SAND98-2707C

# THE SIMULATION INTRANET ARCHITECTURE

Victor P. Holmes, John M. Linebarger, David J. Miller, and Ruthe L. Vandewart
Sandia National Laboratories
P. O. Box 5800
Albuquerque, New Mexico, 87185
E-mail vpholme@sandia.gov

KEYWORDS
Distributed computing, Intranet, Framework architecture, CORBA, High performance simulations

## ABSTRACT

The *Simulation Intranet* (SI) is a term which is being used to describe one element of a multidisciplinary distributed and distance computing initiative known as DisCom$^2$ at Sandia National Laboratory (http et al. 1998). The Simulation Intranet is an architecture for satisfying Sandia's long term goal of providing an end-to-end set of services for high fidelity full physics simulations in a high performance, distributed, and distance computing environment. The Intranet Architecture group was formed to apply current distributed object technologies to this problem. For the hardware architectures and software models involved with the current simulation process, a CORBA-based architecture is best suited to meet Sandia's needs. This paper presents the initial design and implementation of this Intranet based on a three-tier Network Computing Architecture(NCA). The major parts of the architecture include: the Web Client, the Business Objects, and Data Persistence.

## ARCHITECTURE DESCRIPTION

### Overview

The purpose of the *Simulation Intranet* architecture is to satisfy Sandia National Laboratory's long term goal of providing designers and analysts an integrated set of product realization and virtual prototyping services which include high fidelity simulations in a high performance (HP), distributed, and distance computing environment. The initial focus of the architecture is on the development of a distributed object framework which allows users Web-based desktop access to applications that require high performance distributed resources for modeling, simulation, analysis, and visualization. This framework satisfies the following characteristics:

- works within a heterogeneous, distributed computing environment
- is object-oriented
- is machine-independent and tool-independent (based on open standards)
- exhibits network transparency
- contains components which are largely decoupled
- makes new and existing applications appear as distributed object services
- provides for sequencing, launching, and monitoring of sets of applications
- coordinates archival and retrieval of information
- allows for adding or upgrading capabilities
- provides a consistent, integrated operator interface.

In addition, there are three main thrusts that are addressed by the first version implementation of this framework. First and foremost, the framework architecture must provide users transparent desktop access to the Computational Plant (CPLANT), a massively parallel computing resource constructed of commodity parts. This is accomplished by advancing the use of distributed software object concepts coupled with Web and intranet technologies, and focusing on emerging open standards in these areas. To provide desktop access to such high performance machines, it is necessary to integrate distributed object computing and Web-based computing with HP computing, areas of computer science which have in the past taken separate research paths and now are merging to create a component-based architecture for modeling, simulation, and analysis. Some of the technologies being employing include object request brokers (ORBs), object-oriented databases (ODBMS), and Java Beans, coupled with applications employing the Message-Passing Interface (MPI) and parallel visualization techniques.

The second aspect of the framework architecture involves integrating visualization with analysis in order to monitor progress and potentially inject computational steering. The current model of HP computing normally

This work is supported by Sandia National Laboratories, a multi-program laboratory operated by Sandia Corporation (a Lockheed Martin company) for the United States Department of Energy under contract DE-AC04-94AL85000.

involves long running batch-oriented computations followed by complex postprocessing to finally visualize the results. In some cases, the analysis has to be resubmitted with modified component meshes and parameters. In the Simulation Intranet model, the framework can initially provide image snapshots of simulation time steps to the desktop so the users can determine if the analysis is on track without having to wait until it is completed. The implementation of this capability is described later in the visualization sections. Eventually, the framework will expand upon this *spyglass* concept and allow the user to virtually move around in 3D space to look at the data, potentially interrupt the analysis to tweak the mesh or parameters, and ultimately use the power of the CPLANT clusters to perform real-time parallel visualization of data as it is generated.

The third thrust involves knowledge management through a concept known as the Product Design Object or PDO. This concept includes the ability to capture the large quantities of data generated by these analyses in an object repository, classify this data into some kind of taxonomy which is consistent with the Laboratory's business model, and then make relevant information available to the users to solve new problems and achieve new insights, particularly in the area of nuclear weapon stockpile stewardship.

