NASA-TM-107022 19960003335

NASA Technical Memorandum 107022

An Open Simulation System Model for Scientific Applications

Anthony D. Williams
Lewis Research Center
Cleveland, Ohio

September 1995

National Aeronautics and
Space Administration

AN OPEN SIMULATION SYSTEM MODEL FOR SCIENTIFIC APPLICATIONS

Anthony D. Williams
National Aeronautics and Space Administration
Lewis Research Center
Cleveland, Ohio 44135

SUMMARY

A model for a generic and open environment for running multicode or multiapplication simulations—called the open Simulation System Model (OSSM)—is proposed and defined. This model attempts to meet the requirements of complex systems like the Numerical Propulsion Simulator System (NPSS). OSSM places no restrictions on the types of applications that can be integrated at any stage of its evolution. This includes applications of different disciplines, fidelities, etc. An implementation strategy is proposed that starts with a basic prototype, and evolves over time to accommodate an increasing number of applications. Potential (standard) software is also identified which may aid in the design and implementation of the system.

INTRODUCTION

The traditional engine analysis and design process requires separate analyses for each engine component. These are typically performed using specialized codes that operate within specific domains. The results of the individual analyses must then be (manually) combined to understand the entire system. Several iterations of this process must occur before meaningful results can be obtained. The complexity of this process grows with that of the engine design, resulting in costly design cycles.

To improve this process, engine researchers and designers need the ability to perform (integrated) multidisciplinary, multifidelity, and multicomponent analyses. This would provide them with more accurate and complete results in less time. It requires both new codes, and the ability to simultaneously access and use the capabilities of existing ones. Various groups are presently investigating methods for accomplishing these tasks (refs. 1 to 4). An example is the Numerical Propulsion System Simulator (NPSS) Project at the NASA Lewis Research Center (refs. 1 to 2).

NPSS is a proposed engine simulator for conducting multidisciplinary analyses, being jointly designed and developed by NASA, industry, and academia. Several key elements have been identified for enabling this capability: (1) standard interfaces for data exchange; (2) modular and flexible programs constructed using object-oriented programming techniques; (3) integrated multidisciplinary and multifidelity techniques for modeling engine systems; and (4) high-performance parallel and distributed computing systems.

New procedures and methodologies are required for integrating and running the resulting simulations. The definition (and enforcement) of data, programming, and communications standards are a major part of this effort. A software environment for running these simulations will also have to be developed, that allows users to:

(1) Define and create complex simulations from disparate codes (and assist users when necessary)
(2) Access the wealth of expert knowledge available about these codes and simulations
(3) Control (start, stop, resume), monitor, and debug simulations
(4) Distribute simulations among heterogeneous computers at potentially different sites
(5) Perform these operations quickly, easily, and efficiently

The ideal system would accommodate numerous types of codes, and be usable by (and made available to) many different organizations. Each group should be able to customize the system for their own specific needs. To facilitate this, it must also be:

(1) Constructed using modular programming techniques
(2) Constructed using existing standards (wherever possible)
(3) Free or inexpensive to acquire and use

This paper proposes a generic environment for running multicode (or multiapplication) simulations, called the Open Simulation System Model (OSSM). This model attempts to meet the requirements of an NPSS-type operating environment as described above. An implementation strategy is proposed that starts with a basic prototype, and evolves over time to accommodate an increasing number of applications. The OSSM model places no restrictions on the types of applications that can be integrated at any stage of its evolution. This includes applications of different disciplines, fidelities, etc., as long as they meet the requirements for OSSM compliance.

Several key issues, both generic and domain-specific, must be addressed in order to successfully implement such an environment. The generic issues deal with software implementation and integration requirements common to most large distributed systems. These include the need to adopt standard programming languages and paradigms, tools, data formats, interfaces, and communications protocols. The domain-specific issues deal with problems unique to the integration of scientific applications. These are determined by the capabilities, domain, algorithms, accuracy, and other specifications of the codes (and other applications) being used.

Put another way, the generic constraints determine a code's **ability** to communicate with other applications, and the mechanisms for doing so. The domain constraints determine the practicality or **validity** of the integration. This paper focuses on the generic aspects of application integration. The domain-specific issues are left to the experts currently working those problems, and are beyond the scope of this paper. They are only briefly mentioned where appropriate. Once solved, the resulting information can be integrated with the rest of the system as appropriate (though this may require some minor system redesign).

Numerous systems already posess many of the OSSM features. Most of these were developed for specific applications, and consequently, have unique design goals and methodologies. Although none of them has all of the features listed above, much can be gained by studying their designs and reusing software and ideas as appropriate. Thus, the capabilities of several Lewis-based projects are briefly described later, highlighting the features that are relevant to the OSSM design.

This paper represents the first attempt at defining the OSSM model and its requirements. It identifies the major (software) issues that need to be resolved before building such a system, and some of the work currently being done to address those issues. It also identifies potential software (commercial and other) for the OSSM implementation. Because of the size and complexity of the system, the OSSM design is expected to undergo many changes (due in part, to the comments, suggestions, and criticisms of others) before an implementation is attempted. Teams should be formed to identify (and address) any key areas which have not been properly addressed here, and the model revised to reflect that new information. The rest of this paper discusses the first OSSM model and possible implementation strategies for it.

THE OSSM MODEL

Overview

The OSSM model defines an environment for easily integrating scientific codes and applications into complex simulations. These (applications) may use different languages, data formats, algorithms, fidelities, disciplines, etc. The model consists of four distinct components or software types: the Application Development Tools and Libraries, the Application Libraries, the Information Base, and the Application Executive (fig. 1).

The Application Development Tools and Libraries are used to create and administer other software. The Application Libraries contain the code modules and programs used to build simulations, and to process information stored in the Information Base. These three OSSM components (the Application Development Tools and Libraries, the Application Libraries, and the Information Base) are logical entities comprised of numerous software modules—each with a specific functionality or use. They may, thus, be distributed across many different hosts (and sites) and accessed via network as needed. The Application Executive is a distinct piece of software that resides at each site where the OSSM environment is to run. It directs the simulation process by integrating and then executing Application Library modules. Each of the OSSM components is described in detail in the following sections.

Application Libraries

An OSSM application is anything that processes (reads, writes, displays, analyzes, etc.) information (fig. 2). The primary applications are the scientific codes and subroutines used to model the various engine components. The current OSSM model also defines as applications, the utilities used to monitor, edit, or analyze simulations at runtime. Examples of these include information managers, graphics and analysis programs, and expert systems. A complete description of each application, along with its input and output data requirements, is stored in the Information Base. A generic template for this information is shown in figure 3.

Some applications are standalone programs which can be run independently, while others are software modules or subroutines which have to be run with other programs. A simulation description specifies how applications are combined to perform more complex (e.g., multidisciplinary) analyses. A valid simulation should contain at least one of the primary applications mentioned above. Examples of a graphical and textual description of a simulation containing two scientific codes and a graphics program are shown in figure 4.

Applications are grouped into Application Libraries. This grouping can be by type (see fig. 2), physical location, or a combination of these and/or other characteristics. For simplicity, it is assumed that the contents of each Application Library are contained on the same physical device (or computer), though this need not be the case. Multiple libraries can be stored on the same device or distributed among many different computers on a network (fig. 5). This concept will be further discussed later.

The major goal of the OSSM model is the integration of disparate codes and other applications. Specifications will be required to define the kinds of applications that the system can handle (i.e., the system scope) at various stages in its development. At a minimum, standards need to be defined, and tools and libraries selected, for (1) building consistent and compatible user interfaces, (2) accessing and managing information, (3) representing and exchanging program and graphical data, and (4) communicating between different applications. These are discussed in the next section. Applications meeting these general specifications will be considered OSSM-compliant.

The OSSM environment can simultaneously support applications constructed with different algorithms, programming methodologies and languages, user interfaces, data formats, communication protocols, etc. Though OSSM-compliant, these may still not be compatible with each other. More information is needed to determine application interoperability (ref. 5). The primary determinants for this are based on the capabilities of the code(s) used and the requirements of the simulation to be run. Domain specific constraints like these must be worked out by the simulation experts, and are beyond the scope of this paper.

Other constraints are based on the (software) implementation details. For example, codes that use the same data format(s) will be easier to integrate than codes that do not. For the latter, an attempt must first be made to convert the data. If this cannot be done, the applications are considered to be incompatible, and thus, cannot be integrated. The same is true for communications protocols, grid specifications, etc. The Application Executive consults the Information Base to determine whether applications are compatible, and if so, the best way to integrate them. This process is discussed more in the following sections.

Application Development Tools and Libraries

The Application Development Tools and Libraries are used to create OSSM-compliant applications (fig. 6). The Application Development Libraries (ADLs) are the building blocks of the codes and programs stored in the Application Libraries and parts of the Application Executive. They contain standard functions and routines that perform basic operations for all applications. These make for more efficient and compatible designs. Examples include functions for reading, writing, and searching for data, numerical computations, process communications, and information display.

The Application Development Tools (ADTs) are the mechanisms by which users create and maintain applications with the above libraries. They produce software that meets the standards and specifications required for OSSM compliance. Their use promotes a uniform and consistent approach to developing applications, and minimizes the effort required to integrate new applications into the OSSM environment. Examples include computer aided software engineering (CASE) tools, graphical user interface (GUI) builders, and program development environments.

Interoperability is as important for the ADTs and ADLs as it is for the application programs. A reasonable application could contain functions from a user interface library, a data access library, and a communications library (see fig. 6). These must all work together to be useful. Program development and/or CASE tools used to create other OSSM program modules should be compatible with them as well. The tools should also be able to share and exchange information with each other as necessary.

There are several ways of achieving this interoperability (refs. 6 to 9). The easiest is to pick products that have a common interface or framework for information exchange. This provides interoperability with minimum effort. The drawback is that the selection of tools and their capabilities may be limited. An example of this kind of interface is HP's SoftBench (ref. 9). SoftBench is a unix-based integration framework for CASE tools that currently supports over 70 different products spanning the entire software development cycle.

A more difficult approach would be to select the tools first, and then build the required interfaces (or a framework) for them. This would allow selection of the best or most appropriate tools for each desired function. However, the integration of these tools could require substantial effort (ref. 7). Perhaps the most difficult approach would be to build the entire system—integration framework and tools—from scratch. This method could produce the most sophisticated systems. However, the level of effort required makes it appropriate only for the most demanding needs.

Application development tools, libraries, and standards need to be evaluated for the OSSM programming language and environment; the user interface; information storage, access and management; interprocess communications; etc. A brief description of these follows.

Programming Language & Environment.—A "preferred" programming language should be selected for new application development. Although other languages would also be supported, this one would produce the most compatible programs. The best choice for this is currently C/C++ (ref. 10). Fortran remains important because of its large installed base in the scientific community; but it does not support truly modular or object-oriented programming (more on this later). As cross-language compilers and linkers become more robust, the significance of this decision may decrease.

An integrated programming/development environment and appropriate tools should also be selected for the design and analysis, integration, and implementation of application software. Examples of this include NeXT's NextStep and SunSoft's Solaris—both based on the NeXT OpenStep API (refs. 11 to 12).

User Interface.—Standards should be set for the general style and appearance of all user interfaces. Then, the appropriate libraries and development tools can be selected for implementing them. The large and increasing support for X make it a good choice for the graphics (i.e., windowing) interface, as it is already supported on most unix workstations (ref. 13). Open-GL (ref. 14) is another potential candidate.

