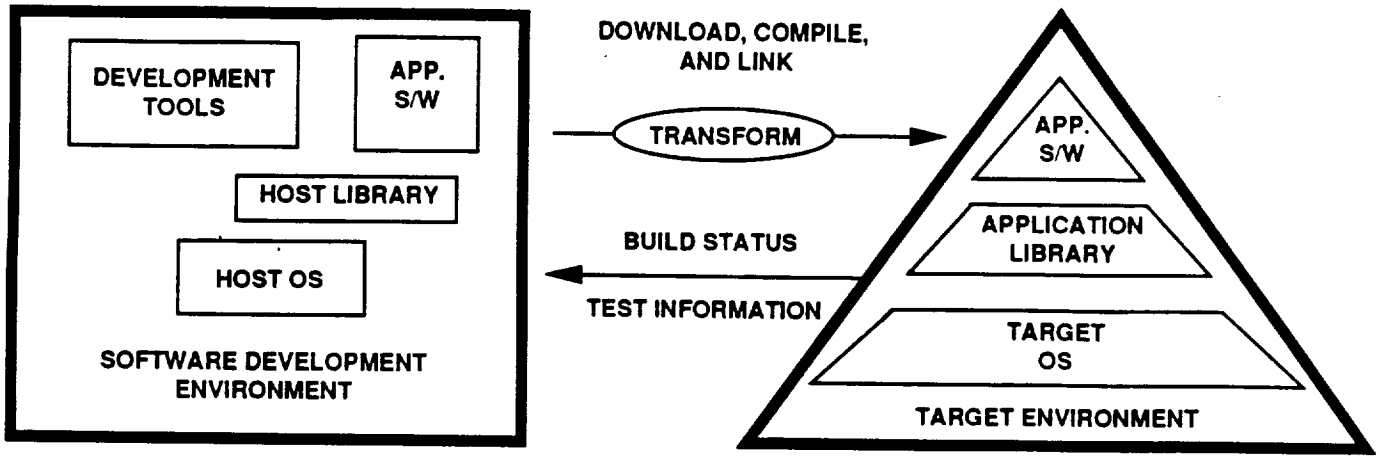
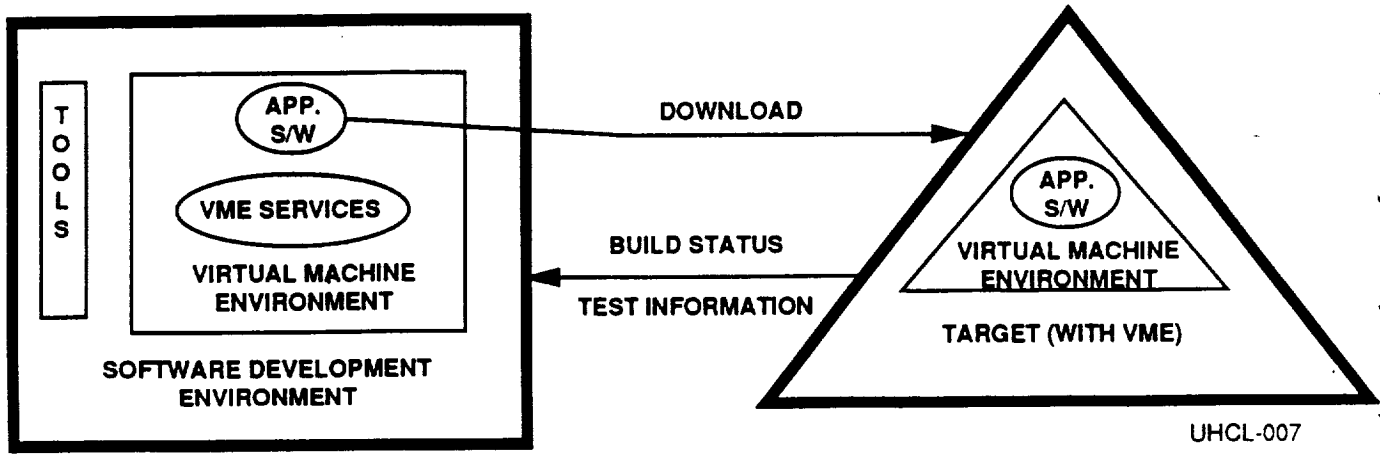


Figure 2-3. Bare-Machine Targeting



KEY:
APP. S/W: APPLICATION SOFTWARE

Figure 2-4. Peer-Machine Targeting



KEY:
APP. S/W: APPLICATION SOFTWARE
VME: VIRTURAL MACHINE ENVIRONMENT

Figure 2-5. Virtual Machine Targeting

This model requires that VME implementations exist or be developed for all host and target computers. The VME may impose a performance penalty on the target, which may be unacceptable. The most common example of this approach is the use of Unix as a platform-independent operating system; the POSIX standard is intended to make the Unix model more consistent. Another example is Cronus, which provides an interface definition and supporting software for peer-to-peer and application-to-data interfaces.

It may be possible to apply this model to some or all of the mainframes and workstations in the IVTEs. Further study of requirements and available commercial environments, as well as prototyping of some interfaces, will be used to resolve this issue. Sections 2.2.2 and 4.3 provide further discussion of this issue.

2.2.3 Obstacles to Host-Target Development

As previously noted, there are problems associated with developing code on one type of computer for operation on another. Each of the development models described in subsection 2.2.2 has a different set of problems, advantages, and cost factors. In brief, those problems include the following incompatibilities:

- Object-code and addressing
- Low-level operating system (OS) facilities (e.g., interrupts)
- Peripheral hardware (I/O, network, etc.)
- Resources available (e.g., memory)
- Timing (performance and clock services)
- High-level (system call) OS services
- Library routines (standard libraries)
- Compilers

The characterization and mitigation of each of these problems, with respect to the models defined above, is discussed in the following subsections.

Object code and addressing. The object code for different processors is, of course, different. Addressing schemes may also be different in subtler ways; logically equivalent data structures may exhibit differences in performance due to segmentation and memory architecture. Most such problems are solved with the used of high-order languages, but there can be differences (e.g., in hardware arithmetic) that have subtle impacts.

The bare-machine model may address the problem by simulating in detail the performance of the target computer. The mixed-machine and virtual-machine models must rely on precise format specifications and careful record-keeping to mitigate the problem. Ada provides some support for detailed specification of formats, and some virtual machines (e.g., Cronus) provide standard formats and conversion routines.

Low-level OS facilities. Many operating systems have program information stored in specific locations, such as program status word, job control and file control blocks, or video display memory. These facilities generally cannot be tested except on the target machine. The software applications that use these facilities can be tested if they are simulated on the host. This is often the case on bare-machine model systems.

Peripheral hardware. Control of hardware devices that will attach to the target computer presents a considerable difficulty. These hardware devices include analog-to-digital (A/D) and digital-to-analog (D/A) converters, communications interfaces such as Ethernet transceivers and modems, recording devices, display devices, switches, etc. In some instances, the device may be able to connect directly to the host computer for integration and testing. In other cases, there may be equivalent devices for both target and host. In still other cases, it may be necessary either to simulate the device on the host or to delay testing until transition to the target. (The last option does nothing to reduce risk; it simply permits testing of other elements of the system).

Resource availability. Even if the host and target are otherwise compatible, differences in resources may pose an obstacle to integration and test. Main memory size is an obvious example; development machines often require and have more available main memory than do targets. Disk space and speed and coprocessors are other examples of critical resources. These may not affect the logical construction of application programs but may impact the integration and test of applications. (For example, a very real concern for Space Station flight software is that applications that work in a development environment may not fit in the memory available on a Standard Data Processor.) Resource limits such as these must be identified for each host-target pair and may be different for different applications depending on requirements. Once identified, checks must be made to ensure that application software uses only resources that will be available on the target.

Timing considerations. In a real-time environment, the availability of support for timing and clock services is critical. This is one area in which the target is more likely than the host to have the necessary facilities. Interactions among task elements may be difficult to test in the host environment if the target's real-time environment cannot be accurately simulated. Clock and timer services provided by the target may be unavailable in the host, or may be available only to the system supervisor. Planning is necessary to ensure that timing considerations are addressed during development. Benchmarking of the target may be needed to establish simulation parameters on the host.

High-level OS services. The services that are readily accessible to high-order languages, such as file operations and interprocess communications, are more or less platform specific. Generally, they are straightforward and easily simulated. If the operating system is standardized (e.g., POSIX-compliant) the simulation is made even simpler. High-level system services present an obstacle to cross-development, but one that is

easily addressed (e.g., by writing interface routines to translate from one context to another).

Library-based services. In addition to the services provided by the platform OS, there are support libraries for specific compilers, data base systems, network services, etc. These libraries are often duplicated on both host and target, but the duplications are of variable fidelity. Experience with Ada support libraries on different systems, for example, has shown that careful testing is needed to ensure functional equivalence in operational settings. Past history and good recordkeeping are valuable in assessing the magnitude of this obstacle to cross-development.

Compilers. The differences among various CPUs and memory systems are largely camouflaged by the use of common high-order languages. However, compilers themselves can introduce differences, especially when produced by different vendors. The Ada compiler validation process tends to reduce such problems for Ada, but does not eliminate them. The Federal Aviation Administration's (FAA) Advanced Automation System (AAS) has identified many compiler differences, as noted in section 2.2.4. As with library service differences, experience and recordkeeping are important to addressing the problem.

2.2.4 Experiences with Cross-Development

NASA has considerable experience in cross-development, using all three models of host-to-target development. Flight computers are generally too small to support compilers and development tools, and are typically supported with host development systems. This falls into the "bare-machine" model. Typically, the software for an onboard computer is written and cross-compiled or cross-assembled on a general-purpose computer (such as a VAX) and downloaded to the target for testing. The target computer generally remains connected to the host for test control, test data, and perhaps simulated data.

Autonomous Attitude Determination System. An example of this class of cross-development was the Autonomous Attitude Determination System (AADS). The target was a 16-bit Intel processor (not flight-qualified); the host was a VAX 780. The AADS system was first constructed on the VAX and functionally tested, then cross-compiled for the target and downloaded. Support software developed on the host included a full data simulation system, a test harness, and a ground command simulator. The simulation system served the purpose of the simulation interface buffer (SIB), providing all of the signals that the AADS would receive if it were actually on board a satellite. (It was generalized from dynamics simulators used for testing mission-specific onboard computers). The test harness provide the controls for transmitting data and retrieving results, with facilities for adjusting the time step and for restarting the simulation. The ground command simulator provided the operator interface (distinct from the test operator) to exercise the AADS.

The bulk of development was performed on the host, with testing being performed on the target.