**Related Work in Distributed Frameworks**

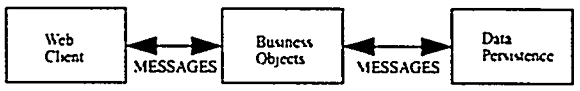
Although this area of research is fairly new, other framework architectures and related services are being developed which contribute to the advancement of the technology. Some of the more recent approaches include CONDOR (Litzkow et al. 1988), Legion (Grimshaw et al. 1997), GLOBUS (Foster et al. 1997), WebOS (Vahdat et al. 1997), and Javelin (Christiansen et al. 1998). The Legion research attempts to provide shared object and name spaces within a fault tolerant environment. Globus may be viewed as an environment that provides a "metacomputer" in the form of networked supercomputers or workstation clusters. It builds a layer of services for communication, resource location and allocation, authentication, and information access. WebOS attempts to provide similar services for wide area applications, including resource management, remote process execution, and security. The Javelin architecture is an all-Java global computing infrastructure which defines brokers, clients, and hosts. A client seeks resources, a host offers resources, and a broker coordinates the supply and demand between clients and hosts. By using Java exclusively, Javelin attempts to circumvent issues regarding user login access and maintenance of binaries across architectures.

With the growing popularity of Java, other researchers are also pursuing Java-based efforts to establish software infrastructures for distributed and distance computing. These include ATLAS (Baldeschwieler et al. 1996), Charlotte (Baratloo et al. 1996), ParaWeb (Brecht et al. 1996), and Popcorn (Camiel et al. 1997). These projects attempt to provide capabilities for parallel applications in heterogeneous environments.The use of Java for building distributed systems will continue to flourish as Internet-based programming becomes more viable.

**Three-Tier Component Model**

The three thrusts or current areas of development discussed above fit nicely into a Three-Tier Network Computing Architecture (NCA) model depicted in the following figure.

Three-Tier Network Computing Architecture



The following sections discuss the general characteristics of each of these tiers. A subset of these characteristics has been implemented within the prototype distributed object framework.

**Web Top** A typical web browser can be viewed as the user's desktop operating environment of the future. The browser provides access to all applications and services required by users to accomplish their laboratory missions. The browser component of a three-tiered architecture is concerned with aspects of presentation without knowledge of business rules. It provides a user-centered, document-centered, coarse-grained, stateless world view.

In the context of the Simulation Intranet, the browser displays the "web pages" which provide access to modeling, simulation, and analysis capabilities. These capabilities can be displayed through numerous web-based technologies such as HTML, Java, Javascript, or ActiveX. The first instantiation of the framework creates Java Bean components which represent the various applications and services for the Simulation Intranet. In addition, an applet serves as the container or "bean box" for graphically programming a simulation sequence, configuring each simulation component in the sequence (customizers, introspection), invoking the sequence (event model), and monitoring its progress. This approach provides a component-based architecture which transforms legacy applications into application components, or "business objects." The enterprise gains new distributed object-based applications from different assemblies of repackaged legacy components.

**Distributed Access to Applications and Services** Transparent connections and communication must be provided between the Web Top and the distributed applications and services. This messaging element of the three-tiered architecture provides the notion of distributed blobs of computing resources which are available to the Simulation Intranet. These resources may be local to a site (LAN) or remotely accessed (WAN), and the messaging interface makes them appear as if they are on the user's local machine. The initial implementation uses the Java Bean event model coupled with event model adapters which can potentially accommodate various protocols, including IIOP (CORBA), RMI, RPC, HTTP, and COM+. These adapters provide the transparent mechanisms for browser-based beans to connect and communicate with the distributed applications and services. Currently the adapters are implemented using CORBA and IIOP.

As an example, a Java Bean component can represent a desktop instantiation of some application code. The bean includes all of the properties needed to configure and initiate execution of the code, as well as visual representations of its component for drag-and-drop programming. The event model associated with this bean includes a series of events for triggering the initialization and execution of the actual legacy code. A CORBA-based adapter associated with this bean serves as a listener for these events. Upon receiving these events, this adapter reacts to them by establishing a connection with and making remote calls to a CORBA wrapper for the actual legacy code which resides wherever the code is to execute, such as the CPLANT. These remote calls are defined by an IDL interface, and their execution is carried out through the use of an ORB and the IIOP protocol. Therefore, the event model adapter, serving as a listener for the application code bean, is written in Java, but the wrapper for the code itself can be written in another language such as C++. The adapter behaves as a CORBA client and the legacy code wrapper behaves as a CORBA server.