High-level libraries can be used to create application interfaces that port to a number of operating systems, windowing environments, and hardware platforms (ref. 15). GUI development environments can also be obtained which facilitate the creation of interactive screens and displays. Several of these also use high-level libraries to create portable applications. Example GUI builders include Neuron Data's Open Interface, NASA's TAE, and TeleSoft's Teleuse (refs. 16 to 18).

Information Storage, Access & Management.—Information storage is discussed in detail in the next section. However, it is worth noting here that standard libraries can be used to access most OSSM information. These serve as interfaces between an application and data stored in the Information Base. This is particularly useful when the data formats are different, or the data is being used by more than one application. Libraries and standards should be chosen that maximize portability (refs. 19 to 20). Tools are also needed to facilitate the design and maintenance of the information storage and management system (i.e.., the Information Base).

Interprocess Communications.—Standards are needed to define the procedures for communicating between different applications. These same (or similar) standards should apply to the individual processes of a single application in a parallel environment. Functions are required that can accommodate multiprocess execution in both single and multiple computer environments, including distributed and/or heterogeneous systems. Potential libraries include PVM (the Portable Virtual Machine—ref. 21), APPL (the Application Portable Parallel Library ref. 22), and MPI (Message Passing Interface—ref. 23).

In order to realize the true benefits of such libraries, sophisticated tools are also needed to assist users in the intelligent decomposition of single and integration of multiple applications (refs. 24 to 25).

Special Software.—In addition to the generic software discussed above, other tools and libraries may have to be developed specifically for the OSSM environment. This includes tools for browsing, editing, and integrating OSSM applications. The browsing tools enable a user to access and search through the Application Libraries distributed among the OSSM sites. The editing tools allow users to edit these applications (or copies of them), and to modify the contents of the Information Base. The integration tools are used to combine applications into complex simulations to be run by the Application Executive (discussed later).

Depending on the implementation details, it may be possible to use existing products for some or all of these tools. For example, a generic C++ class browser may be able to serve as the OSSM object/schema browser. This requires that (1) the OSSM information be stored in a C++ format; and (2) the browser be able to access information distributed across a network. A high degree of interoperability is essential for optimum use of these tools. Further decisions regarding the requirements and implementation of special software can be made when other OSSM details are more finalized.

The Information Base

The Information Base (IB) stores all of the information (data, knowledge, etc.) used and generated by the system, and is constantly evolving in both content and structure (fig. 7). This makes it the largest and most complex part of the OSSM model. The IB contains detailed descriptions of the Application Libraries, ADTs, and ADLs. It also contains generic information about the simulations, experiments, engine models, and computers used in the OSSM environment. This information may be stored as text files, binary files, data base files, etc., in an infinite number of data formats.

The IB must specify what applications exist; what their capabilities, specifications, and requirements are; where they are located; and who has access to them to ensure correct OSSM operation. Appropriate data structures need to be developed to store this information. A typical code entry might describe its use, valid operating conditions, input and output parameters, and recognized data formats. Figure 3 shows a generic template for such an entry. The values for the listed attributes may be single- or multivalued text, numbers, or other more complex data types depending on the item. Other information specific to scientific codes would have to be added to this template. This would include: a description of the code's algorithm(s) and computational methods; a history of the data generated by previous runs; and both expert-provided and system-derived knowledge about the data.

Similar data structures are needed for the rest of the IB contents as well. Much of the knowledge required to do this is still being learned. For example, very little is known about the interactions between various disciplines, codes, and algorithms. More information about computer system architectures and configurations, operating system intrinsics, parallel and distributed applications management, and methods for handling secure data is also needed to effectively take advantage of heterogenous and/or distributed computing environments.

All of this information must be stored (and made accessible) in multiple levels of fidelity. This requires a powerful and flexible data representation paradigm that can model diverse types of information and their interrelationships. The most flexible one to date is the object-oriented (OO) paradigm (refs. 26 to 27). OO design methodologies make it easy to create and manipulate real world models, by describing things in terms of their structure, relationships, and behavior.

Scientific data is typically accessed using implementation-specific procedures (or data interfaces) hard-coded into an application (fig. 8(a)). When more than one application shares data, these procedures must be duplicated and simultaneously maintained (fig. 8(b)). Since each application has direct access to the data, it is difficult (if not impossible) to enforce data integrity or security beyond that provided by the operating system. In object-oriented systems, these procedures are moved from the applications to the data (fig. 9). This is known as encapsulation. Programs (or applications) access data by sending messages to the encapsulating procedures (known as methods), which then perform the desired actions and return the appropriate results. This "protects" the data from undesired results.

A distributed OO Information Base (OOIB) could be implemented in a similar way. Procedures could be written for each data element (or data group), to control access to it and hide its implementation details (fig. 10). Applications would then use these procedures instead of directly manipulating the data files themselves. Code could also be added (if desired) to perform data integrity and security checks, etc., though this may require a substantial amount of additional programming.

It would be fairly easy to design new applications to work in this manner. Existing ones, however, would have to be modified (or augmented) to do so. This could be a major task, but still not as big as recoding the applications entirely. Left unmodified, they could produce the same data integrity and consistency problems discussed above. Software for managing the information distributed throughout the system would also have to be developed. This Information Manager (IM) should be able to determine what information exists globally and where it is stored, to direct requesting applications to the correct location (fig. 11).

Several groups are currently working to define standards for object models and interfaces for integrated, large-scale, distributed information systems like this one (refs. 28 to 31). One such group, the Object Management Group (OMG), is defining the Common Object Request Broker Architecture (CORBA, refs. 30 to 31). CORBA is a specification aimed at ensuring compatibility between different commercial object-oriented environments. Its rising support and popularity increases the potential for compatibility with a large number of software products in the future.

The above implementation distributes the control for each data element with the data itself. Consequently, each becomes an independent entity with its own unique behavior (data format(s), access protocols, etc.). Application queries involving multiple data elements (and thus multiple sources) must thus be aware of the implementation specifics for each of them. An alternate implementation moves the functionality of the individual data access procedures to the IM, expanding its role to include access and control of the distributed data (fig. 12). This new IM would be able to process local, remote, and distributed data requests, while enforcing any restrictions applied to that data (e.g., write protection) by its owner. It would also provide a consistent view and access mechanism for this data, regardless of its type or format (fig. 13).

This kind of IM can be implemented using an object-oriented data base management system (ODBMS, ref. 32). Several commercial systems are available that could effectively handle this task (refs. 33 to 34). Although a custom system could be developed using an OO programming language like C++, this costly endeavor would merely duplicate the efforts already made by these companies. Also, the complex requirements associated with this system are best met by an already tested and mature product.

Commercial ODBMS systems combine the flexibility to store complex data objects with the power of traditional database management systems. Standard database features include persistence, secondary storage management, concurrency, backup and recovery, security, and ad hoc querying capabilities. Other features particular to ODBMSs include object identity, encapsulation, inheritance, polymorphism, and change management. Most are designed to work directly with an OO programming language like C++ or Smalltalk, and have provisions for interfacing to other (standard) languages like C and Fortran.

Any ODBMS acquired for the Information Base should:

(1) allow easy schema and database evolution, to accommodate the constant changes expected in the IB's content and structure;

(2) support distributed databases and client/server operations over a network in a heterogeous environment;

(3) provide access to data stored in other data base formats (including user-defined) in addition to its own

(4) have available appropriate development, browsing, querying, and debugging tools.

Candidates for this software include Objectivity/DB, ObjectStore, Ontos, and Versant (refs. 33 to 34). An evaluation of the use of these products for this purpose is given in ref. 35. If desired, several ODBMSs (or ODBMS servers) could be connected together to create a distributed IM and to improve overall throughput (fig. 14). These could even be different products, as long as they shared a common interface protocol. This is required for inter-DB communications, and to enable global information access and management. As above, a standard like CORBA could be used to accomplish this. Several ODBMS vendors are currently working on adding CORBA compatibility to their products. The OSSM implementation team should closely follow the progress of CORBA and related standards, and eventually choose one (or more) for the OSSM implementation.

A centralized IM that processes all data requests can decrease the likelihood of data integrity problems when the information is shared by multiple applications. On the other hand, it also decreases performance. In many cases this is intolerable. Much thought (and research) needs to go into finding the right solution for this problem. The actual IB/IM implementation may consist of some combination of the two methods discussed above and possibly others.

For example, local environments could exist outside of the OSSM domain that used whatever implementations were appropriate for that installation. This includes choice of code(s), tools, data formats, etc. Information to be added to the IB would have to first be converted to the correct format using various OSSM tools. It could then be transferred using a "checkin" mechanism like that provided by many commercial database products. This would check the data for problems or inconsistencies, and notify the user if any were detected. Otherwise, the information would be added to the IB.

The Application Executive

The Application Executive (AE) is the program that controls and monitors the execution of applications and their resulting simulations in the OSSM environment (fig. 15). As such, it functions as an operating system, an intelligent GUI, a data base manager, an expert system, etc. Unlike the other OSSM software components, which are logical groupings of physically distributed software modules, the Application Executive is an actual executable program. Copies (of it) can be distributed and run on multiple systems and sites to provide simultaneous access to global information, tools, and libraries.

The Executive starts with a user generated simulation description that outlines which applications are to be used and how they are interconnected (special ADTs may be used to create them). These descriptions may be graphical or textual, as illustrated in figure 4. The Executive determines the validity of the resulting simulation using information from the Information Base (see fig. 3). A valid simulation is one in which: (1) the corresponding applications are compatible; (2) their integration is meaningful; and (3) the data requirements of each can be satisfied.

The actual codes and programs to be integrated are stored in the Application Libraries. These may be distributed among various computers at various sites. Applications can either be transferred to a single computer and run there (fig. 16(a)), or assembled and run in a distributed manner (fig. 16(b)). Combinations of these methods are also allowed. The Executive consults the IB when setting up the necessary communications procedures. Special data translation routines (obtained from the ADLs) are used when needed to resolve incompatibilities.

Applications can communicate in several different ways. Direct communications occur via shared computer memory, message passing, or similar methods (fig. 17(a)). These must usually be hard coded into the participating applications. Indirect communications occur through one or more files (fig. 17(b)). This can be done without either application being aware of the other(s), and with minimum or no changes to existing software. If special processing is required, other applications or the Executive can become part of the communications interface (fig. 17(c)). If neither of these options is viable, then the simulation is invalid and cannot be run.

Once a simulation has been defined and set up, the Executive should be able to start, stop, and monitor it. It should also be able to perform some basic analyses and diagnostics on intermediate and final simulation results. More complex analyses should be handled by programs designed specifically for that purpose. These would be integrated into the simulation in a manner like that described above for other applications. It is reasonable to assume that several standard and OSSM-compliant applications, like a data editor, various analysis tools, and an expert system, may become part of a standard OSSM setup.

The Application Executive is a highly complex piece of software that must interact with many different kinds of applications. No single software product is currently available that can perform all of its functions. It may be possible, however, to combine various products together to build it. Custom software can be added as needed to augment the capabilities of these products and integrate the various pieces together.

Much work has been done at Lewis and other organizations to study the potential benefits and various implementations of OSSM-like executives. An example is the Integrated CFD and Experiments (ICE) project (refs. 36 to 38). This project is aimed at creating high speed, interactive computing tools for propulsion systems development. Its design philosophy is to provide a synergistic environment for integrating and analyzing computational fluid dynamics (CFD) and flow physics experimental data. The ICE implementation features an OSSM-like executive that can be used as a model in the OSSM design.

Many other systems less familiar to the author (including much of the NPSS-related work) could also be used as OSSM models. These should be sought out, studied, and used as appropriate before any final designs are proposed. At a minimum, the Application Executive must have: a system/user interface; a data base access and management system; an application integration and operation manager; and an expert system. Each of these is discussed below. The capabilities and implementation of ICE and other systems are also presented where appropriate to illustrate various design options.