Solar Maximum Mission Attitude Determination. NASA's operational ground system computers are heavily committed to operations in some instances, and cannot provide adequate resources for development of new systems or maintenance of current systems. Since many of these systems involve highly specialized equipment (operator consoles, orbit track displays, etc.), development must proceed with limited access to parts of the system. This was the case during the development of the Solar Maximum Mission (SMM) attitude ground software at Goddard Space Flight Center.

The operational system used a specialized console interface program called GESS (Graphics Executive Support System), which was closely tied to the display devices and operating system of the operations computer system. To support development offline, a PDP-11/70 computer was used to emulate the IBM 360/95 target environment. The differences between Digital Equipment Corporation (DEC) and IBM FORTRAN were mapped out, and many utility routines that had been developed for the target were rewritten for the host. The GESS program was simulated on the host, using VT-100 terminals as substitutes for 2250 displays.

After development and functional testing, the code was ported to the target. Changes were needed where the GESS simulation was not a true copy of the target version, and where language and machine architecture differences caused problems. Final integration and redevelopment were performed on the target, which did have a full complement of development tools. The developed code was never returned to the host after being ported to the target. (This is typical of traditional cross-development practices).

Advanced Automation System. Another example of cross-development (although not a NASA experience) involves developing software on Rational machines for operation on IBM mainframes. The Advanced Automation System being built for the FAA is currently being developed in Ada. Rational R1000 development computers are used for initial code design and development. The code is functionally tested and then recompiled on the target with the Telesoft Ada compiler. The system has both real-time and fault-tolerant elements, and requires extensive integration and system testing.

It has been found that Ada code developed on the Rational for the IBM system does not perform exactly the same on host and target. The differences between the Rational version of the library and the Telesoft library are enough to cause problems in this particular environment. Because of the superior performance of the Rational compiler, the developers make every effort to complete development on those computers; but integration must still be performed on the target. When software is moved back to the Rational for software fixes, configuration management is compromised.

2.2.5 Implications for Different Models

With the bare-machine model, the host computer must provide extensive simulation and cross-development facilities. As with the AADS example, the target computer is only a small portion of the total integration environment. The test framework and whatever interface support is required must be built in the host to enable the tests to be performed. If the target itself can be simulated in the host (a common condition with many small bare computers), the development process can be streamlined very well. As more of the target environment is replicated in the host, the cost and risk of cross-development is greatly reduced. Of course, the cost of reproducing the target environment must be evaluated.

When the target machine is of the same class as the development computer (i.e., both are general purpose systems, the peer-machine model), it is not practical to replicate one computer system in another. Compilers and standard libraries can minimize the differences, but (as with the SMM example) there may also be a need to duplicate special-purpose hardware or software to perform even unit-testing. The higher productivity of the host environment is presumed to more than offset the cost of redevelopment after porting. (In the SMM example, the benefit was the much greater availability of the host system).

One risk of using this approach is that, if changes are made to specialized software on the target, those changes must also be made to the host emulation. This increases the cost of sustaining engineering.

Another risk is that anomalies may be reported in the developed code that are in fact due to errors in emulation. Experience with using simulators to test operational software (a common practice) shows that *apparent* errors in the operational software are often due to *simulator* bugs instead of errors in the operational code. Other problem reports can stem from less-than-complete fidelity in the simulation.

The SMM example actually demonstrated some aspects of the Virtual Machine Environment (VME) approach to cross-development. The GESS software on the target was created to isolate operational software from some characteristics and machine dependencies on the target. Instead of developing software for the actual display devices and memory structures, code was developed for the GESS virtual machine. By replicating that capability on the host, it was possible to develop "machine-independent" code.

2.3 Strategies for Cost-Effective Development

The incentive for maximizing the use of the GS/SPF--and delaying as long as possible the shift to the IVTE--is that support for dozens or hundreds of developers can be

provided more effectively and at less cost in the SPF. The IVTE is expensive and is not conducive to high productivity. The GS/SPF can more readily accommodate large numbers of users, more and better tools, and better integration of tools. The SPF can also be expanded more easily than can the IVTEs. The problem is to mask and contain the inevitable cross-development incompatibilities so that GS/SPF-based development can be effective as well as productive.

Two basic, mutually reinforcing strategies can be followed to maximize use of the GS/SPF, and thereby reduce the overall cost of ground software development. First, tools and procedures can be put in place to simplify the cross-development process and encourage development in the host environment. Second, tools and simulations can be installed in the host (and possibly in the target) to minimize the differences between host and target. The first strategy involves the process of cross-development; the second strategy directly affects the design and implementation of application software.

Both of these strategies must be tailored to the systems being developed. The development process strategy must encompass the use of the Amdahl and Rationals as well as various network-interfaced workstations. The applications strategy must accommodate the specific requirements of the two systems as well as the expected use of existing code.

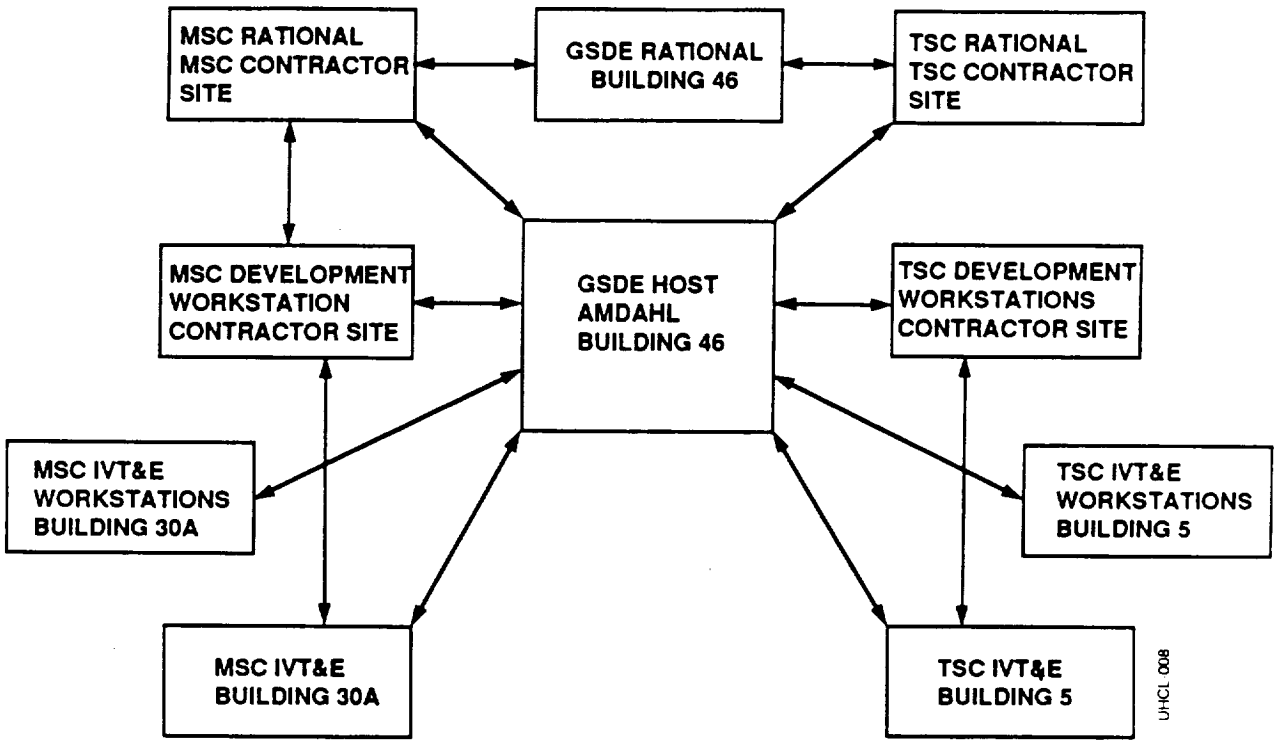
Section 2.4 discusses the characteristics of the various development and operational environments and describes the process of assembling the requirements to tailor these strategies. Sections 3 and 4 discuss the strategies themselves.

2.4 Ground System Development

This section describes the different computing environments that were reviewed to derive interface requirements for the GSDE.

2.4.1 Ground Support Development Environment

The GSDE system is a distributed system with users of the system networked to a central facility located in Building 46 at JSC. Each of the users have development computers within other facilities. Figure 2-6 shows the overall architectural layout of the GSDE, with logical communications interfaces shown by the arrows. This layout includes an overview of the portions of the GSDE that are in the facilities of the TSC and MSC contractors.



KEY:
ARROWS SHOW LOGICAL (NOT PHYSICAL) COMMUNICATIONS PATHS

Figure 2-6. GSDE Communications Architecture

Since the GDSE is the central portion of the development environment, it will be the central repository for all developed software. It will maintain a centralized configuration control accounting and be a centralized reporting point for software testing reports. Figure 2-7 depicts the functions performed by the different elements within the GDSE.

Software developed within the GS/SPF will be ported to IVTEs (one for the SSCC and one for the SSTF). This interface (from GS/SPF to IVTE) is critical to the success of the planned use of the GS/SPF as the central software development facility for developing software for the SSCC and the SSTF. This GS/SPF to IVTE interface must be transparent to the developed software to reduce the need for development activities on the IVTE (target) machines.

The target machines are not yet specified, except in general terms (e.g., performance estimated in millions of instructions per second (MIPS), support for specific programming languages). Based on anticipated requirements, it seems probable that the target systems will include:

- Mainframe computers and midrange computers from the set of IBM 370 or 390 series computers, DEC VAX computers, or systems from Control Data Corporation or from Unisys Corporation.
- Workstations based on the Unix operating system, such as Sun (Sun OS), Apollo (Domain), IBM 6000 (AIX), or DEC VAXstation (Ultrix).
- Masscomp workstations
- Special purpose hardware (unique to each IVTE)

Once the computer systems have been selected as a result of the OADP procurement, the requirements for interfaces to the target systems will be specified in detail.

2.4.2 Space Station Training Facility

Software development for the SSTF will be a distributed process. The bulk of the software will be developed on workstations (Apollo) and Rational development hardware and software located in the Link Flight Simulation Building. These development machines will be connected to the GSDE host located in Building 46 at JSC. Figure 2-8 illustrates this configuration, and shows the allocation of functions to different TSC facility components.

It is intended that software developed at the CAE-Link facility will be stored on the GSDE host. Configuration control will also be maintained on the GSDE host.