**Modeling, Simulation, and Analysis "Business Objects"** The business objects in the three-tiered architecture are the legacy codes and new codes provided by Sandia scientists and commercial vendors which implement the applications and services required by the Laboratory's missions and provided by the Simulation Intranet. These applications should be presentation and data storage independent, usually have connections with other business objects (perhaps through other bean components and adapters), and may need to store and retrieve information from the persistent store. As opposed to the Web client, business objects provide an application-centered, fine-grained, stateful world view.

To achieve the concept of business objects, most legacy codes require wrappers which handle the distributed object aspects of their use. The encapsulation strategy used to wrap legacy codes should be a component-first approach which involves performing a domain analysis to generate an object model, identifying public interfaces for this object model, and encapsulating the legacy application to populate the specific functions of this public interface. A complete wrapper should perform connection protocol management, data translation and information processing, error detection and recovery, and environment management, which includes insulation of the users from changes and upgrades.

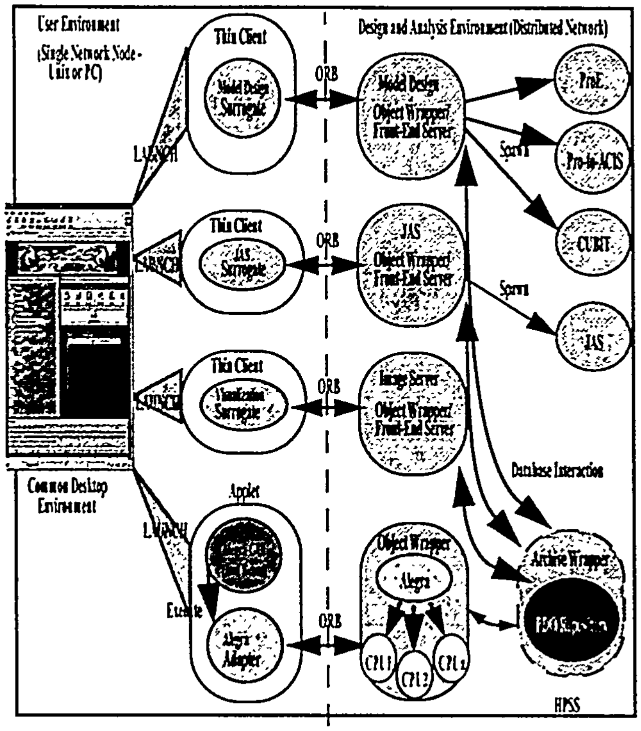
**Knowledge Management** The final tier of the architecture, data persistence, should ultimately take the form of a knowledge management system. The value of Sandia's information assets requires more than just the ability to store and retrieve them in a distributed fashion. It should also be possible to dynamically match information to specific processes or unknown situations and leverage that information to achieve new results and insights into mission-related problems. Some key elements of a knowledge management system include distributed object databases and associated tools for legacy data conversion, knowledge creation analysis, collaboration, web content management, intelligent agent implementation, and visualization.

Some of the functions which transform information into knowledge include capturing data in an object repository and organizing it into a classification framework or taxonomy which reflects the Simulation Intranet business model, the ability to make information available to a knowledge seeker, and the application of that knowledge to solve new problems. Knowledge management solutions should be context sensitive, user sensitive, flexible, heuristic, and suggestive.

## ARCHITECTURAL COMPONENTS

### Overview

A generalized pictorial representation of the architecture is shown in the following figure.