System/User Interface.—The ICE implementation consists of various subsystems that perform different operations for the user. The interface to each of these is built using functions from a GUI library (one of several ICE libraries). This library contains functions for displaying and editing documents, data fields, graphics, etc. These can be used in applications to create complex and functional user interfaces that have the same look and feel as the other ICE processes. An example of a GUI building library for scientific applications that uses object-oriented concepts is described in ref. 39.

Other systems take advantage of the capabilities of existing products for their user interface implementations. References 40 and 41 describe two projects which run under the Application Visualization System (AVS). AVS provides numerous tools for processing and displaying scientific data, and a Network Editor that allows users to create, modify and save programs. Using these capabilities, developers have been able to construct systems which allow users to graphically construct arbitrary engine configurations, select and control steady-state and transient operation of the engine, and view results in a graphical form as the simulation is executing.

The ideal interface allows its users to intuitively navigate, identify, evaluate, modify, and share information (ref. 42). This should be the goal of the OSSM Executive interface. At a minimum, it should allow users to easily access, assemble, and run simulations. It should be consistent (both visually and functionally) throughout, and hence, set the standard for other OSSM applications. It should also be easy to use; run on as many different hardware platforms, operating systems, and windowing environments as possible; and be easy to modify and maintain.

Standard GUI development tools and libraries can be used to develop this interface. Specialized (high-level) libraries like the ones described above could also be used. The use of such tools facilitates the addition of future software into the system by others. The selected tools and libraries should produce code that can run on as many different hardware and software platforms as possible, and support various standards. The requirements for such tools are discussed in the Application Development Tools and Libraries section of this paper.

Data Base Access and Management.—Most of the systems discussed in this section use a centralized data base (DB) or knowledge base (KB) to store information. The information is then made available to any application that needs it. The ICE design also supports multiple storage methods. It was initially set up to use unix and Oracle files (through the Oracle Pro-C interface), and is currently being modified to work with a commercial ODBMS (the results of this effort will be published at a later time). ICE provides an extensive library of services which its processes can use to access this information.

MIRIAD (Module Integrator and Rule-based Intelligent Analytic Database) is a framework for integrating scientific modules into a single application program (refs. 43 to 44). It uses a Data Dictionary to manage the flow of data between modules. This set of instructions describes the relationships between data and the procedures for deriving new (updated) data. New code modules are added by describing their input/output characteristics to the Data Dictionary. Design parameters, environmental conditions, and simulation results are stored in MIRIAD database files.

All OSSM information is stored in and managed by the Information Base and Information Manager. The Executive must be able to access all of this multifarious and distributed information (this is true for the other OSSM applications as well). This can be accomplished either: (1) directly through the IM (probably the easiest route, though it assumes that the Executive and IM have compatible interfaces); (2) through program code that interfaces with the IM (i.e., using data access routines provided in the Application Development Libraries); or (3) through each of the supported data formats (the most difficult and least feasible solution). Each of these methods is illustrated in figure 18.

The Executive and IM share the responsibility for handling secure applications, data, and etc. This can be facilitated by the use of a commercial ODBMS as the IM, since most of these have built in security provisions. Further protection can be added at the operating system or network levels, or programmed into specific applications as required. The details concerning what information and applications need to be protected, the level of security required for them, and how this would be implemented across a global multiorganization network have yet to be worked out. For more details on OSSM information management, see the Information Base section of this paper.

Application Integration and Operation.—The primary function of the systems discussed here is the integration and execution of disparate simulation codes and/or modules. The ICE design allows multiple applications (ICE processes) to run and exchange data via shared memory. Processes can be started and stopped interactively by the user to perform various operations. A standard interface (the ICE clipboard) is provided for ad hoc information exchange between applications. Other methods of communication can also be used that do not rely on ICE software for their implementation.

MIRIAD creates a single executable program from individual code modules. It uses a data dictionary to generate the necessary data transfer and translation routines for intermodule communication. The AVS-based systems take advantage of its integration capabilities to perform similar functions. AVS's graphical environment allows them to visually construct simulations on the screen. AVS also accommodates multiprocess operation, and helps users set up required data transfer routines.

The OSSM Executive should have all of the above functionality. It should support the integration of standalone codes, code subroutines, and other applications, as long as their intended use is valid. This can be determined by consulting the IB and IM. Users should be advised of invalid simulations, and assisted while debugging them. If more than one code is involved, the appropriate communications procedures must be set up and any required data translations performed. If the applications are distributed, then that too must be accounted for. Tools to support this entire process should be available to run with the Executive.

The implementation of these capabilities requires the use of many different technologies. An operating system-like interface is needed to execute and control the various applications. It should have parallel and distributed application management capabilities to accommodate multiprocess simulations and distributed computing environments. The software should be able to initialize and perform dynamic load balancing for the various computers. A standard communications protocol (like PVM or APPL) can be used in its implementation. A GUI interface would simplify the simulation definition and integration process. An expert system would assist users in all of these operations. Many of these tools and technologies are discussed in other sections of this paper. All should be further investigated for potential OSSM use.

Expert System.—Several Lewis-based systems were developed specifically to investigate the applicability of Artificial Intelligence (AI) and Expert Systems (ES) technologies to the simulation process. One system, called PROTAIS (ref. 45), began as an intelligent front-end to a 3-D Navier-Stokes flow-solver code named Proteus. PROTAIS was designed to help novice users of the code run as effectively as experts. Its features included the ability to: help users identify and use previously run cases with similar characteristics; check user defined parameters for accuracy and consistency; and perform intelligent diagnostics and pre- and post-processing of simulation data.

Another system, the Artificial Intelligence Integration System (AIIS—ref. 46), was started by the NPSS team to study the use of AI for complex multidisciplinary simulations. Its goal was to prototype a system that would help researchers configure, analyze, control, and diagnose these simulations. Unlike PROTAIS, which is a front-end, the AIIS design integrates scientific (code) modules into the system itself. Considerable effort was placed on the design of a robust object-oriented data structure, and an interface that would facilitate access to and control of that information.

The OSSM ES will provide the means to intelligently process OSSM information. As such, it should work in conjunction with all of the other system modules previously discussed. It will be used to integrate application modules, diagnose problems and discrepancies, and provide overall help and guidance to users. Most expert systems store domain specific information in a knowledge base, along with rules that can be applied to that information. Since all OSSM information will be managed by the IM, the selection criteria for the ES can concentrate solely on its use in developing and processing rules.

The ES chosen must, thus, be able to access the IB/IM. Various methods for accomplishing this are discussed in the Data Base Access and Management section above (see fig. 18). Each of these requires the ES to be able to communicate with external programs and data bases. It should also have a robust rule development and processing system, and an inference mechanism that can be adjusted for different kinds of applications. Tools should be available for creating, browsing, and diagnosing rule structures, and for tracing decision processes. Examples of this type of software include NASA's Clips, Neuron Data's Nexpert Object, and Intellicorp's ProKappa (refs. 47 to 49).

IMPLEMENTATION STRATEGIES

The ideal OSSM implementation would accommodate many different kinds of applications, tools, etc., and allow users access to their favorite programs. Initially, however, the scope will have to be limited. Several systems have been identified which may serve as models for the OSSM design and implementation. An attempt should be made to locate others, and to incorporate their best features into the OSSM design as appropriate. When this is completed, a prototype can be built to test that design.

Suitable test cases (applications) must be identified and specifications written for the prototype. The relevant information to be stored and the procedures for interapplication communications can then be determined. This will require significant input from others, especially those currently exploring methods for multidisciplinary application interactions. Lessons learned from the prototype can be used to revise the design as necessary. New features and information can be added to accommodate new applications. Successive iterations of this process will eventually produce a functional system.

The OSSM implementation should utilize software built on (recognized) standards wherever possible. This applies in particular, to the Information Base and Information Manager, the Application Development Tools and Libraries, and any other software used for development purposes. These should be judged by their overall capabilities, flexibility, maintainability, and cost. Preference should be given to public domain software, or freeware, whenever possible, to facilitate mass distribution and use of the system. Custom software will have to be developed when no suitable alternatives exist.

A major issue yet to be addressed, is the support and maintenance requirements (both hardware and software) for the system. An OSSM administrator group will be needed to: (1) update and maintain the OSSM software; (2) answer technical questions about the applications, their capabilities and specifications; and (3) help new users navigate through the system. A Steering Committee (or user group) will also be required to guide the development and use of the system. This group would be responsible for selecting implementation standards, prioritizing new features to be added, and establishing rules and procedures for the system's use. The committee should contain members from the different OSSM user organizations, and be chaired by a nonpartial (e.g., government) member. The system administrators would be responsible for implementing the decisions of the Steering Committee.

Numerous OSSM implementation strategies are possible. The entire OSSM software could be stored on one or more computers at one physical site (fig. 19). Anyone wishing to run a simulation would have to do so at that site—although remote access and control should be possible. This would require massive computer processing and storage capabilities at that site—especially if more than one simulation is to be run at a time. This should, however, minimize the overall hardware costs.

Centralization should also minimize the staff required to develop, maintain, and support the system software (although a substantial size group would still be required), by minimizing communication and coordination bottlenecks. This should allow them to operate in an efficient manner. They would, however, have to learn about all of the different OSSM applications, tools, and libraries available on the system, to answer user questions concerning their proper use.

The OSSM Steering Committee would be responsible for determining a fair method for distributing the system resources (hardware, software, and staffing) among the various user organizations, as well as the operational costs of those resources. Ground rules would be needed for (1) selecting new hardware and software for the system; (2) establishing run priorities to jobs, people, and organizations; and (3) handling shared and/or proprietary information. Adequate security would have to be provided to prevent unauthorized access to proprietary information.

A more complex, but perhaps more practical implementation, would distribute the OSSM software across the many different user sites (fig. 20). A main site could be established to hold global applications and information. Other sites would contain software used or maintained by specific groups. This software could then be made available to local, global, or select users and groups as desired. This implementation gives OSSM users the most flexibility. It allows each organization to have its own customized environment, and to control to some extent the applications in it.

In this scenario, each organization would be responsible for acquiring and maintaining its own hardware. Thus, an organization's simulation capabilities may be directly related to how much hardware it can afford to purchase (or borrow). Each organization would also be responsible for supporting its own administrators. Their job would be to develop, maintain, and support the OSSM software at that site, while following the guidelines established by the Steering Committee. This should include answering any questions about that software, whether from on- or off-site users.

The Committee and its administrators (if any) would then have to monitor and coordinate the activities of each of these sites—each with different applications, hardware configurations, and priorities. This would significantly increase the communications and coordination requirements for them. It would also increase the chance for software incompatibilities, access and security violations, and other problems to occur. However, the complexity of some of their other duties as listed above should reduce. All of these issues must be further investigated before deciding on a final OSSM implementation.

SUMMARY/CONCLUDING REMARKS

A model for a generic and open environment for running multicode or multiapplication simulations—called the open Simulation System Model (OSSM)—has been proposed and defined. This model attempts to meet the requirements of complex systems like the Numerical Propulsion Simulator System (NPSS). The model emphasizes the use of existing standards wherever possible. Several systems have been identified which may serve as models for the OSSM design and implementation. Others need to be sought out and included as appropriate. Potential software (both public domain and commercial) has also been identified for the possible implementation.

More research is needed in the areas of multidisciplinary code integration and software interoperability (standards and technologies). Some of these issues are currently being addressed. New efforts must be initiated to address others. Suitable test codes and applications must then be identified, and specifications written, for a prototype. The design and development of such a prototype will require teams formed with individuals from different backgrounds, disciplines, and organizations. When completed, the OSSM design can be tested, evaluated, and modified as necessary.