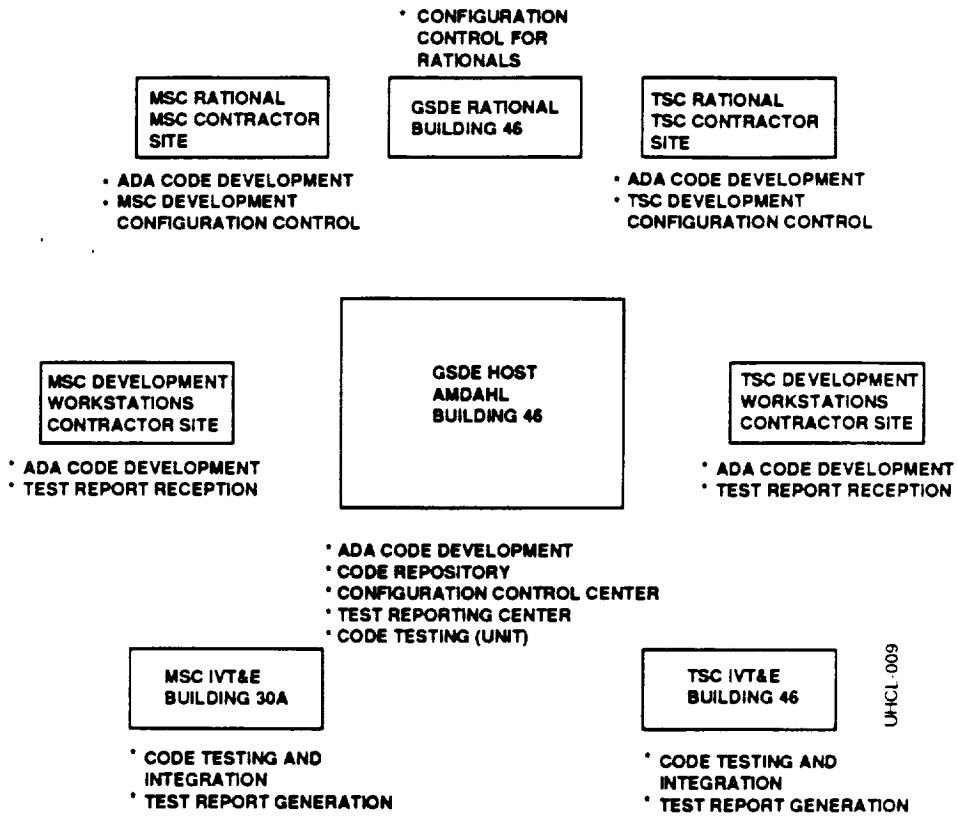


Figure 2-7. GSDE Functional Architecture

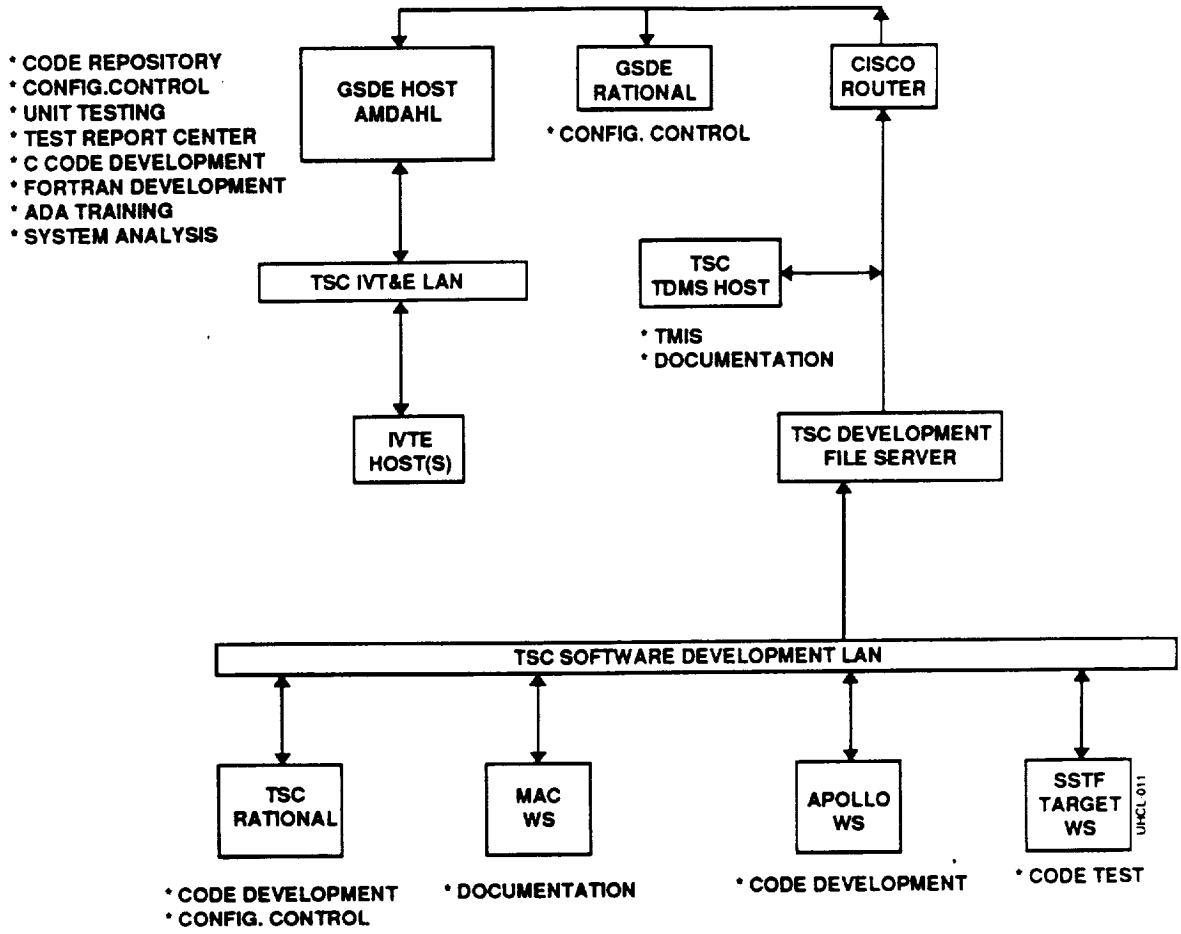


Figure 2-8. TSC Development Facility

The target machines for the developed software will be the IVTE for the SSTF located in Building 5 at JSC. The IVTE hardware is expected to be identical to the operational hardware.

The SSTF IVTE will include support for SSFP SDPs and other specialized devices, including "aural cue" (voice data processing) systems and visual scene processing hardware. These items, along with other special-purpose devices in the SSTF IVTE, will be identified as the SSTF design progresses.

The software for the SSTF is largely a complex, highly distributed, real-time event-driven simulation and control system. The software must perform simulations (e.g., of the Space Station environment, of the modules, of the onboard computer systems) in real time to provide a highly accurate emulation of the real Space Station. The training facility needs the ability to model the various stages of construction and assembly, and so must be flexible and reconfigurable.

2.4.3 SSCC Software Development

Software development for the SSCC will be a distributed process. The bulk of the software will be developed on workstations (Apollo) and Rational development hardware and software located in the FAC building. These development machines will be connected to the GSDE host located in Building 46 at JSC. Figure 2-9 illustrates this configuration, and shows the allocation of functions for components of the development facility.

It is intended that software developed at the FAC facility will be stored on the GSDE host. Configuration control will also be maintained on the GSDE host.

The target machines for the developed software will be the IVTE for the SSCC located in Building 30A at JSC. The IVTE hardware is expected to not always be identical to the operational hardware.

The SSCC will include special purpose hardware for communications processing, for large-scale visual display, for recording, and for communications link management. These devices will be specified during the SSCC design process.

The software in the SSCC is primarily oriented toward receiving, processing, displaying, and capturing real-time telemetry data from the Space Station. In general, it does not include closed-loop real-time processing, although some of the communications equipment may have such constraints. The intent is to provide information to operators for assessment and action. The SSCC software is characterized by high data volumes, complex processing, and human interaction performance goals.

Unlike previous manned space flight control centers, the SSCC will be operational full time for the life of the Space Station Freedom. The SSCC must therefore be highly

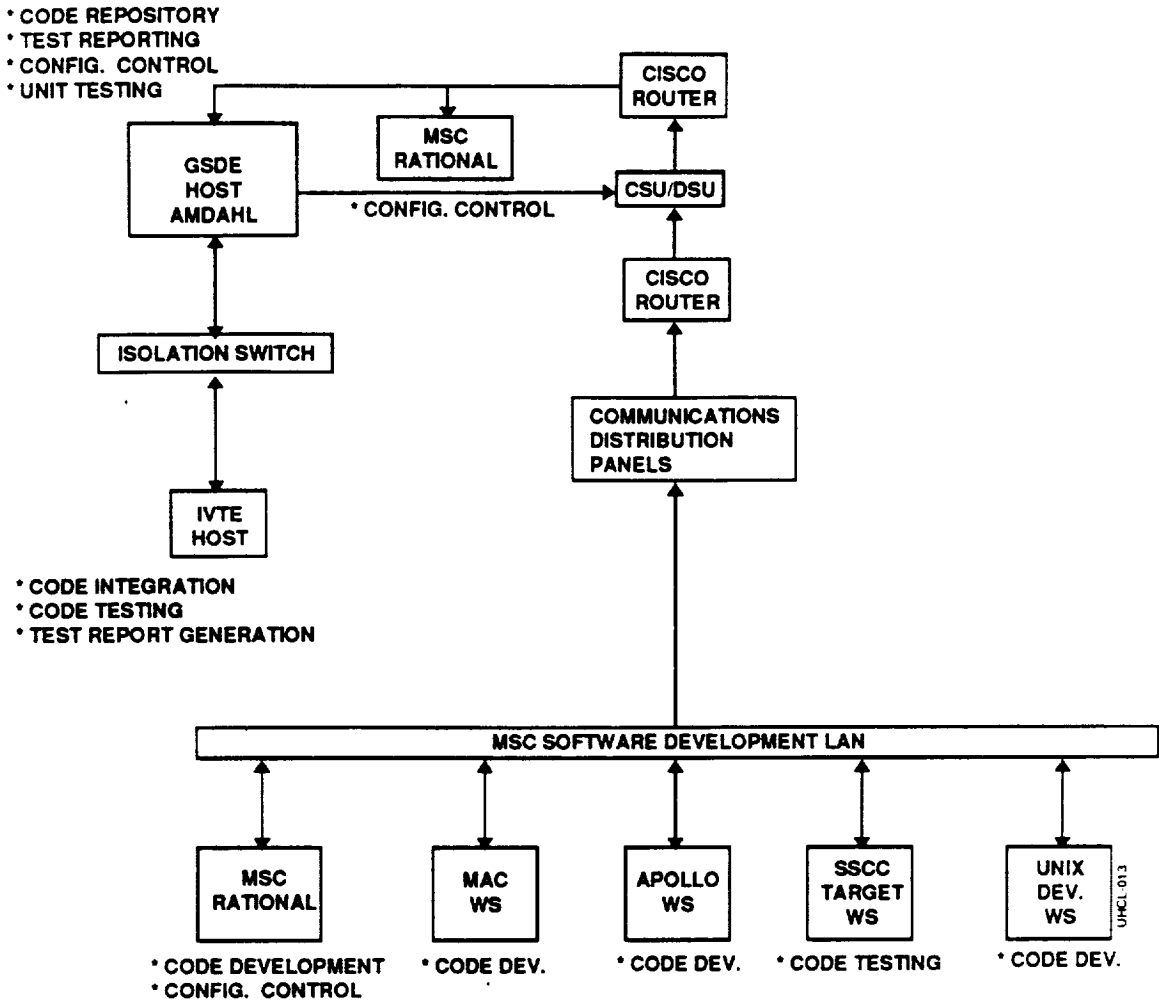


Figure 2-9. MSC Development Facility

reliable and fault tolerant, with appropriate load-sharing and failover capabilities. The software architecture will have to reflect this consideration.