User Environment (Single Network Node - Unix or PC)
Design and Analysis Environment (Distributed Network)
Thin Client
Model Design Surrogate
ORB
Model Design Object Wrapper/ Front-End Server
ProE
Pro-A-ACIS
CUBIT
JAS
Spawn
LAUNCH
JAS Surrogate
JAS Object Wrapper/ Front-End Server
Visualization Surrogate
Image Server Object Wrapper/ Front-End Server
Database Interaction
Common Desktop Environment
Applet
Execute
Alegra Adapter
Object Wrapper
Alegra
CPU1
CPU2
CPU3
Archive Wrapper
PDO System
HPSS

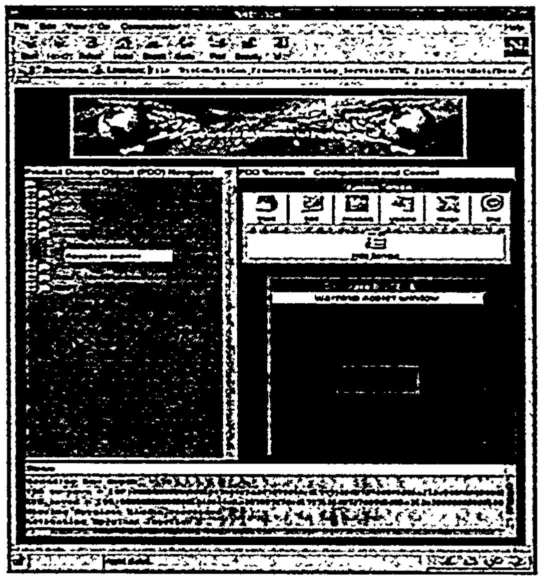
On the left is the desktop environment and on the right are the distributed computing resources which are available. A Java-based applet running inside the browser represents a prototype desktop environment. There is a navigation panel that shows the available products and services being worked on, a work panel that displays the individual GUI's for the various services as well as the images sent back by the visualization service, and a status window. In the context of the framework as described in the previous section, this browser is one of the entities of the three-tiered architecture. The designer uses this common desktop environment to configure, link together, and launch various applications, and each launched application transparently locates, connects to, or acquires the appropriate distributed resources for that application.

For legacy codes, there are wrappers which allow older codes to become network-aware and behave as distributed object services. Wrappers may appear as front-ends to codes which cannot or should not be modified, or there may be tight integration between the wrapper and the analysis code. For new codes, they should be developed as network-based components from the start and would not require wrappers. The wrappers represent the second element of the three-tiered architecture which is sometimes referred to a the business model or business objects. Sandia's business in the framework context is not accounting or payroll, but rather applications such as solid modeling, meshing, finite element analysis, and visualization. Some codes execute on workstations and others are launched on HP massively parallel computers, depending upon the code's requirements and the users' needs.

Finally, the third tier of the architecture at the back-end involves data persistence and knowledge management which is represented as a PDO repository. Both the wrappers and the desktop are capable of communicating with the PDO repository. All distributed communication is performed using CORBA.

## The Web Top

**User Interface Applet and PDO Bean** The following is a prototype display of the Web Top interface.



The user first starts up a browser and navigates to the Simulation Intranet home page where information and documentation are available for the distributed object framework. This home page contains a link pointing to an HTML file which initiates execution of the framework applet that provides the container for the user interactions with the desktop services. The applet performs three initialization functions. It first provides a login dialog which allows the user to enter a username and password. Then it creates the desktop which consists of three panels. The panel on the left serves as a navigation panel and contains a tree of products and services available to that user in the database. The panel on the right serves as a workspace panel where the user can interact with the various services. If an application has a GUI associated with it, this GUI can be displayed within the workspace panel. In addition, images being sent from a visualization service may be displayed here as well. The bottom panel serves as a status display for messages from both desktop and distributed services. After these panels are created, the applet completes its initialization process by instantiating the PDO bean which essentially controls all of the framework's subsequent activities.

The PDO bean first creates a CORBA-based adapter object and uses it to connect to the database. Then it uses the adapter to retrieve all of the PDO's from the database and build the navigation tree from this information. This tree then becomes the starting point for the user. By selecting a product and service from the tree, the user generates an event which causes the PDO bean to instantiate an application or service bean. The names of the beans are available from the database, and therefore, any new application or service can be plugged into the framework by creating a bean for it which adheres to some minimal design patterns, and then providing metadata for the database about the new application or service. The core framework code itself (applet and PDO bean) does not need to be modified. The creation of application and service beans allows the user to configure, launch, monitor, and display results for modeling, simulation, and analysis tasks associated with a product.