REFERENCES

1. Claus, Russell W.; Evans, Austin L.; and Follen, Gregory J.: Multidisciplinary Propulsion Simulation Using NPSS. Interdisciplinary Technology Office, NASA Lewis Research Center.
2. Afjeh, A.; and Reed, J.: Report on the NPSS-MOD0-ADPAC Coupling: Zooming on EEE Fan Stage. Interdisciplinary Technology Office, NASA Lewis Research Center, August 1993.
3. Cole, Gary L.; Melcher, Kevin J.; Chicatelli, Amy K.; Hartley, Tom T.; and Chung, Joongkee: Computational Methods for HSCT-Inlet Controls/CFD Interdisciplinary Research. NASA TM–106618, ICOMP–94–10, AIAA-94–3209, June 1994.
4. Myklebust, Arvid; and Gelhausen, Paul: Putting the ACSYNT on Aircraft Design. Aerospace America, September 1994, pp. 27–30.
5. Donovan, John W.; Nance, Barry; Fetterolf, Peter; Vaughan-Nichols, Steven J.; Anderson, David M.; Sherwood, Bruce A.; Hubley, Mary; Rasmus, Daniel W.; and Ullman, Ellen: Interoperability (A Collection of Articles on Interoperability). BYTE Magazine, Vol. 16, No. 12, November 1991, pp. 185–267.
6. Zarrella, Paul F.: CASE Tool Integration and Standardization. Technical Report #SEI–90–TR–14, Software Engineering Institute, Carnegie-Mellon University, December 1990.
7. Radar, Jock;, Brown, Alan W.; and Morris, Ed: An Investigation into the State of the Practice of CASE Tool Integration. Technical Report #SEI–93–TR–15, Software Engineering Institute, Carnegie-Mellon University, August 1993.
8. Honeywell, Inc.: DOS Design/Application Tools. Final Technical Report #RADC-TR–90–203, Vol. 1–3, Rome Air Development Center, Air Force Systems Command, September 1990.
9. Leach, Norvin: HP's SoftBench for CASE Gains Multivendor Tool Support. PC Week, Vol. 10, No. 11, March 22, 1993, pp. 61–63.
10. Atkinson, Lee; and Atkinson, Mark: Using C/C++. Que Corporation, 1993.
11. Semich, J. William: OpenStep (NeXT Inc. and Sun Microsystems Object-Oriented Operating System). Datamation, Vol. 40, No. 6, March 15, 1994, pp. 29(2).
12. O'Brien, Timothy: SunSoft Lays Out the Road Map to "Distributed Objects Everywhere." Distributed Computing Monitor, Vol. 9, No. 5, May 1994, pp. 23(5).
13. Kaare, Christian: The X Window System: a Universal Graphical Interface. PC Magazine, Vol. 10, No. 10, May 28, 1991, p. 323.
14. Prosise, Jeff: Advanced 3-D Graphics for Windows NT 3.5: Introducing the OpenGL Interface, Part 1. Microsoft Systems Journal, Vol. 9, No. 10, October 1994, p. 15(13).
15. Ga Cote, Raymond: Code on the Move. BYTE Magazine, Vol. 17, No. 7, July 1992, pp. 206–226.

16. Cunningham, Cara A.: Neuron Data Set to Ship Update for GUI Builder (Open Interface 2.0 program development software). PC Week, Vol. 9, No. 19, May 11, 1992, p. 16.
17. Szczur, Martha R.: Transportable Applications Environment (TAE) Plus, A User Interface Development and Management System. ESA SP-308, October 1990.
18. Pallatto, John: TeleSoft's Tool Targets OSF/Motif GUI Design. PC Week, Vol. 8, No. 26, July 1, 1991, pp. 45–46.
19. Sosinsky, Barrie: Graphically Speaking (A Guide to Graphics File Formats). MacUser, Vol. 10, No. 1, Jan. 1994, p. 145(5).
20. Revenge of the NURBS (Nonuniform Rational B-Splines). AI Expert, Vol. 9, No. 6, June 1994, p. 54.
21. Sunderam, V. S.: PVM: A Framework for Parallel Distributed Computing. Concurrency: Practice & Experience, Vol. 2, No. 4, December 1990.
22. Quealy, Angela; Cole, Gary L.; and Blech, Richard A.: Portable Programming on Parallel/Networked Computers Using the Application Portable Parallel Library. NASA TM–106238, July 1993.
23. Message Passing Interface Forum: MPI: A Message-Passing Interface Standard. ARPA/NSF Grant No. ASC–9310330, March 31, 1994.
24. Chang, Long-Chyr; and Smith, Brian T.: Classification and Evaluation of Parallel Programming Tools. Technical Report #UNM-TR–CS90–22, Dept. of Computer Science, New Mexico University, 1990.
25. Murphy, C. G.: Literature Survey on Tools. Technical Document 1853, Science Applications International Corp., July 1990.
26. Graham, Ian: Object Oriented Methodologies. Prentice Hall, 1990.
27. Byard, Cory: Object-Oriented Technology a Must for Complex Systems. Computer Technology Review, Fall 1990, pp. 15–19.
28. Nicol, John R.; Wilkes, C. Thomas; and Manola, Frank A.: Object Orientation in Heterogenous Distributed Computing Systems. IEE Computer, June 1993, pp. 57–67.
29. Wayner, Peter: Objects on the March. BYTE Magazine, January 1994, pp. 139–150.
30. Object Management Group: Object Management Architecture Guide, Revision 2.0. OMG TC Document 92.11.1, September 1992.
31. Object Management Group and X/Open: The Common Object Broker Architecture and Specification, Revision 1.1. OMG TC Document 91.12.1, December 1991.
32. Khoshafian, Setrag: Modeling with Object-Oriented Databases. AI Expert, October 1991, pp. 27–33.
33. Hurson, A. R.; Pakzad, Simin H.; and Cheng, Jia-bing: Object-Oriented Database Management Systems: Evolution and Performance Issues. IEEE Computer, February 1993, pp. 48–60.
34. Bryant, David, Datapro Information Services Group: An Overview of Object-Oriented Database Management Software. McGraw-Hill, Inc., October 1992.
35. Williams, Anthony D.: An Evaluation of Object Oriented Data Base Management Systems for Scientific Applications. Computational Technologies Branch of the Internal Fluid Mechanics Division, NASA Lewis Research Center.
36. Szuch, J. R.; Arpasi, D. J.; and Strazisar, A. J.: Enhancing Aeropropulsion Research with High-Speed Interactive Computing. NASA TM–104374, 1991.
37. Babrauckas, Theresa L.; and Arpasi, Dale J.: Integrated CFD and Experiments Real-Time Data Acquisition Development. ASME Paper 93–GT–97, May 1993.
38. Stegeman, James D: User's Manual for Interactive Data Display System (IDDS). NASA TM–105572, 1992.
39. Curlett, Brian P.: An Adaptive Graphical User Interface Framework for Object Oriented System Simulations. Interdisciplinary Technology Office, NASA Lewis Research Center.
40. Reed, John A.; Afjeh, Abdollah A.; Follen, Gregory; and Putt, Charles: A Summary of the Numerical Propulsion System Simulation Mod1 Graphical User Interface. Interdisciplinary Technology Office, NASA Lewis Research Center.
41. Homer, P.T.; and Schlichting, R.D.: Using Schooner to Support Distribution and Heterogeneity in the Numerical Propulsion System Simulator Project. Concurrency: Practice and Experience, vol. 6(4), June 1994, pp. 271–287.
42. Jacobson, Bob: The Ultimate User Interface. BYTE Magazine, Vol. 17, No. 4, April 1992, pp. 175–182.
43. Jongeward, Gary A.; Kuharski, Robert A.; Rankin, Thomas V.; Wilcox, Katherine G.; and Roche, James C.: The Environment Workbench: A design tool for Space Station Freedom. NASA Report #NAS3–25–347.

44. Lilley, John R.; and Greb, Agnes: A code for conducting systems-level survivability analysis of nuclear power systems. Proceedings of the 26th Intersociety Energy Conversion Engineering Conference (IECEC '91), Aug. 1991, Paper #A92–50526 21–20.

45. Williams, Anthony D.: The Development of an Intelligent Interface to a Computational Fluid Dynamics Flow-Solver Code. NASA TM–100908, 1988.

46. Pimentel, Guillermo: Initial Operating Capability of the Artificial Intelligence Integration System for the Numerical Propulsion Simulation System. Interdisciplinary Technology Office, NASA Lewis Research Center, 1994.

47. Riley, G.: CLIPS 6.0—The C Language Integrated Production System, Version 6.0. Report #MSC–22429, NASA Johnson Space Center.

48. Gevarter, William B.: The Nature and Evaluation of Commercial Expert System Building Software Tools. NASA TM–107914, 1987.

49. Bloor, Robin: New Intelligence: Expert Systems are Reemerging to Satisfy Broader Needs than their Predecessors. DBMS, Vol. 6, No. 13, Dec. 1993, pp. 12(2).

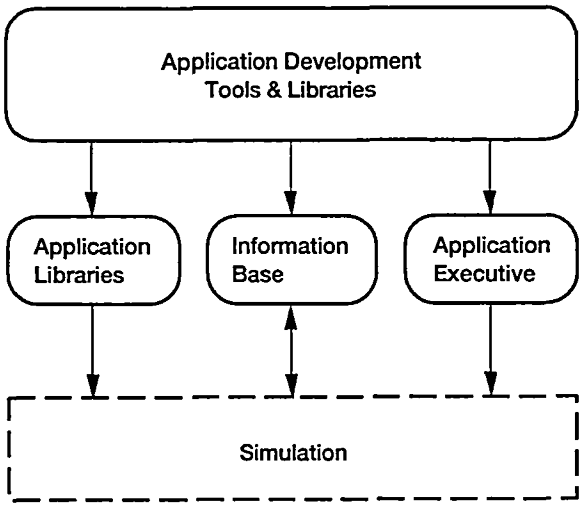
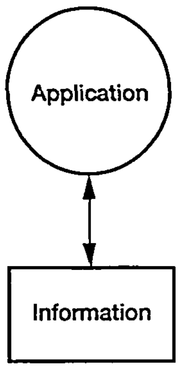


Figure 1.—The OSSM model.



Application:

- Any program that processes information

Examples of application types:

- Science codes/modules
- Information managers
- Graphics programs
- Data analyzers
- Expert systems
- Auto-grid generators
- Data acquisition programs

Figure 2.—The application model.

Applic. ID:	CODE1
Name:	MEGA-CODE
Type:	2-D Navier Stokes
Description:	This code calculates ...
Specs:	To operate correctly, ...
•	
•	
•	
Inputs:	5
Input1:	I1
Type:	INTEGER
Unit:	N/A
Default Value:	0
Minimum Value:	0
Maximum Value:	100
Input2:	I2
Type:	INTEGER
Unit:	N/A
Default Value:	0
Minimum Value:	0
Maximum Value:	50
Input3:	I3
Type:	REAL
Unit:	DEGREES K
Default Value:	288
Minimum Value:	0
Maximum Value:	1000
Input4:	I4
Type:	REAL
Unit:	KG
Default Value:	0
Minimum Value:	0
Maximum Value:	_
Input5:	I5
Type:	REAL
Unit:	M/SEC
Default Value:	0
Minimum Value:	0
Maximum Value:	1000

Figure 3.—Simple example illustrating the information stored in the Information Base for each application. Application specific information can be added as needed.