2.5 Requirements Collection Process

This section describes the process used to collect data for this study as well as how that data was used. Data was collected through a series of interviews with personnel responsible for the various pieces of the GSDE. It is important to note that the GDSE is still being defined in terms of development methodology and operational scenarios. Information was collected from the MSC prime contractor, but information is still needed from the TSC prime contractor.

Much of the hardware and software tools to be used for the development of ground software have been purchased and are in place within the building 46 facility. The data collection process involved defining the general structure of the SSTF and SSCC, mapping the development of the necessary software onto the elements of the GSDE, and identifying interfaces where software and control information and status information will have to be communicated from one system to another. The obstacles to smooth communications and transitions were identified in general and were discussed with representatives of the TSC and MSC contractors.

The data collection process also involved reviewing various distributed computing support environments to identify problems that are likely to be found. The examples examined ranged from conceptual analyses to actual, commercially available environments. On the basis of these researches, anticipated interface problems were identified and discussed with TSC and MSC personnel.

Section 3 - Development Process Interface Issues

This section discusses the standard software processes for development and configuration management, and investigates the problems that result from cross-development. This section also discusses various issues within the GSDE (including the SSTF and SSCC) that impose added requirements on the use of this standard process. The normal flow of development is analyzed to expose issues that might hinder this flow.

3.1 Standard Software Process

The process of developing software in the SSE-supplied environment is described at length in SSE documentation. Specific adaptations for ground software development and for the SSTF and SSCC hardware environments are documented by the TSC and MSC contractors, respectively.

The SSE is designed for development on one system with targeting to one other specific platform--the Standard Data Processor. To support this cross-development, the SSE uses actual target machines (DMS kits) and simulation support (a SIB). Ground software development will use some of the same strategies, but has a much wider array of potential targets ranging from mainframes to special-purpose programmable devices.

This development process, for the most part, matches the "peer machine" model of cross-development as defined in Subsection 2.2.2. In the following discussion, areas where the VME approach might alleviate problems are pointed out.

In order to maintain control of the process, cross-development in the GSDE will make use of code management and configuration control services in the GS/SPF for all code, including code that has been ported to the IVTE. Figure 3-1 shows the general sequence of activities.

The implementation process includes the following basic steps:

- Create or adapt source code
- Build host-based executable files
- Perform unit testing on host
- Create target-build scripts
- Build executable files on target
- Perform integration testing on target.

The first four steps are performed entirely in the host environment. The last three steps involve both the host and the target. Figure 3-1 shows the procedures in sequence clockwise from upper left.

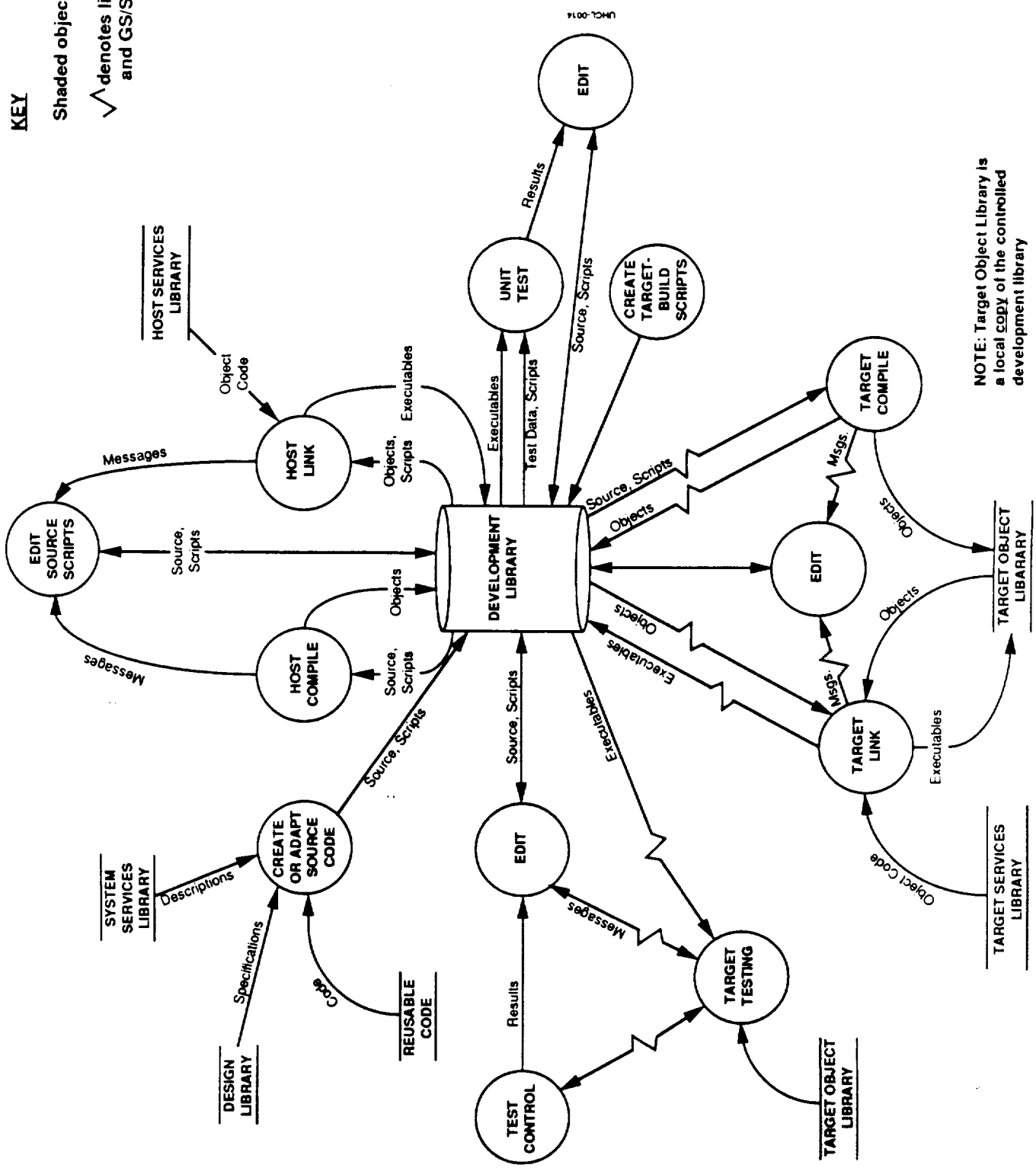


Figure 3-1. Cross Development

Create source code. Working from design information and/or existing code, the tools in the host are used to create application software source code. The presumption is that all Ada code will be developed on Rational systems, using the incremental compiler, Code Management and Version Control (CMVC) system, and other tools as needed. Non-Ada code (FORTRAN and C) will be developed on workstations or on the GS/SPF Amdahl. Reusable code will be placed under configuration control (if not already controlled) and included in the application source files. Configuration control is essentially at the level of the individual developer or development group.

In cross-development, the developer must be familiar with both the host and the target compilers and support tools. The VME approach would reduce the need to learn two different systems to develop software for just one.

Along with the source code, scripts will be created to control the compilation and linking of the application software. These "build scripts" will serve as module reference lists, and will be the basis for target-build scripts to be created later. These scripts may be used as lists for code-reading and certification as well.

Build host-based executable files. Syntactical and interface accuracy will be verified by compiling the source code on the host and linking it with system support files. The code is still controlled at the developer level. The host environment editing system is used to make any changes to the source code and build scripts. The code must be designed to work on the target, but must be compilable on the host. If there are a variety of target environments, this can be difficult to provide in any one host.

Perform unit-testing on host. Still in the host environment, executable files are unit tested using data sets designed to exercise as much of the functionality as possible. Because the host environment does not include all of the interfaces and characteristics of the target, some elements will need to be stubbed out or ignored. Some of these target-only features may be simulated on the host, permitting functional testing to be performed.

A VME approach to supporting cross development would minimize the number of different features to be simulated. Due to the real-time nature of the problem, the performance impact and significance of this approach must be evaluated for each interface.

A common way of performing unit tests is to establish an executable image of part of the system with stubs for incomplete segments and then to plug in new components for testing in place of stubs. The developing subsystem executable image serves as a test framework and ensures that intermodule interfaces get heavily exercised.

The feasibility of performing interprocess and interprocessor communications testing depends on the level of simulation and support provided in the host. Mechanisms for providing such support are discussed later in this report.

Create target-build scripts. The scripts needed to perform compilation, linking, and testing on the target are typically created by modifying the scripts used in the host environment. The Rational Ada Development System has the capability of building such scripts for certain target processors. The requirements and characteristics of the target-machine compilers and linkers must be understood in order for these scripts to be built.

One of the questions involved in planning the target-build process is how much object code is retained on the target. The cleanest way to ensure consistent, repeatable testing is to recompile everything that is not part of the target operating system. In practice, some large systems take so long to compile that total recompilations are to be avoided. This requires careful recordkeeping so that the scripts created for target-building can reliably ensure that all software objects are current.

The target-build scripts themselves are created with editors or special utilities in the host environment. They are subject to the same level of configuration control as the source code and test data.

Build executable files on the target. This step mirrors the host-build process of compiling, identifying, and correcting syntactical and interface errors and then rebuilding. The source code is resident on the host and is transmitted (along with the target build scripts) to the target machines. Compilation and linkage reports are returned to the host environment where any necessary changes to the source code are made. The changes needed for compilation on the target may be incompatible with the host development system (e.g., language-sensitive editor, compiled unit library), in which case the source code must be stored as text. Successful compilations and links produce object files which are returned to the host for storage.