**Intranet Beans for Applications and Services**
Each application or service to be accessed through the Simulation Intranet should have a bean implementation associated with it. This bean normally provides the desktop side of the service for the user to interact with. To become a part of the framework, an Intranet Bean should adhere to some minimal design patterns.

First of all, the bean may import the PDO service and implement an adapter class which allows it to communicate with the database and other distributed services. This is not an absolute requirement if the service has no need to communicate with the database or contains its own communication interfaces. The only requirement is that some metadata about the service be resident in the database so that the service will appear on the navigation tree, and in t hat way become accessible to the user. If the bean does wish to communicate with the database and/or to a server which manages the actual service provided by the bean, then the framework provides some simple template classes which simplify the adapter's implementation. The bean event model is used by these templates to communicate between the bean and its adapter. When a user interaction generates an event which requires distributed services, the adapter, acting as a listener for those events, will respond to them and provide the communication needed to complete the transaction.

The second major design pattern is associated with the way the bean displays its services to the user. If a bean has some form of GUI and it wishes to use the applet's workspace panel as a container for the GUI, then the bean must implement a Java interface known as the *FrameworkBean*. This interface consists of a single *instantiate* method which the PDO bean will use to instantiate the application bean. This method enables the PDO bean to convey the handles needed for the application bean to interact with the applet's panels. The PDO bean uses introspection to determine if the application bean implements the *FrameworkBean* interface, and if so, calls that bean's *instantiate* method to start it up. The framework provides templates for implementing the *instantiate* method. This method usually includes creation of the bean's adapter, creation of the GUI, setting up of the event model, and display of the GUI in the workspace panel. Of course, if the bean wishes to have a GUI which is displayed outside of the browser's context, then it need not implement the *FrameworkBean* interface. The only requirement is that it provide a constructor with no parameters so that the PDO bean can instantiate it. Once the bean has been instantiated and its GUI displayed, the user can interact with it to utilize the distributed service that this bean provides the doorway to.

**Bean Adapters for Distributed Communication**
As discussed above, each application bean will normally include an adapter class for distributed communication. There are several reasons why an adapter is preferable to embedding the communication within the bean itself. First of all, the adapter provides a nice encapsulation mechanism for all of the communication code and makes the design more object-oriented in nature. This encapsulation subsequently allows for multiple implementations of the adapter without affecting the bean application code. Such multiple implementations allow for different communication technologies and protocols which may be necessary when switching between distributed computing and distance computing. Another advantage of an adapter is that it can be implemented as a separate thread which prevents bottlenecks from occurring on the desktop. When the user submits a request to the service, it can be carried out by this thread without affecting the user's ability to perform subsequent desktop interactions. Finally, various multiplexing and queuing algorithms can be inserted into adapters, allowing the user to batch requests to a service without having to wait for one to complete before issuing another one.

**Applications and Services**

**Database Service** An object-oriented database is a key component of the framework, maintaining information and methods for each product design and analysis. A CORBA interface is required to the database so that it can be accessed from the desktop or from applications which are running on workstation clusters or high performance computers. This CORBA interface includes a persistent server application (PDO server) which manages the PDO object implementation for the IDL interfaces to the database. This server resides on machines which contain PDO's that may be accessed by the framework. This server creates and returns PDO object references to clients who need to communicate with the database. When a client connects to the PDO server and receives back PDO object references, these references are associated with particular products or product services. A set of IDL interfaces for these PDO's provide the operations needed to initiate user sessions, launch applications and services, retrieve status, and manage the information generated.

**Application Launch Service** Because some of the ORBs used by the framework do not provide an automatic launching capability of application servers, it is necessary for the framework to provide this so that desktop users do not have to deal with such details. For the short term, this capability will be provided by a simple CORBA-based Launch server which must be persistently active on machines where applications may be run within the framework. This server manages an application launching object which implements a simple IDL interface containing a single launch operation. This operation accepts as input a string containing the command line required to start an application. The operation uses this string to perform a spawn of the application executable. Obviously this spawning action is operating system specific. The framework currently only supports the launching service on Unix-based systems. The launching operation throws an exception and informs the database if it detects that the launch call fails. Normally, the launching operation will start up a CORBA-based server which manages an application object. This application object or wrapper serves as a front-end to an application code that may execute on an HP computer.