Outputs:	5
Output1:	O1
Type:	REAL
Unit:	FT
Minimum Value:	_
Maximum Value:	_
Output2:	O2
Type:	REAL
Unit:	FT/SEC
Minimum Value:	_
Maximum Value:	_
Output3:	O3
Type:	INTEGER
Unit:	N/A
Minimum Value:	0
Maximum Value:	100
Output4:	O4
Type:	REAL
Unit:	PSI
Minimum Value:	_
Maximum Value:	_
Output5:	O5
Type:	REAL
Unit:	DEGREES K
Minimum Value:	_
Maximum Value:	_

●
●
●

Site/Location:	NASA Lewis Research Center, Cleveland, Ohio
Host:	app-server.lerc.nasa.gov
Contact(s):	A. WILLIAMS (216) 433-4000
Comments:	This code does not work with ...

Figure 3.—Concluded.

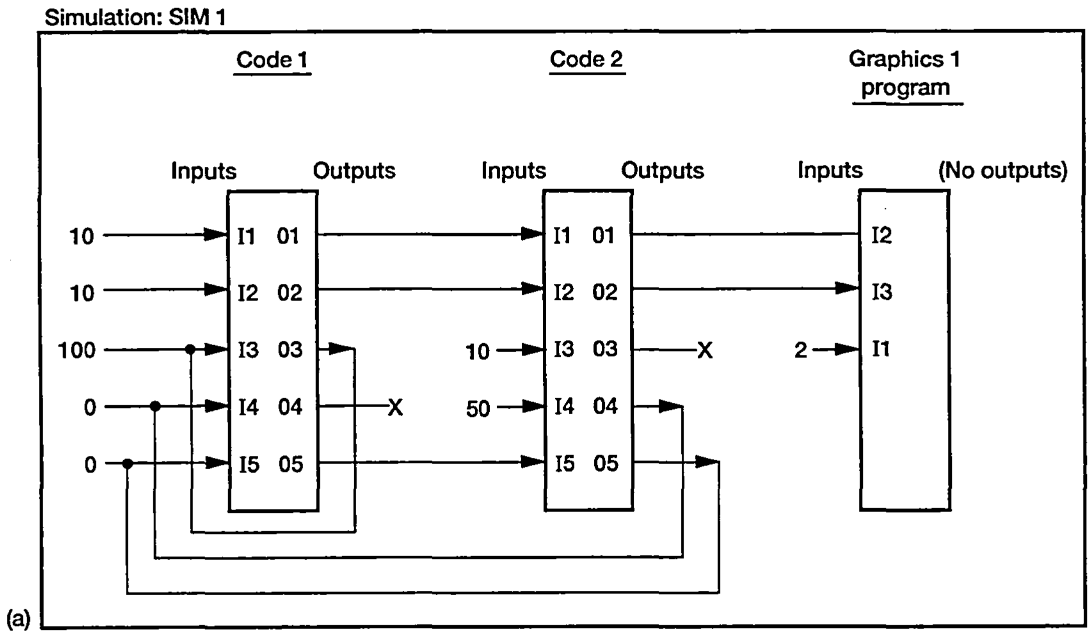


Figure 4.—(a) Example of a graphical simulation description containing 2 scientific codes & 1 graphics program.

```
Simulation:     SIM1
Type:           Engine Simulation
Exec Host:      major.lerc.nasa.gov
Mode:           Interactive
No Apps:        3
No Iterations:  1000

Application:    CODE1
Type:           2D Scientific
Exec Host:      major.lerc.nasa.gov
Input:          I1
Initial Value:  10
Update Value:   N/A
Input:          I2
Initial Value:  10
Update Value:   N/A
Input:          I3
Initial Value:  100
Update Value:   CODE1::O3
Input:          I4
Initial Value:  0
Update Value:   CODE2::O4
Input:          I5
Initial Value:  0
Update Value:   CODE2::O5
Output:         O1
Output:         O2
Output:         O3
Output:         O4
Output:         O5
Stop Cond:      I2 > 50
Stop Cond:      O3 > 100

Application:    CODE2
Type:           1D Scientific
Exec Host:      colonel.csu.edu
Input:          I1
Initial Value:  CODE1::O1
Update Value:   CODE1::O1
Input:          I2
Initial Value:  CODE1::O2
Update Value:   CODE1::O2
Input:          I3
Initial Value:  10
Update Value:   N/A
Input:          I4
Initial Value:  50
Update Value:   N/A
Input:          I5
Initial Value:  CODE1::O5
Update Value:   CODE1::O5
Output:         O1
Output:         O2
Output:         O3
Output:         O4
Output:         O5
Stop Cond:      N/A

Application:    GRAPHICS1
Type:           Analysis
Exec Host:      private.lerc.nasa.gov
Input:          I1
Initial Value:  2
Update Value:   N/A
Input:          I2
Initial Value:  CODE2::O1
Update Value:   CODE2::O1
Input:          I3
Initial Value:  CODE2::O2
Update Value:   CODE2::O2
```

(b)

Figure 4.—(b) Example of a textual simulation description containing 2 scientific codes & 1 graphics program.

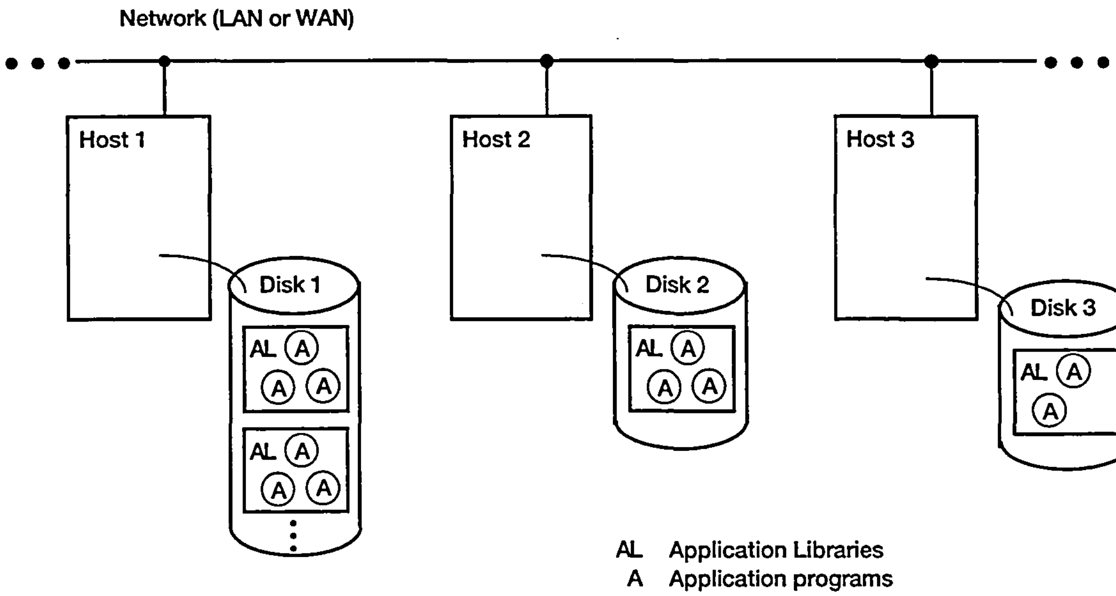


Figure 5.—Distribution of Application Libraries.

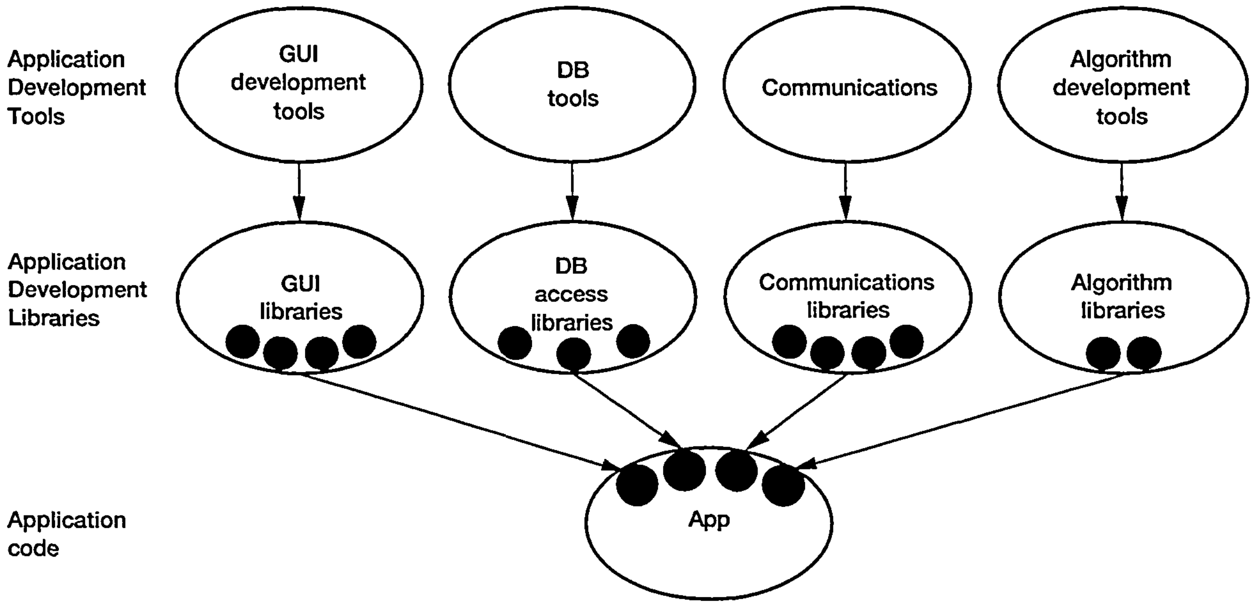


Figure 6.—ADTs & ADLs help create OSSM-compliant applications.

Information base

Code descriptions	Other application descriptions	Development tools & libraries info	Experimental models
Simulation models	Engine models	Algorithms	Grids
Run data	Site info	Computer info	User info

Figure 7.—Contents of the Information Base.

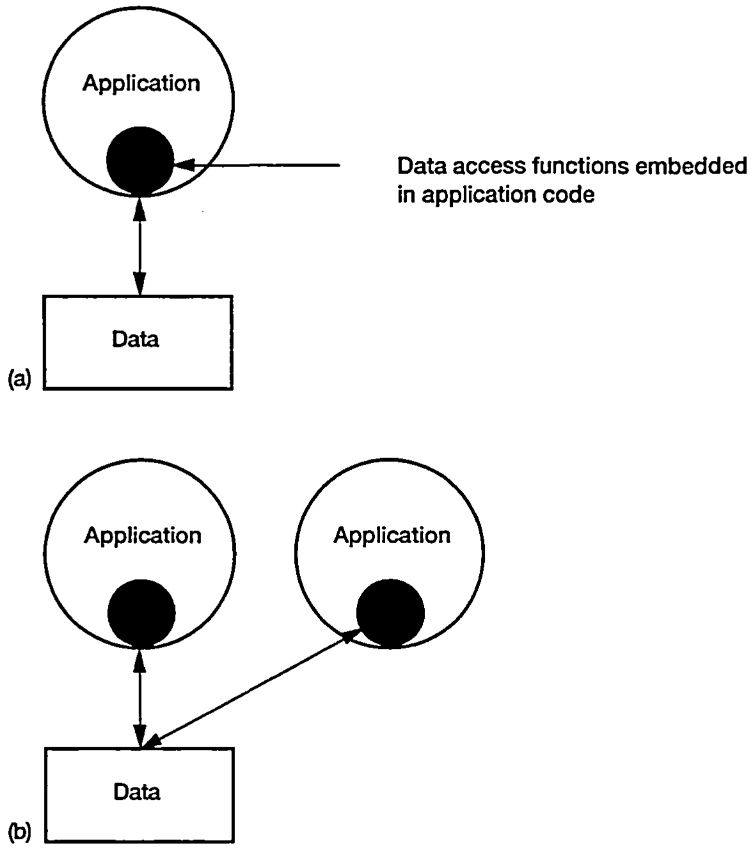


Figure 8.—Embedded data access functions in (a) single and (b) multiple applications allow free (uncoordinated) access to data.