Depending on the target environment, the object files *may* also be retained in the target for use in subsequent compilations. The main requirement is that the target environment must support the use of a *single instance* of an object anywhere in the distributed target. This requirement ensures that outdated files are not left sitting where they might be inadvertently used.

In any case, the objects will be stored on the host, and can be downloaded along with source code, to facilitate system building without massive recompilation. Configuration control remains in the host. If more than one developer has access to a target machine (as is probable), configuration control must be elevated above the developer level.

Perform target-based testing. Executable images and test scaffolding are either downloaded from the host or loaded from the target object library. Test reporting software is included in the scaffolding so that tests can be controlled and evaluated from workstations in the host environment. (Some types of tests, particularly those involving hard real-time constraints, may not be compatible with this procedure. Those tests will be performed directly on the target.)

The outcome of the testing is reported, in real-time or as test log files created during the test, to the developer in the host environment. Any changes necessary to the source code or test data and scripts are made in the host environment. Configuration control stays in the host, even when testing has shifted to the target environment. This ensures that all related development takes advantage of the most up-to-date version of the software.

3.2 Distributed Configuration Management

A major characteristic of software development in the GSDE is that CM must deal with files that are created on several different kinds of machines, providing appropriate levels of control for each stage of development. Before software is delivered for central control, it must be tracked and controlled at the level of developer, development team, or test organization. Following delivery, control will be centralized with provisions for "in use" copies to be moved around the GSDE.

These two stages of control are discussed below.

3.2.1 CM During Code Development

When code is first created, it will be controlled on the machine used for its creation. For Ada code, that generally means one of the Rational R1000s. For FORTRAN and C code, the development machines will generally be workstations. The source code control tools on each machine will be used to control and track the necessary files. At this stage, there is no need for CM *per Se* because there is only one version of the source code. That version is under the control of the originator.

For tracking purposes, the module lists developed during detailed design will serve as control lists to identify the family of files associated with each module. (The "family of files" may include build scripts, test scaffolding and data, primary and included source files, and package and body files.) These lists will also be used in status accounting and for quality assurance (e.g., certification of peer review).

Most developers make use of the available CM tools to track their own work, so the tools must be available at the individual workspace and session level. The files are not placed under official CM, however, until they are delivered or provided for others to use and test.

Once a module is publicized, control passes from the developer to the team leader or group software configuration manager. The actual files will be placed in group- or project-level development libraries, and (typically) checked back out to the developer for refinement and testing. The *development library* can be a single, centrally controlled disk store, but may also be a distributed, logically integrated set of storage facilities.

The critical aspect of CM is that controlled objects are not changed without proper approval and recordkeeping. If that goal can be met with a distributed development library, the cost of CM (and the impact on developers) may be considerably reduced. CM extends to source files and to all object files created from the source code. Since a given object file may be needed in several places at once (by different developers on different machines), there is some risk that a fully centralized library system could become a bottleneck.

On the other hand, if a distributed library system is used, there is danger of inconsistency and of outdated versions of files being used. One of the areas of investigation of this study effort is the availability and dependability of procedures for distributed access and control.

Another area of concern is object naming and name-space considerations. If an object module (under CM) is required for testing, the distributed system should be able to determine where that module resides. (There is little sense in downloading a large object library from the main storage facility if the object is already resident locally. When development is geographically distributed, the cost and time of substantial downloading becomes even more significant.) Objects under control should be uniquely named and identifiable throughout the distributed environment.

This concern is also an issue when it comes to deleting superseded modules from local storage. If each version and each copy of a module is identifiable, the process of purging outdated files is more predictable and reliable. This study effort will consider approaches to establishing system-wide naming procedures. The final *GSDE Interface Study Report* will detail findings in this area.

3.2.2 Post-Delivery CM

Once a module has been delivered for operational use (or possibly for acceptance testing), it is placed under central CM. The source code and all associated files will be stored and controlled on the GS/SPF Amdahl. As far as practical, object files created from source code will also be stored centrally. (This applies only to target-built object modules, not to any host-built object code that may be retained on the development machines.)

This study effort will investigate mechanisms whereby the Rational Code Version Control System (CVCS) can be used to manage files stored on the Amdahl. The Rational Ada Development System provides an effective environment for managing source files, object files, and dependencies, all very important to effective project-wide control. The cost and feasibility of such a shared-function system (Rational control, Amdahl mass storage) needs to be determined.

The biggest advantage to using the Rational for CM is the avoidance of unnecessary recompilation. By keeping track of dependencies at a relatively low level, the Rational can determine when changes in one unit require recompilations of other units. Without such dependency information, a change in one unit can ripple through a system requiring massive recompilation that may not be required by the actual change. Since such recompilations may take hours or days for large software systems, the savings can be significant.

The potential problems of having a Rational R1000 become a CM bottleneck must be evaluated to determine the viability of this approach. Section 5 describes the plan for resolving this issue.

3.3 Implementation Status Reporting

Several issues need to be resolved within the GSDE so that its role of supporting distributed development for the MSC and TSC can be accomplished. The issues discussed in the following paragraphs were discovered during the requirements gathering. Some of the issues are targets for prototyping. Plans for prototyping these issues are discussed in Sections 5 and 6.

3.3.1 Test Status Reporting

The first major issue is the reporting of test results from the target machines back to the development environment. This means that the two target "areas" (Building 5 and Building 30A) would report back to the GSDE test reports. This is complicated by the fact that the development of the software tested could be on any number of machines within the distributed development environment. It is also important to understand that this reporting may be required to be done in real time or the requirements could be satisfied with batch process reporting.

The question becomes one of "where are the results reported?" The first suggested resolution to this issue is to report to the Amdahl using batch processing techniques. At this time (the users are still defining requirements) the need for real time reporting has not been established. In addition, there have not been any requirements to provide reporting to development workstations or to the Rational development environment.

The second suggested resolution is to make reports back to the development workstations that are involved in the testing. This would require that the reporting be done in real time.

At present there are no suggested software candidates for meeting this requirement.

3.3.2 Process Status Reporting

The second interface issue in this area involves controlling and reporting on the status of software which is compiled and integrated on the target system. Source code, commands, and possibly object code are downloaded to the IVTE for compilation and load-image generation. The GS/SPF must be able to request and/or receive status reports on the process. The developer must be able to verify, from the GS/SPF, that the build process has completed successfully. The GS/SPF must be able to determine what object code and load images (if any) reside in the IVTE, and what versions those files represent.

One approach to this problem is to use a global naming- and object-management process to identify and locate all objects (typically, files) in the GSDE. Cronus is one such management process, and will be investigated in this context.

3.4 Software Transparency

It is important that software developed within the distributed GSDE be transportable to the target environments with minimal additional development work being done on the target machines. In addition, it is important to minimize the need to simulate target machines within the development machines (at best there would be no need to simulate the target). Although this is an issue that will be detailed in the final report there are some preliminary findings that suggest methods for resolving some or all of this issue.

The first finding is that there may be candidates for a distributed OS that will allow software development on one machine to be compiled and linked on another machine with little or no "redevelopment" on the target. Cronus is such a distributed OS that is being investigated.

The second finding is that there may be development cross-compilers that will allow some transparency between machines.

The third finding is that there may be some methods for developing software that will reduce the amount of "redevelopment". These too are being explored.

The issue of software transparency will be solved only if detailed knowledge is gathered about both the development and target environments. This knowledge will spawn the use of target emulations within the development environment, methodologies for reducing software/machine dependencies, and new technologies for making code more transparent to the machine.

3.5 General Issues for SSTF Software Development

As with the GSDE as a whole, the SSTF development environment (a subset of the GDSE) has specific issues that need to be reviewed. These are issues of multiple languages, multiple target machines, and IVTE machines becoming operational machines. The following paragraphs will describe the issues involved and any suggestions for further study in search of a resolution.

3.5.1 Multiple Languages in the SSTF

Although the majority of the developed code will be developed in Ada, there will be two sources of non-Ada code. The first will be code that is procured as part of some subsystems that will not be developed within the GSDE (e.g., image generation equipment, aural cue equipment, some signal conditioning equipment). The other source will be "carry over code" from the other existing simulators (e.g., Shuttle Mission Simulator, Shuttle Network Simulator). For cost and schedule reasons, it will not be desirable to recode these sources in Ada.

Because of the multiple languages to be used and developed, the TSC/SSTF portion of the development activities within the GSDE will require a development system that goes beyond the Rational environments. It is expected that much of the non-Ada code that is developed or modified (modified from another source such as is the "carry over code") will be developed or modified on workstations, on the Amdahl, or in the IVTE platforms.

Further definition of the SSTF need for multiple languages will reveal whether or not this non-Ada code can be produced or modified within the strict confines of the GSDE. If it cannot be developed within the GSDE, the issues of configuration control and test reporting become complicated.

3.5.2 Multiple Types of Machines for SSTF IVTE

Many of the target machines that will be used by the SSTF will be procured under a center-wide bid for computational equipment (OADP Contract). At present it is not known which vendor will win the contract or exactly which machines (along with OSs and available COTS (commercial off-the-shelf tools) will be selected. This issue makes requirements gathering difficult. However, important information is available.

There will be at least three different "classes" or "types" of platforms that will be targets for the SSTF portion of the GSDE. The first will be computers that are acquired under the OADP Contract. These machines will have the ability to run the selected SSE standard Ada compilers. The OSs of these machines will be required to support a high

degree of code transparency between them. The second "type" of machine will be those that are acquired outside of the OADP. These machines will be part of subsystem procurements (image generation, simulation interface buffer, etc.). These machines may or may not meet the same requirements as the OADP machines. Their OSs may or may not support the selected Ada and non-Ada compilers. The third class of machines are bare machines which are special purpose in nature and will not have an OS. These machines will be procured or built by the TSC contractor (aural cue, signal conditioning equipment, etc.)

With machines from each "type" as targets for the GS/SPF, the impact of porting becomes a major issue. This issue becomes one of deciding how much of the target machine(s) will be emulated or simulated in the GS/SPF. It also means that for "bare bones" machines, additional equipment might be needed within the GS/SPF or might have to be procured by the contractor to transport code.

3.5.3 IVTE Machines as Target Machines

Because much of the computational equipment for the SSTF will be replicated four times to be able to run four simultaneous simulations at one time, the SSTF will be delivered in stages that will allow that the IVTE computers become the operational computers. In other words the IVTE equipment may be used to integrate and verify operational software and then be delivered as another "set" of IVTE machines that will be procured for development of the next simulation capability.

This process is able to continue until all four simulation capabilities are delivered, and a "permanent" IVTE will then be established. This raises the issue as to how much development work can be done in the SSTF IVTE in the beginning and what will the IVTE then need to maintain and develop software for systems that are already delivered.