**Alegra Application Service** The Alegra application is a structural dynamics finite element code written in C++. It was selected as the first candidate application for the framework. It had to be made CORBA-aware so that it could be accessed in a distributed fashion from the desktop using the Web Top aspects of the framework. Therefore, an IDL interface was defined for Alegra operations, and the object implementation of this interface, or Alegra wrapper, serves as a front end to the Alegra software. This wrapper is managed by an Alegra server which is launched by the Launch server whenever a desktop client requests an analysis session that uses Alegra. There is one Alegra server launched per session so that each user has their own exclusive copy of the Alegra software when performing an analysis.

The IDL interface for the Alegra wrapper object itself simply provides operations for starting an execution, retrieving status of the execution, and aborting or terminating an execution. The execute operation allows the desktop client to begin an analysis. This operation normally initiates an HP computing cluster process known as "yod"to start a parallelized Alegra run on compute nodes available in a CPLANT cluster. The execute operation also makes requests from the PDO service to determine where the execution is occurring, since the startup process may be different for workstation clusters than it is for HP machines. This operation also retrieves the parameter file and data files from the PDO before starting the execution. The parameter file resides in the database, so it must be retrieved and written as an ASCII file to the local file system where it will be accessible to the legacy code execution. Only the paths to the data files are stored in the database. These files must be available on the local CPLANT file system as well. Once all the information has been gathered, system calls and scripts can be used to launch the actual application. If the application is to be run on an HP computer, the system calls will start yod with the appropriate parameters for the Alegra execution. In addition, since the only mechanism currently available for getting back Alegra status is through yod's stdout, this execute function also includes the launching of a status process. The stdout of yod is piped into the stdin of the status process such that status messages coming back from Alegra may be interpreted and acted upon by the status process.

The status operation in the Alegra wrapper allows the desktop client to find out the status of an application's resource allocation and execution state. The wrapper can use the services of the HP computer to obtain this information and send it back to the desktop as part of this operation. Normal status of an application, such as its current time step if it involves a time-based simulation, will be available to the desktop from the database which stores messages sent to it by the status process.

Finally, the abort and terminate operations allows the desktop client to abort a run and terminate the service respectively. This assumes that the wrapper has some way of communicating with the compute nodes or yod to abort the execution. No restart capability is provided by the framework for this version.

**Application Status Process** This process is started by the application wrapper and waits on stdin for messages coming back from an application, usually through yod's stdout. Upon receiving a message, it must parse the message and determine the appropriate action to be taken. This process is a client to the PDO server and a client to the visualization snapshot service. If periodic status messages come back from an application, this process forwards them to the database. If a message comes back indicating that a data file is ready for visualizing, then this process sends a message to the visualization service indicating where to locate the data. This process connects to the PDO server through some well known port or naming service. It obtains an object reference for the visualization service by requesting it from the database which records status information and object references that contain knowledge of when the visualization service is running and ready to accept data for a particular analysis. If the status process starts before the visualization service and is unable to obtain an object reference for the visualization service on startup, then the status process must attempt to request it again when the first data file is available for imaging. If the status process successfully makes a connection, it can request the visualization service to process the data. If it cannot make a connection, the data files will still be available for post-processing after the run but there will be no quick look available.

**Visualization Snapshot Service** This service handles the quick look capability which allows the user to receive a snapshot image of a currently running application. This service is configured from the desktop and launched by the PDO server as another application associated with a particular lifecycle analysis. It is a CORBA-based server and performs initialization functions similar to any application server. It connects to the PDO server, creates a visualization application object, stringifies that object, and sends a message back to the PDO indicating it is ready for data. It also sends back its stringified object reference to the database which can return it to any application that communicates with the visualization service. Normally there would be only one client associated with a particular visualization service, and that client could be the application status process which knows when data is available for processing. In addition to these standard init procedures, the visualization service also requests from the PDO server an object reference which communicates with a desktop bean which displays an image within the browser. The visualization server should request the desktop object reference first from the PDO before attempting to create the visualization application object. That way, this bean object reference can be passed into the constructor of the visualization application object so that it will have the means to communicate with the desktop when it has an image ready to send. Once initialized, this server enters its event loop and waits for messages from the status process. Upon receiving a message, this service retrieves the data, post-processes it, converts it to an image, and sends it to the desktop for display. Then it goes back and waits for the next available data and iterates on this sequence of steps.