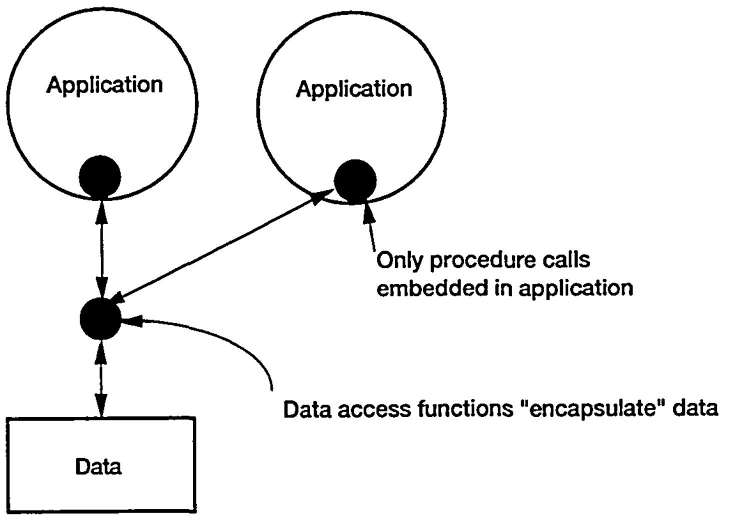

Application
Application
Only procedure calls embedded in application
Data access functions "encapsulate" data
Data

Figure 9.—Data encapsulation controls access to data and provides consistent interface to it.

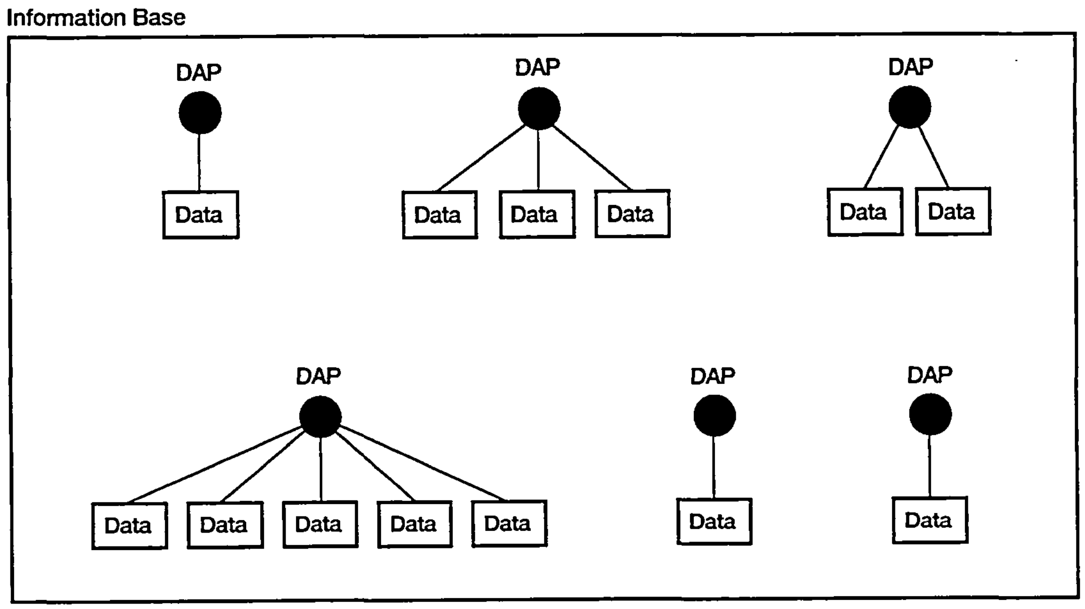


Figure 10.—A distributed OOIB implementation that uses data access procedures (DAPs) to access data elements.

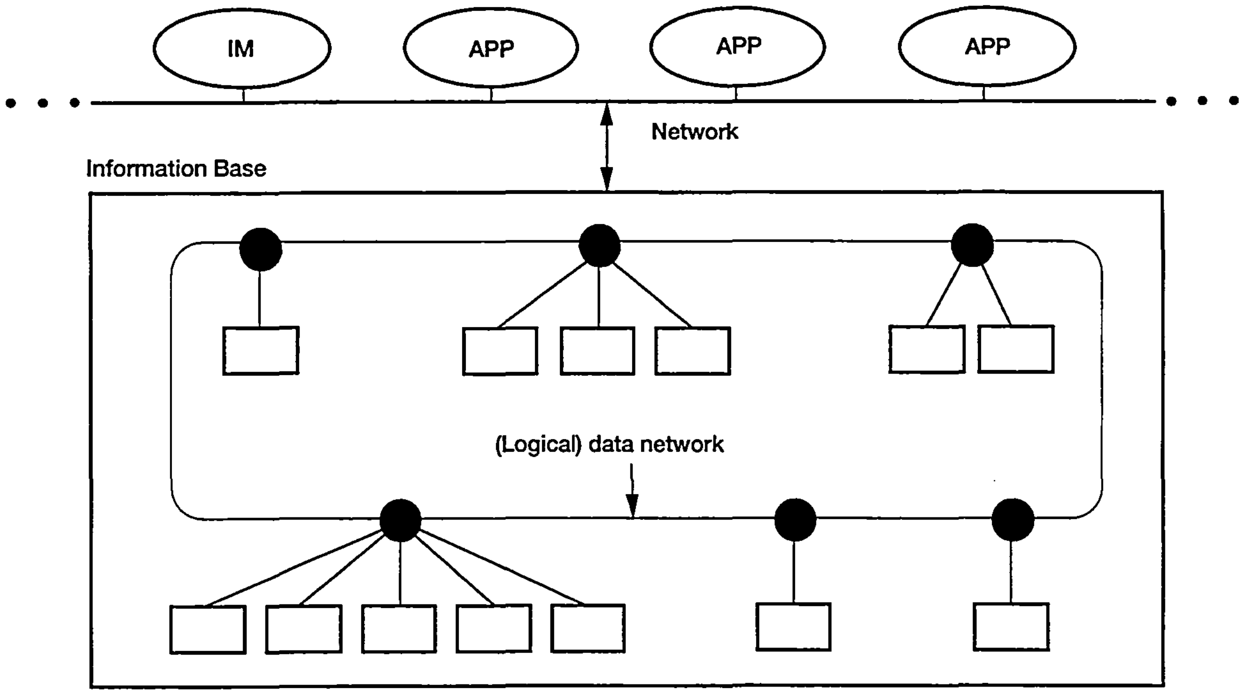


Figure 11.—The Information Manager (IM) notifies applications of the existence and location of requested data in the Information Base.

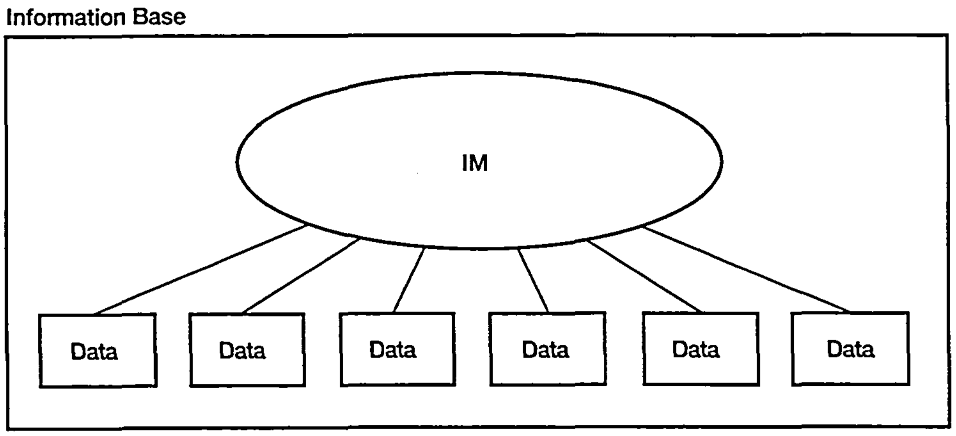


Figure 12.—An OOIB that uses the Information Manager to access data. This data may be stored at multiple sites in multiple formats.

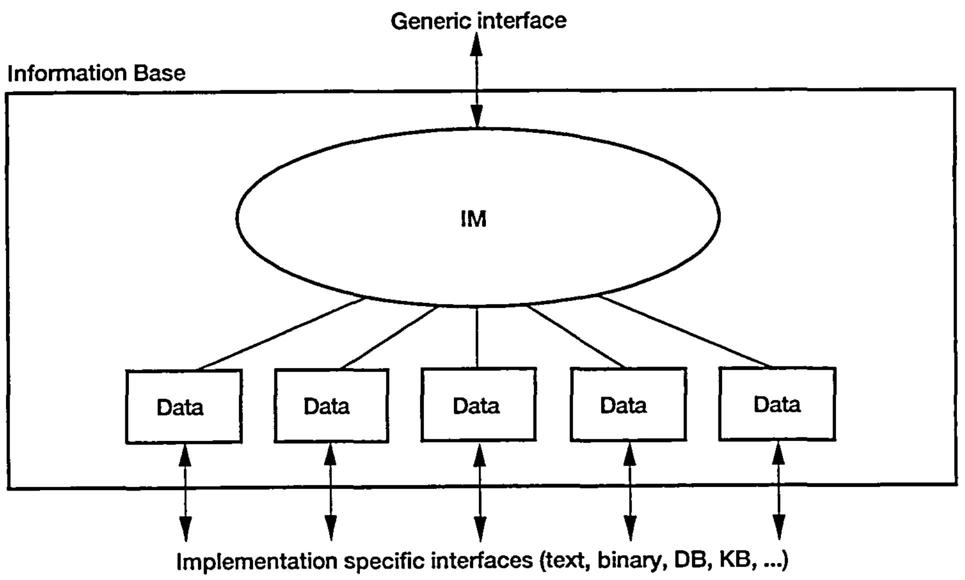

Generic interface
Information Base
IM
Data
Data
Data
Data
Data
Implementation specific interfaces (text, binary, DB, KB, ...)

Figure 13.— The IM provides a (generic) consistent interface to all OSSM data.