It is apparent that there are two IVTEs to be considered for the SSTF. The first is the IVTE that will be delivered as operational, and the second is an IVTE that will not be delivered but will be used to do continuing development and maintenance coding. Each of these will have to interface with the GS/SPF in a different manner.

The SSTF IVTE that is delivered as operational could be used as a development environment and therefore will have different interfaces (both physically and logically) to the GS/SPF. It would be expected that if the IVTE is used as a development environment that the issues of configuration control and test status reporting will be different than if the IVTE was the target of the GS/SPF.

The SSTF IVTE that will be "permanent" will have more traditional interfaces to the GS/SPF. That is to say, that the interfaces will be such that the GS/SPF will be the primary development facility and the IVTE will be the target; the GS/SPF will be the center for configuration control and the center for test status reporting.

Decisions must be made early as to whether the SSTF IVTE will in fact be a development facility for part of its lifetime or will be a target for the GS/SPF from the beginning.

3.6 General Issues for SSCC Software Development

Several issues surround the development of software for the SSCC. These issues include the mixing of Ada and non-Ada and the multiplicity of target machines. The following paragraphs discuss these issues.

3.6.1 Use of Ada and non-Ada in the SSCC

The code to be used in the SSCC falls into three language categories. The first is the development of new code that will be written in Ada. The second is the "carryover" code from the Mission Control Center (MCC) that is non-Ada code and the third is the development of new code that is non-Ada.

The multiplicity of languages to be used in the SSCC will place demands on the GSDE to support the development as well of the configuration control of these mixed languages. Therefore, it may not be possible to relegate the development of the code to a single system within the GSDE for development or configuration control (i.e., Rational environments). It will require the GSDE to support several different compilers and cross compilers, as well as different sets of development tools. This issue can only be solved by allowing code to be developed on workstations, Rationals, and the Amdahl and controlled from each with a central configuration reporting system.

3.6.2 Multiple Target Machines for the SSCC

Like the SSTF, the SSCC will have a multiplicity of targets for the development system to contend with. There will be three "types" of targets; those targets that will be procured under the OADP Contract; those that will be procured as part of subsystems procured under the MSC; and those special targets that are produced by the MSC contractor. Some of these targets will use SSE standard compilers and tools, others will use non-SSE standard compilers and tools, and some will be "bare bone machines" with no OS.

The issue arises as the GSDE will have to support this kind of multiplicity while maintaining a system of configuration control as well as test result reporting. It is anticipated that the Amdahl will play a major role as a repository for all code developed for the SSCC as well as being the center for configuration control and test result

reporting. The tools needed to allow this operation have not been identified yet but are the subject of the final report.

Section 4 - Host-Target Transition Interfaces

Although the host-target transition process cannot be fully characterized until the OADP computers are specified and all special-purpose processors are identified, there are many transition processes that can be investigated with available information. For some of these processes, prototyping may be needed to assess the feasibility of proposed interface support mechanisms.

Three general classes of transition interfaces were investigated. First, operational procedures were identified for moving software between the host and target environments, and for controlling software in either environment. Second, a VME approach to masking the differences between host and target computers was investigated. Third, the study considered the use of simulations and emulations of special-purpose devices to support testing on the host side of the transition.

These three areas are discussed in the following sections.

4.1 Operational Procedures

One type of GSDE interface involves the movement of application code objects from one environment to another, and between machines within an environment. (An "application code object" is a file or related set of files that may include source, object, or operations scripts.) This object-transfer interface must support file transfer, file location and status tracking, and reporting of results of operations (e.g., compilations). These requirements include elements of configuration management and of implementation status reporting.

There are many different protocols, file managers, and communications tools available on different platforms to provide low-level interface support. Code management systems and library support systems also exist in various forms, supporting different languages on different computers. The requirements for GSDE interface support include performance, reliability, consistency across different environments, and connectivity to all systems in the GSDE.

To demonstrate the feasibility and utility of specific mechanisms of interface support, prototypes can be constructed to permit exercising those mechanisms. The following sections discuss the operational goals and interface requirements of these support mechanisms. Subsection 4.1.1 discusses transport and tracking of application code objects. Subsection 4.1.2 discusses reporting on the status of operations on those objects.

4.1.1 Object Transport and Location Tracking

Application code objects (or just "objects" in this subsection) are created, edited, compiled, tested, revised, and stored, typically all on a single platform. In the GSDE context, those objects are also relocated, tracked, backed up, remotely compiled, remotely tested, and superseded on other, usually different platforms. Mechanisms for managing these objects must take into account the low-level transport and communications mechanisms that exist in the GSDE, and must also address the range of different target platforms.

This subsection presents and discusses functional and operational requirements for a general object transport and location tracking mechanism. In general, those requirements are as follows:

- Global, unique object naming and namespace management
- Operations on all types of objects, including text, object code, structured data, load images, and collections of objects
- Support for redundancy of objects (i.e., backup copies) without confusion or risk of incomplete deletion or replacement of an object
- Redundancy and reliability of the namespace manager
- Distributed support for naming operations and inquiries, with reconciliation and coordination mechanisms among distributed namespace managers
- Transmission control protocol/internet protocol (TCP/IP) support for moving objects over LAN and wide area network (WAN)
- Support for network and environment security measures
- Logging of all operations so that audit trails and configuration reports can be generated. (This may apply to a selectable subset of operations or types of objects)

The operations to be supported are listed in the following. All of these operations must be accomplished within the security constraints of the GSDE. For example, the mechanism for deleting an object must require proper authorization.

- Replicating an object, locally or remotely
- Archiving objects
- Inquiry about the location, status, and attributes of an object

- o Transporting objects (moving or duplicating) between devices
- o Creating and naming objects (i.e, from files)
- o Deleting objects
- o Replacing an object with a new file or set of files (producing a new version of the object, with appropriate labeling)
- o Executing objects

4.1.2 Object Execution and Status Reporting

In addition to the need to move objects around in the GSDE, there is a requirement to support execution of objects with feedback on the results of such execution. (The meaning of "execution of an object" is dependent on the object. For an executable or batch command file, the meaning is obvious. The term may not apply to all objects.)

For example, a collection of files (source, object code, commands) may be assembled into a Target-Build object. (The Rational Target Build Utility (TBU) works something like this.) "Execution" of this object implies execution of the commands on the data, after the object has been moved to the target platform. The reports from the compilation process need to be captured and transmitted to the sender of the object.

The types of "execution operations" that will require this support are not yet enumerated. There is no requirement identified to date for real-time status reporting. It may be adequate to capture, package, and transmit the status of any operation after the completion of all object operations. More analysis is needed to clarify this issue.

It is possible that many of the operations identified in Subsection 4.1.1 may actually be supported by defining the operations as characteristics of the objects, and then using the execution facility to invoke any desired operation on an object. Examples of such operations on objects are

- o move (yourself) to platform xxx
- o compile (yourself)
- o print (yourself).

This is an issue that needs further analysis into reporting requirements, performance implications, operations required to be supported, and integration with the namespace management mechanism.

4.2 Using a Virtual Machine Environment (e.g., Cronus)

As noted in Section 2, there are problems associated with transitioning software from one platform to a different one. There are a variety of products, systems, standards, and concepts that address this problem. The common goal is to mask the differences between disparate computer systems. Methods include placing constraints on the applications developers, providing software (e.g., operating systems, standard libraries, simulations) to hide the differences from application code, and providing conversion systems that translate applications from one platform context to another.

The specific platform dependencies that must be masked in the GSDE are not yet known. The ground support systems are still in requirements definition and design, and the hardware components of the IVTEs are not yet selected. At present, this study will emphasize investigation of off-the-shelf solutions to parts of the interface problem. When more details are available about the specifics of the IVTEs, the study effort will concentrate on the specific requirements for masking platform differences.

In other words, the present study effort is directed at identifying a wide range of available products, standards, tools, and environments, so that when more detailed VME requirements are known appropriate solutions can be quickly identified. The research effort of this study, which is described in Sections 5 and 6, will address the types of interfaces anticipated in the GSDE.

Although specifics are not yet available, many characteristics of the target systems will help to bound the range of research. First, the development environment is well specified. Second, the target environments will be based on selections from a limited set of machines. Third, previous experience with control centers and simulation systems provides guidance on the general nature of the software to be supported.

The characteristics of the GSDE computing platforms and the software characteristics of the SSTF and the SSCC are described in Section 2.

4.3 Simulations and Special Devices

Off-the-shelf methods and tools will be investigated for general platform-to-platform interfaces. For special-purpose devices and processors, however, generic solutions are unlikely to be found. Support for cross-development targeted to unique hardware and software will be addressed on a case-by-case basis. Examples of such special purpose devices include the SSFP Standard Data Processor, communications hardware for the SSCC, and visual processing support for the SSTF.

In each case, there will be several alternatives to review in supporting cross development. Those alternatives in general will include the following:

- Connecting a copy of the device directly to the GS/SPF for system development and testing
- Building or buying a device simulator that will run in the GS/SPF (or on a development workstation)
- Placing software on the device (e.g., a standard run time executive) that makes it functionally equivalent to devices that are already supported in the GS/SPF
- Stubbing out interactions with the device (for testing purposes) until the software is transitioned to the target.

As special-purpose devices are defined and detailed requirements are provided by the MSC and TSC contractors, these options will be analyzed and recommendations made for resolving these interface issues.

Section 5 - Proposed Prototype Work

This document specifies prototyping efforts in the three key problem areas identified in sections 3 and 4. The three prototyping efforts are as follows:

- Development of standard interface support for all elements of the GSDE (i.e. implementation of a standard POSIX interface) including development of standard application-to-application communication mechanisms
- Support for software development operations such as distributed CM and implementation status reporting
- Analysis of COTS packages, industry standards, and interface concepts that may aid in solving the problems.

5.1 Virtual Machine Environment

The creation of a VME requires that two different kinds of interfaces be developed. The first of these is application-to-operating system. This involves developing standard interfaces for use on all elements of the GSDE and target environments. This will allow developed software to be ported from the development environment to the target environment without requiring any source code changes before compilation on the target environment. The second interface involves application-to-application communication, which will provide data interoperability so that information may be shared between applications without the need to reformat the data.

5.1.1 POSIX Interface

As POSIX compliance has been mandated for all Unix-based target computer environments within the SSFP, there is a need to provide this standard interface to elements within the GSDE. CSC will provide a prototype POSIX/Ada binding that will be compatible with the Rational Ada Development Environment. This will allow developers to write Ada software that utilizes the POSIX/Ada binding and does not have to be modified to compile in the target environment. CSC will also provide a prototype back-end POSIX simulation on the Rational so developers can perform a minimal amount of unit testing of the POSIX systems calls within the developed software.