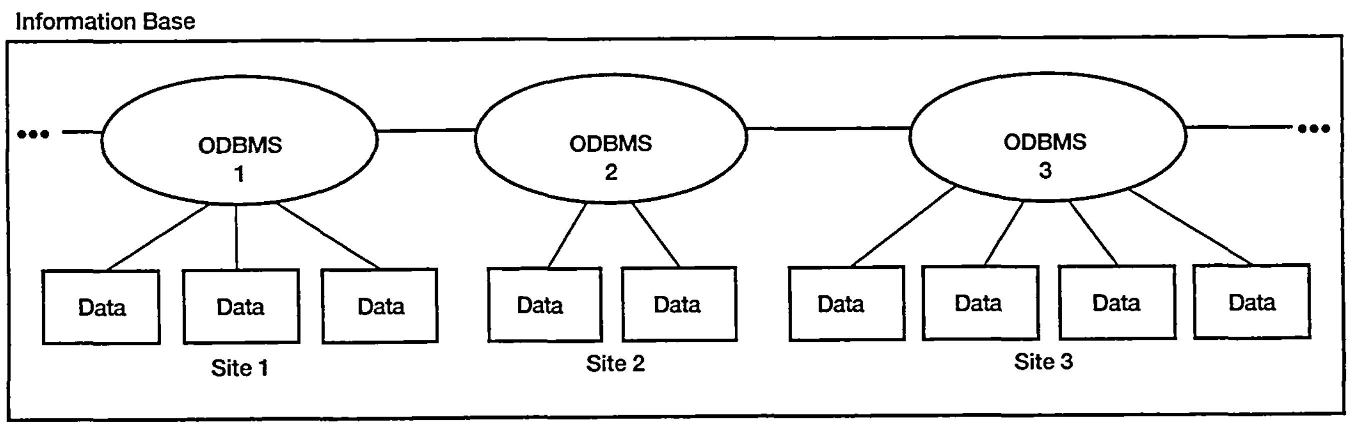


Figure 14.—An OOIB implementation that uses multiple interconnected ODBMSs (or servers).

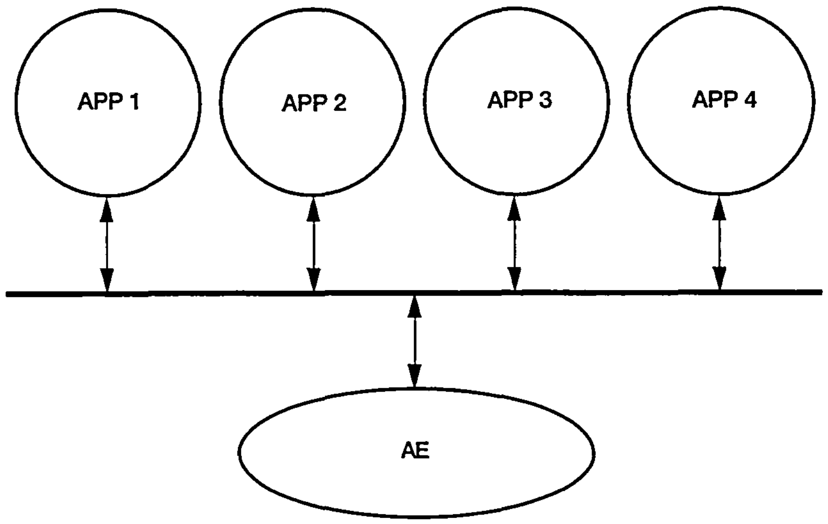


Figure 15.—Application integration & execution.

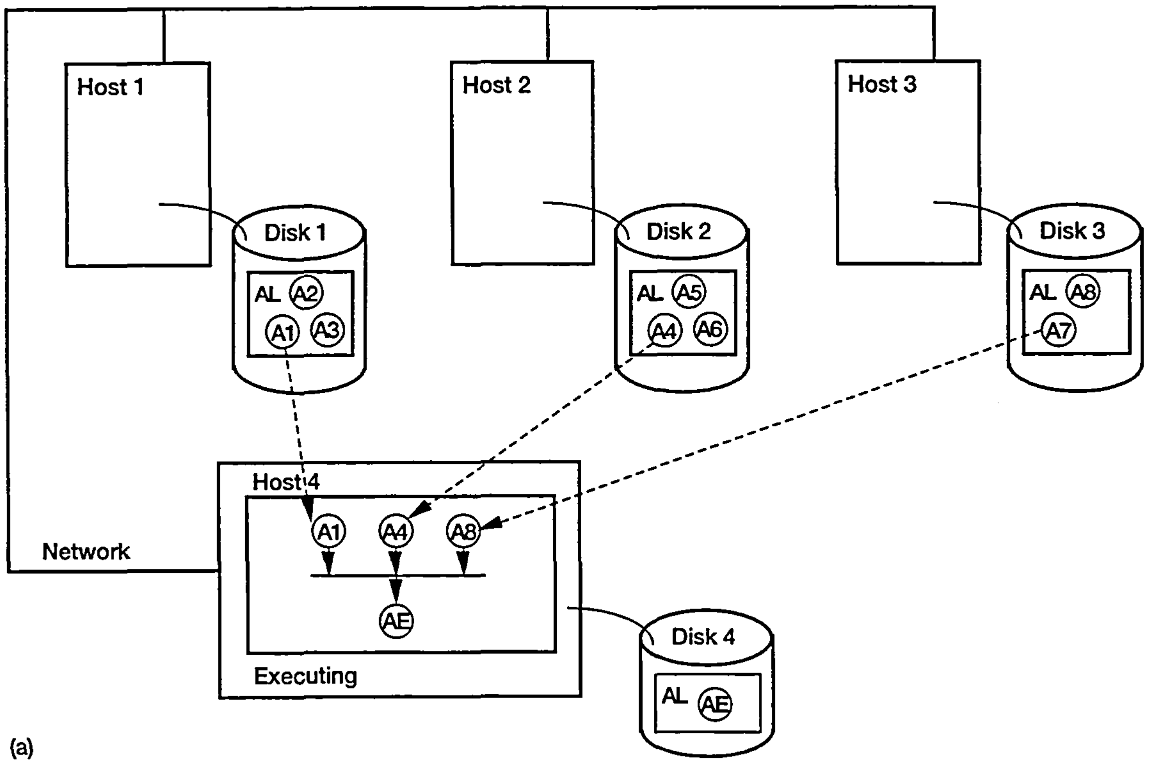


Figure 16.—Application integration in a distributed environment. (a) Applications can be transferred to a single computer and run there.

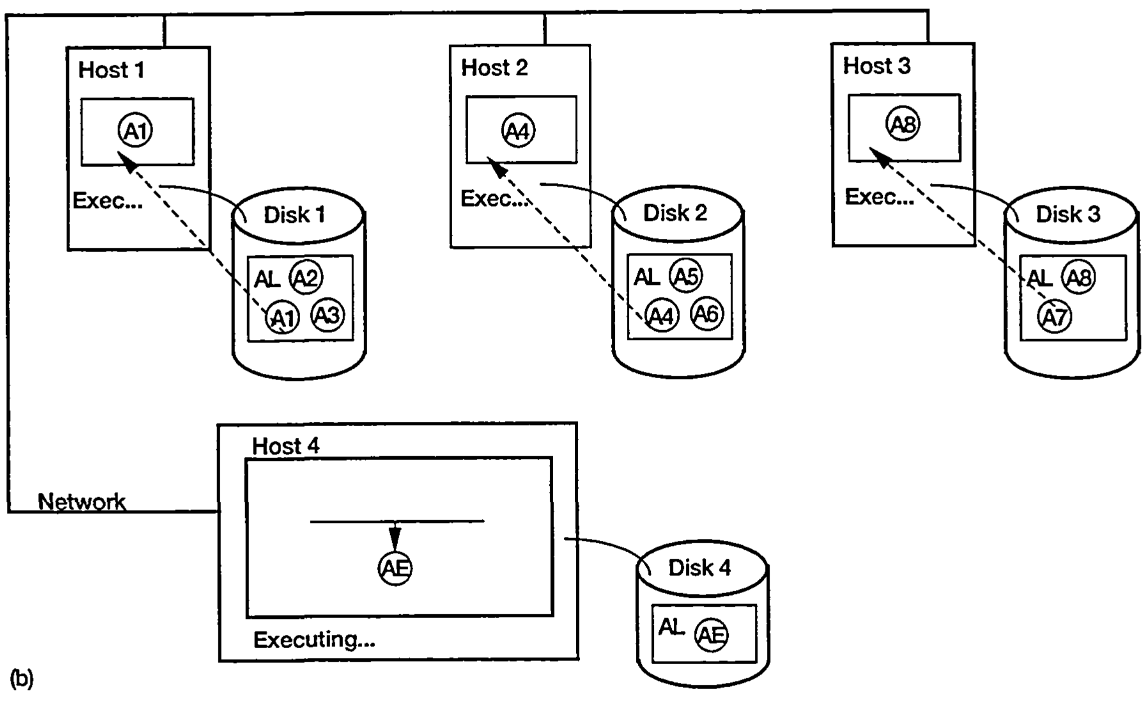


Figure 16.—(b) Applications can also be integrated and run remotely.

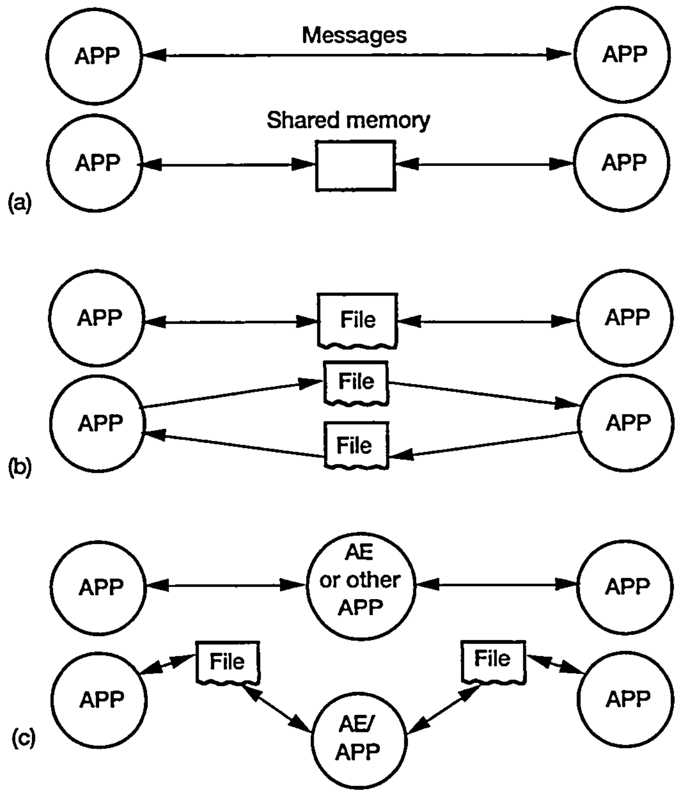


Figure 17.—Methods of application communications. (a) Direct communications. (b) Indirect communications. (c) Assisted communications.

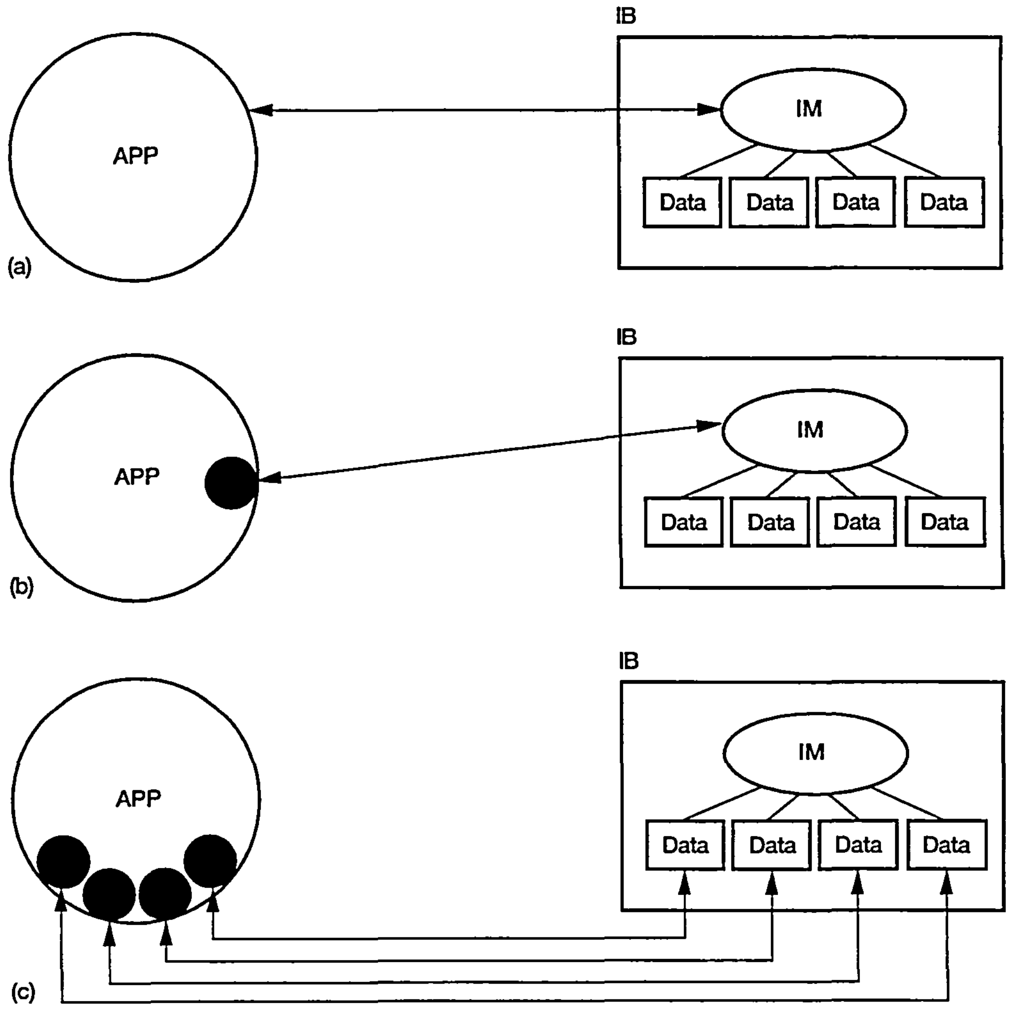


Figure 18.—Methods of accessing the IB. (a) Access via native IM calls. (b) Access via libraries (ADLs) that interface application to the IM calls. (c) Access using implementation-specific interfaces for each data element.

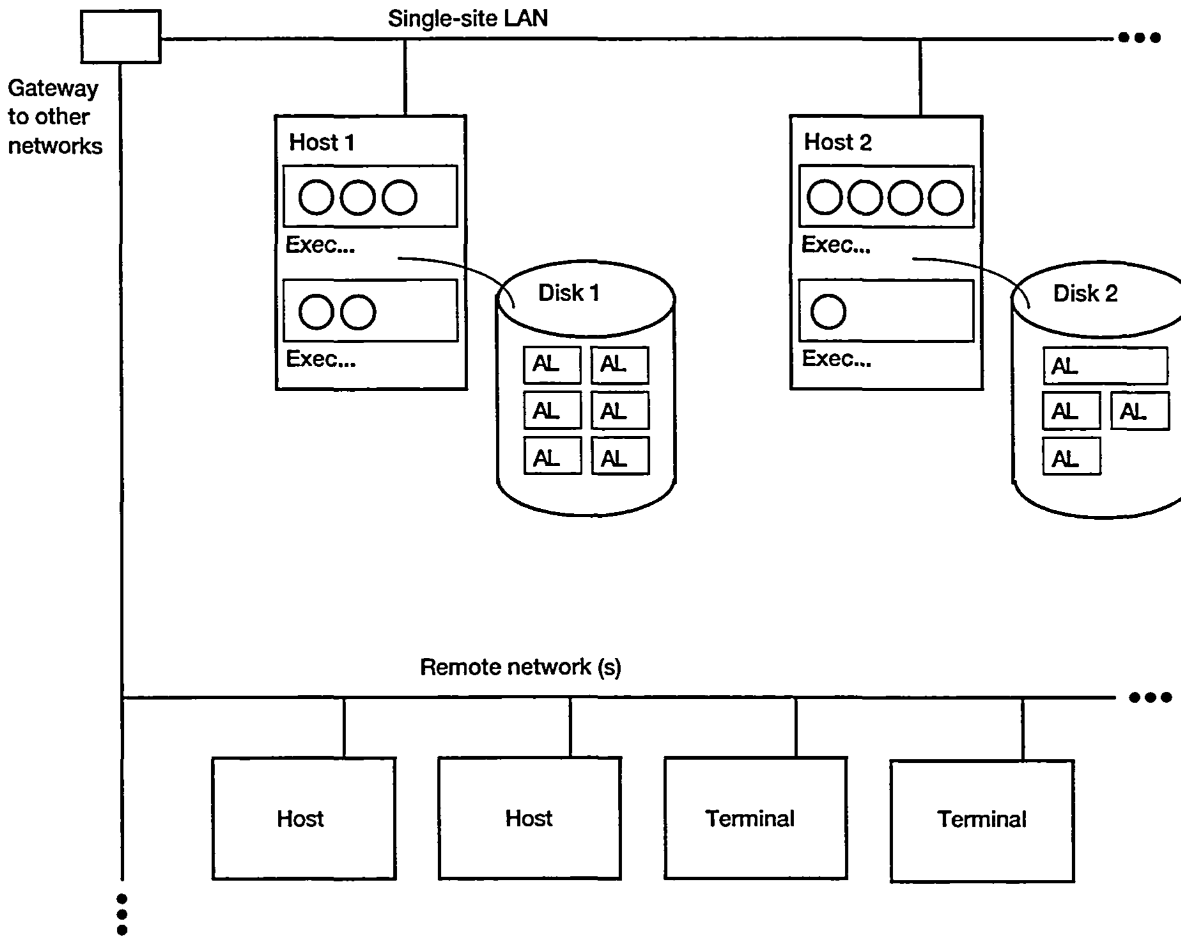


Figure 19.—OSSM software is stored & run at a single site; can be accessed by remote hosts & terminals.

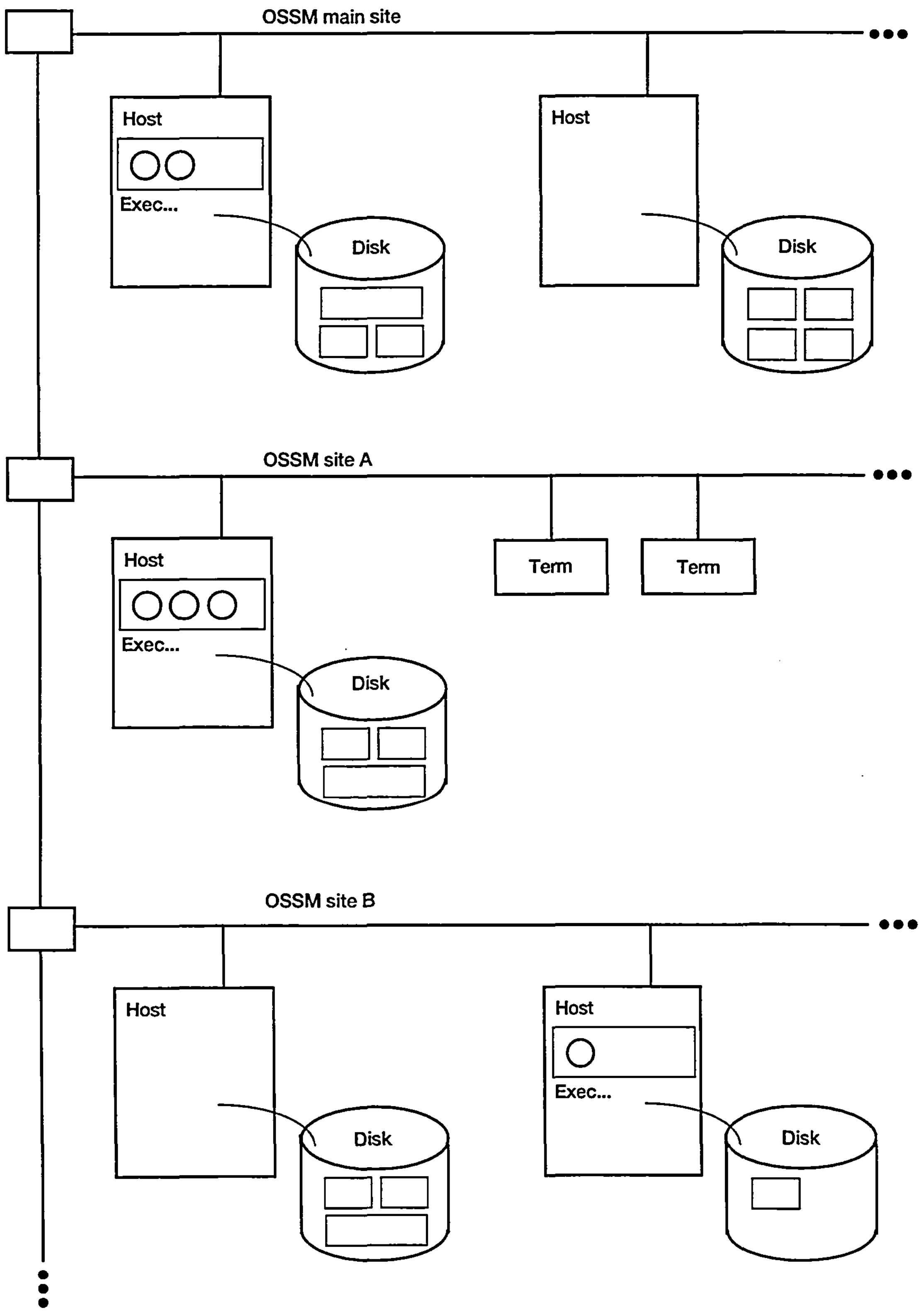


Figure 20.—OSSM software is distributed & run at multiple sites.

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY (*Leave blank*)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 1995	Technical Memorandum

4. TITLE AND SUBTITLE An Open Simulation System Model for Scientific Applications	5. FUNDING NUMBERS WU–505–62–52
6. AUTHOR(S) Anthony D. Williams	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) National Aeronautics and Space Administration Lewis Research Center Cleveland, Ohio 44135–3191	8. PERFORMING ORGANIZATION REPORT NUMBER E–9826
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Aeronautics and Space Administration Washington, D.C. 20546–0001	10. SPONSORING/MONITORING AGENCY REPORT NUMBER NASA TM–107022

11. SUPPLEMENTARY NOTES

Responsible person, Anthony D. Williams, organization code 2620, (216) 433–3611.

12a. DISTRIBUTION/AVAILABILITY STATEMENT	12b. DISTRIBUTION CODE
Unclassified - Unlimited Subject Categories 05 and 62 This publication is available from the NASA Center for Aerospace Information, (301) 621–0390.	

13. ABSTRACT *(Maximum 200 words)*

A model for a generic and open environment for running multi-code or multi-application simulations — called the open Simulation System Model (OSSM) — is proposed and defined. This model attempts to meet the requirements of complex systems like the Numerical Propulsion Simulator System (NPSS). OSSM places no restrictions on the types of applications that can be integrated at any stage of its evolution. This includes applications of different disciplines, fidelities, etc. An implementation strategy is proposed that starts with a basic prototype, and evolves over time to accommodate an increasing number of applications. Potential (standard) software is also identified which may aid in the design and implementation of the system.

14. SUBJECT TERMS	15. NUMBER OF PAGES	16. PRICE CODE
Computer simulation environment; NPSS; Multidisciplinary analysis; Engine analysis design; Computer standards; OSSM	30	A03

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	

National Aeronautics and
Space Administration
Lewis Research Center
21000 Brookpark Rd.
Cleveland, OH 44135-3191