By providing a POSIX Ada binding within the Rational Environment, developers will be able to make references to POSIX system calls directly in the software being developed, rather than having to stub POSIX system calls as Ada comments that are then changed on the target machine to allow compilation and execution. The POSIX Ada binding will implement all aspects of POSIX as expressed in the POSIX standard.

The back-end simulation of the POSIX interface will provide developers with increased capabilities for unit testing. The simulation will allow developers to execute software directly in the Rational Environment. This saves the costly effort of porting code to the target platform to perform testing. Furthermore, as this testing may involve some level of debugging, the Rational Environment's debugger could be used to aid in the testing effort.

This prototype will be delivered in three builds. The first build will consist of an installable Ada package specification(s) that supports the complete set of POSIX interfaces. This gives the developers the ability to code POSIX calls directly in Ada software being developed on the Rational R1000. The second and third builds will consist of an executable body for the specification which simulates the actual operation of the POSIX system calls, providing developers with the ability to perform limited unit testing within the Rational Environment. A report that provides the implementation and simulation details for this prototype will also be provided.

5.1.2 Interoperability

Since there is a need for different applications to share data within both the SSTF and SSCC, a standard application-to-application communication mechanism must be developed. Current efforts are underway to determine if commercial packages such as Cronus are capable of meeting this need. After this applicability is determined, CSC will be able to provide a prototype application-to-application communication mechanism. This capability will be prototyped within the GSDE and the target environments for the SSTF and SSCC.

There are two different kinds of communications between applications: synchronous and asynchronous. In synchronous communications, one application must wait until the other is ready to communicate. The waiting application can do no other work until the communication is complete. Asynchronous communication allows one application to deposit a message in a mailbox, which can then be picked up and read by the second application. In asynchronous communication, neither application has to sit idle while waiting for the communication to occur. Both applications may be performing other functions. Figure 5-1 shows the differences between these two modes of communication.

Cronus provides the tools to accomplish both methods of communication. CSC will prototype both modes of communication by using the tools provided with Cronus. In order not to impose any design criteria on either the SSTF or SSCC, a generic communication model will be implemented. This will allow the application-to-application communications to be tailored by the two contractors to meet their needs.

This prototype will be delivered in three builds. The first build will provide an example of synchronous communication between applications. The second build will provide an example of asynchronous communication between applications. The third build will

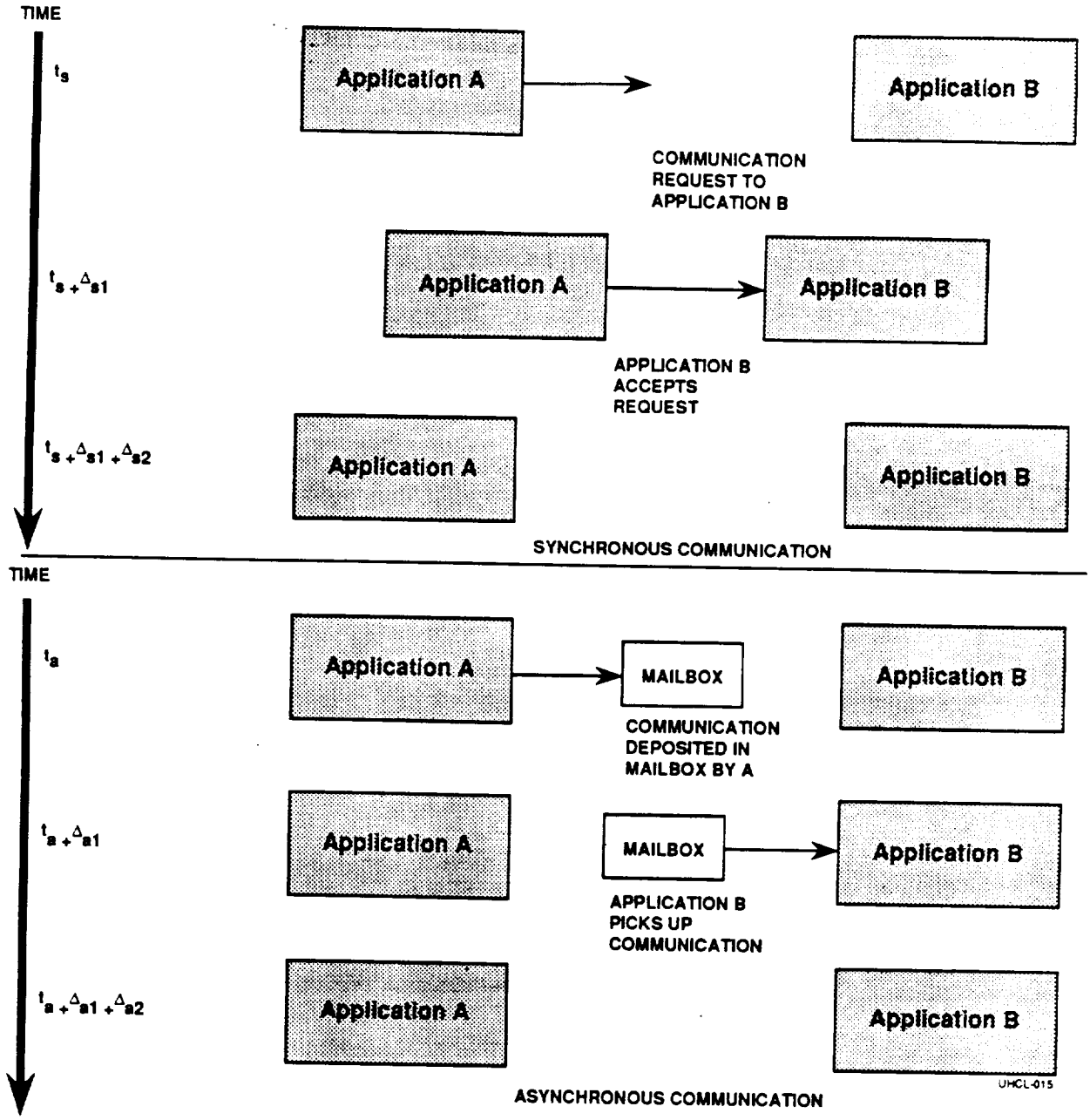


Figure 5-1. Communications Modes

enhance these models on the bases of customer and ground system contractor feedback. Sample applications will be selected as appropriate for planned development within the GS/SPF. A report that provides the implementation details for both models will also be developed.

Both sample communication models will consist of multiple Ada programs that will make calls to Cronus for the communication services. The synchronous model will utilize the Rational and the Sun Workstation to execute two Ada applications that will communicate through Cronus. The asynchronous model will be the same as the synchronous model, with the addition of Ada tasking. Ada's tasking features will be used to make the Cronus calls to allow the applications to continue with other processing while waiting for the communication to complete.

5.2 Software Operations

5.2.1 Distributed Configuration Management

Because the environments for development and system delivery are different, CM is an area for concern. The basic toolset for CM will be provided by the SSE, but details of operational use need analysis in the context of the heterogeneous, distributed target systems of ground software.

Distributed CM provides a means to gain control over all objects (or files) within the entire GSDE. A single point of control can be established for all objects.

CSC will prototype a distributed CM system, integrated with the SSE-provided support, based on Rational's Configuration Management and Version Control (CMVC) system. This prototype will provide for the CM of all information related to the development efforts of both the SSTF and SSCC regardless of the location of that information within the network. The prototype will have the capability to access secondary storage on any networked device. The resident CM system on the Amdahl will also be able to access configuration data located within the Amdahl environment.

This prototype activity will implement a distributed configuration management system based on Rational's CMVC system. The distributed CM tool will provide for location-transparent control of objects within the local area network. It will allow all objects (files) on all networked computers to be placed under CM. If access to the Amdahl at JSC is available, the prototype will include access to this system. Otherwise the prototype will be implemented between CSC networked resources. This prototype will be delivered in one build, along with an accompanying report that provides details on the use of the prototype.

5.2.2 Implementation Status Reporting

Along with the need for distributed CM, there is also a need to pass status and other configuration data from the IVTE to the GS/SPF configuration database during the test and integration phase of the life cycle. CSC will prototype a distributed status reporting system that will provide for this capability.

The distributed status reporting system provides developers and managers with a single point of control for all status data associated with a specific development effort. Developers and test engineers can communicate status data in real-time (or near real-time) to a single point of contact. Information is then immediately made available to project management personnel for their use and review.

The use of Cronus as a distributed status reporting tool will be prototyped. This prototype will provide a single repository for the location of all test and process status reports. User interfaces for elements of the IVTE and GS/SPF would be prototyped where Cronus is available.

This prototype activity will provide a client-server model for the maintenance and tracking of status data within the local area network. The server will be built using the Rational implementation of Cronus. Client interfaces will be built for all other network resources for which Cronus is available. This includes the Rational and the Sun Workstation. As Cronus is not yet available on all GS/SPF resources, this prototype may require alternate solutions if it were to be turned into a full-scale development effort. A report will be provided that details some alternative solutions as well as the operation of the prototype. This prototype will be delivered in one build.

5.3 Investigation of Concepts and Environments

5.3.1 Analysis of COTS Packages and Standards

Several efforts are currently underway to attempt to solve the problem of a heterogeneous development/target environment. CSC is currently investigating the applicability of Cronus to help solve this problem. Several other efforts will be investigated. These include Portable Common Tools Environment+ (PCTE), Common Ada Interface Set-A (CAIS), and Portable Common Interface Set (PCIS). The results of this investigation, as well as the applicability of Cronus, will be included in a report on findings of the prototyping effort.

5.3.2 PCEE Concept Prototyping

The Portable Common Execution Environment (PCEE) is a research effort within RICIS that addresses the problem of supporting complex, distributed real-time computing applications. The PCEE concept addresses the concerns of mission- and safety-critical elements of applications in a fault-tolerant system. A primary goal of the PCEE research effort is to define a portable interface between applications and the distributed computing system. The PCEE concept addresses development, integration, and operational environments and prescribes tools (or tool attributes) for each. The PCEE concept proposes a common interface to differing instruction set architectures, data bases, data communications systems, bare machine implementations, and operating systems.

One of the main thrusts of the work being done by CSC for RICIS is to explore methods and produce software prototypes to facilitate interfacing the GS/SPF and the IVTE. This effort is to some degree a subset of the PCEE research. CSC will review the PCEE concept and apply it to the specific problem of interfaces within the GSDE.

PCEE is still a research concept, with no actual implementation. CSC's prototyping effort in this area will address practical application of the concept in two ways. The first is to map the methodology described in the PCEE literature into the SSE-supported procedures for software development in the GSDE. The second phase of PCEE prototyping will be to implement some of the applicable tools and methods in software.

The first effort will focus on the elements of the PCEE concept that apply to the GSDE, and will review methods and tool characteristics proposed for those aspects of ground software development. The outcome of this stage of prototyping will be a report on how the PCEE concept can be mapped into the real world of the GSDE.

The second effort will concentrate on developing prototype software implementing those elements that fit the GSDE interface problem. Those elements will include interface support tools and distributed development control software. Two concepts in PCEE that seem relevant, based on preliminary analyses, are the DIADEM project, with its virtual node approach to distributed real time control, and the described requirement for Ada-based multiprocessing and interface support.

The products of this second stage of PCEE investigation will be software prototypes of tools and utilities for application within the GSDE.

Section 6 - Technical Approach

The technical approach that CSC will use in building the prototypes comprises the following activities: identifying the questions to be resolved, designing the prototype to address the questions, constructing the prototype, using the prototype to answer the questions, and assessing the results. This approach ensures that each prototype meets the specific needs of the customer. This process will be used to build evolutionary prototypes, which may then be used as the basis for a full-scale development effort.

This technical approach is prescribed by CSC's Digital Systems Development Methodology (DSDM). A prototype is an early experimental model of a system, system component, or system function that contains enough capabilities for it to be used to establish or refine requirements or to validate critical design concepts. It is not meant to be as reliable or robust as an operational system and is seldom constrained by stringent performance, safety, security, or operational requirements.

The most important element in planning and using prototypes is to have a clear statement of the objective of the prototype effort. Such a statement will be developed and reviewed before each prototyping effort. The construction and exercise of each prototype will be based on this statement of objective, and the assessment of the prototype will reflect it.

The deliverables for this effort include technical reports and operational prototypes. Section 6 provides the listing of these items.

6.1 Project Organization and Resources

6.1.1 Contractor Facilities

The key resources for this effort are located at CSC's Virginia Technology Center in Falls Church, Virginia, at CSC's STAR*LAB. The STAR*LAB includes various networked workstation technologies as well as a Rational R1000 Ada Development Environment. The Rational R1000 will be used to develop the Ada software required to complete the prototype effort. A Sun 3/260 and other workstations will serve as the target environment for the prototype.

To demonstrate multisite development, CSC will use workstations and mainframes at other sites via an Internet connection. Other sites include both CSC and other contractor facilities.

6.1.2 Software Engineering Environment

Commercial Software. The prototyping efforts described in this plan will require access to the following software:

Rational R1000 Ada Development Environment - includes:
- Configuration Management and Version Control (CMVC)
- Language Sensitive Editor
- Compiler
- Library Manager
- Project Management Tools
- Distributed Tool Integration Architecture (DTIA)

Additional Rational Products:
- Target Build Utility (TBU)
- Rational X-Windows Interface (RXI)

Amdahl Items:
- Ada Compiler System (GFE via network access)
- Operating System (GFE via network access)
- TCP/IP Networking Software (GFE via network access)

Workstation Products:
- Workstation OS (GFE)
- Technical Publishing Tools (i.e. Interleaf, Framemaker, etc.) (GFE)
- CASE Tools (GFE)

Computer Systems. Access to the following items will be required for the completion of the prototyping efforts identified in this plan.

- Rational R1000 Model 100 (at Star*Lab)
- Amdahl 300E (at JSC)
- Apollo Workstations (GFE)

6.1.3 Government-Furnished Equipment, Software, and Services

To accomplish the goals of this prototyping effort, CSC will require the following GFE:

- o Two fully configured Apollo workstations with all SSE software packages; one at CSC's Virginia Technology Center in Falls Church, Virginia and one at CSC's Houston office for distributed prototyping
- o Access to the Amdahl mainframe located at JSC

- Access to standard TCP/IP software on the Amdahl.

6.2 Prototyping Products

This paragraph lists the deliverables and milestones for the prototyping effort described in Section 5. The products of the prototyping effort are as follows:

- POSIX-Ada binding software and documentation (Rational Ada software)
- POSIX simulation software, (Rational Ada software)
  - Prototype design report
  - Build 1 - first-priority executable package bodies
  - Build 2 - second-priority executable package bodies
- Data Interoperability prototype (Cronus-based Ada software)
  - Prototype design report
  - Build 1 - synchronous communications support
  - Build 2 - asynchronous communications support
  - Build 3 - revised/enhanced communications support
- CM Operations prototype (operational interface to SSE CM system)
- Implementation Status Reporting software
  - Prototype design report
  - Build 1 - prototype software
- PCEE prototype
  - Applications report
  - Tool/utility software

6.3 Risk Management

Table 6-1 lists the risks associated with each of the prototype activities described above, and the means to mitigate these risks.

Table 6-1. Risk Association With Prototype Activities

Prototyping Activity	Risk	Mitigation Plans
POSIX Interface	POSIX Standard Changes	Low Risk: Fix binding to match new standard
Interoperability	Cronus does not provide adequate solution	Medium Risk: Determine if other COTS products are available (from results of COTS study)
	Cronus is not available on all required platforms	High Risk: Determine costs of having Cronus ported to all platforms. If this is unacceptable, other solutions will need to be investigated
Distributed CM	Rational/Cronus does not provide adequate solution	Medium Risk: Determine if other COTS products are available (from results of COTS study)
	No access to Amdahl at JSC	Medium Risk: Implement prototype between Rational and Sun at CSC site
	Cronus is not available on all required platforms	High Risk: Determine costs of having Cronus ported to all platforms. If this is unacceptable, other solutions will need to be investigated
Implementation Status Reporting	Cronus does not provide adequate solution	Medium Risk: Determine if other COTS products are available (from results of COTS study)
	No access to GSDE elements	Medium Risk: Implement prototype between Rational and Sun at CSC site
	Cronus is not available on all required platforms	High Risk: Determine costs of having Cronus ported to all platforms. If this is unacceptable, other solutions will need to be investigated

6.4 Technical Information Interfaces

CSC will need to maintain an interface with each of the other contractors (Ford Aerospace, CAE-Link, and Lockheed) involved in SSFP ground system software development. CSC will support the biweekly technical exchange meetings to be held at JSC (or alternate site as identified by the JSC technical monitor).

6.5 Product Assurance Plan

CSC's Digital Systems Design Methodology (DSDM) prescribes a specific set of policies and procedures for assuring system quality and integrity. The intent of CSC's approach to product assurance is to make quality assurance and configuration management an integral part of system development. To that end, guidelines have been established for consistent application of quality assurance (QA) and CM throughout the development process. Essential elements of CSC's approach include peer inspection and certification, independent review, product assurance recordkeeping, CM, and use of proven development methodologies.

6.5.1 Quality Assurance Approach

The small size and investigatory nature of this effort require that product assurance procedures be relatively informal. The end products of the effort are prototypes of software development tools and utilities and the assessments of those prototypes. The products will be constructed rapidly and are subject to frequent changes during use. This development profile makes it important to record changes and reactions to the prototypes and reduces the importance of acceptance testing. The intended use of the prototypes--investigating interface issues and potential resolutions--makes it necessary to streamline the change procedure so that developers can respond quickly.

The quality goals for a prototype differ from those which apply to an operational system. It would be counterproductive to mandate that prototypes meet the same performance and documentation standards as operational systems. Accordingly, the standards that will be applied to this effort will reflect the need for clear statements of objective and usefulness in meeting that objective. Review and approval procedures will emphasize responsiveness to the needs of rapid development and frequent change. Prototypes will not be subject to the same level of CM that would be applied to an operational system, but will serve to ensure that changes are recorded and results are repeatable.

The product assurance plan for this work consists of the following elements:

- Use of the DSDM-prescribed methodology for prototyping, as noted in Section 6.1
- Designation of a product assurance officer to serve as an independent reviewer and records auditor (note that this is not a full-time position, but rather a role to be filled on an as-required basis)
- Use of peer inspections and reviews to insure continuing technical integrity of the prototypes during development
- A specifically defined set of procedures and electronic utilities to record development progress, prototype changes, planned assessments, and reactions of users.

As prescribed by DSDM, the specifics of these product assurance elements will be defined under separate cover. They will be formulated as working notes, and will be documented in the specific prototype design planning reports.

6.5.2 Configuration Management

While CM is not as rigorous in a prototype development as it is for operational software, there is still a need to be certain of what version of software is being tested or distributed. In addition, worthwhile operational experience with distributed CM will be gained through the use of appropriate tools.

6.5.2.1 Software Library

The software library for this prototype effort will be maintained primarily on the Rational R1000 under Rational's Configuration Management and Version Control (CMVC) system. This system allows for the complete configuration management of all developed software items. Non-developed item (NDI) hardware and software will be maintained by tracking the versions of these items in ASCII files within an NDI subsystem. Changes in versions/releases of NDI can then be tracked via changes in the CMVC-controlled text files.

6.5.2.2 Problem/Change Report

Problem/change reports will be kept and tracked via Rational's Project Management Facility. This facility provides the means to electronically define, maintain, and track problem reports.

Section 7 - Summary and Findings

This document reports on a preliminary study of internal GSDE interfaces. The study effort has focused on the requirements for interface support mechanisms occasioned by the separation of host and target environments (the GS/SPF and the IVTEs, respectively). The study has identified specific interfaces that need to be better understood and supported with software, and has led to a prototyping plan to address the problem. The investigation has included examination of off-the-shelf software to see if ready-made solutions can be found. The support that will be provided by the SSFP SSE is being taken into account.

At the present stage of ground system development, the requirements for interface support in the GSDE are not completely defined. More information is required from both TSC and MSC contractors as the two system developments progress. Information about the OADP-procured platforms is also required but will not be available until a contractor is selected. Nevertheless there are general issues of interface that can be defined and investigated within the limits of available information.

More work is needed to characterize the software development process in the GSDE, with particular attention to configuration management and change control of application code objects. The prototyping effort will develop model procedures and tools to demonstrate methods of distributed CM with centralized storage and overall control. Capabilities of the SSE-provided CM tools as well as the Rational CMVC software will be central to this effort.

Other work in the software development process involves procedures for implementation status reporting between platforms. The requirements for such reporting are being defined, and demonstration software will be developed in a status reporting prototype.

There are several initiatives, commercial and academic, to standardize interfaces across different platforms. These include the POSIX standards, Cronus, and PCEE. Each of these initiatives has the potential to solve some elements of the interface problem. The characteristics and maturity of these (and other) initiatives are being investigated in view of general GSDE requirements. Recommendations will be made for incorporating these initiatives into the GSDE.