## Knowledge Management: Product Design Object (PDO) Concept

To achieve the goal of building the capacity to perform high-end simulations without relying on single vendors to produce machines or software, and to relieve the users of the necessity of tracking configuration changes and remembering system details and commands, there must exist a facility to insulate most users from this detailed information about the current configuration and composition of the system: one element of this facility is the Product Framework (PF). The PF defines the product-centric viewpoint of the system and will reduce the need for an engineer or scientist to have current detailed system usage information. These details include specifics of the system architecture and the current configuration of files, environments, data-sources, tools and all the other by-product details that are necessary in order to get to the real purpose of the analysis -- getting the product built. This need is exacerbated by the potential necessity of maintaining a number of binaries and libraries for particular software due to the use of heterogeneous systems and by the utilization of remote, unfamiliar platforms as they are made available through the Distance Computing effort. With the pursuit of highest application performance and the subsequent loss of transparency, the product framework becomes even more basic to the successful deployment of the CPLANT concept, particularly when execution parameters are dependent on datasets.

The PF interfaces with the Simulation Intranet using CORBA or other products that may be supported in the future. This interface provides classes, methods and other components and infrastructure to implement a Product Design Object (PDO). A PDO is a specific instance of product information and methods created for a particular product. The PF supplies persistent, available, and flexible storage for a PDO from its creation through the production and maintenance of the product. This repository provides a valuable archive of product development information and meta-data relating to the PDO instance and its components.

Through the PF, individual PDO instances can be linked together to do regression testing on software modifications and upgrades, thus requiring the PF to act as the first-line user to system and application development personnel. Products will be analyzed with alternative software to not only validate the composition and construction of the product but also to validate and calibrate the analysis software.

The main purpose of the Product Framework is the construction and maintenance of the Product Design Object. The Product Design Object is a complex persistent object that captures the design state, recipe, requirements, results, toolkit elements, and access information for a legacy or in-development product. It provides an easy-to-use and globally available interface by which a PDO instance can be created or accessed for utilization or update. A PDO instance includes not only the specifics necessary to locate any information relating to the product but also includes methods to actually do the access or processing of the particular aspect of the PDO, such as starting the program launcher. It will locate and deliver necessary inputs to the application and receive and retain outputs from that application. Designers, analysts and developers, each from their differing viewpoints, make use of the PDO to record and deliver the detailed and changing information necessary to integrate data, procedures, and locations necessary for executing complex software applications, documentation of decisions made, and lessons learned.

Because the classes and methods for each PDO are be maintained by the PF, all pertinent PDO instances can be updated in one step when information about a location, an access method, or a resource becomes obsolete or is augmented. Shared objects are supported wherever feasible by the PF, and reuse of objects created for support of PDO instances is facilitated. The use of shared objects enables certain information to exist in only one instance and thus require only one update to update objects referencing the shared object. An example of a shared object is the Server object which encapsulates information about a particular server and contains links to Application objects that it supports and to Service objects which have knowledge and methods necessary to execute the Application on the Server. A Session links a User, a Product, its required environment, and possibly documentation and help files, with one or more Service instances and maintains the status of an in-progress analysis. Minimizing the impact of system upgrades and repartitioning, new development, new hardware, and other changes inevitable in a dynamic, developing system, leverages the gains made by reuse of the object hierarchy and provides data management support as the systems evolve and advance.

Preferences of the user, characteristics of the product, and the status of the infrastructure might all have a role in determining the particular choice for a resource to perform an analysis at a particular time and place. For instance, if a user on node "a" wishes to execute a program on node "b" with a portion of the data referenced by the PDO, and this data resides on a node other than "a", it is not cost-effective to relocate that data to "a" before finally relocating it to "b". The methods utilized must make intelligent determinations for data transfer or inform the user if there is not a satisfactory alternative given the current constraints. Other decisions that must be made might include whether the capture and storage of interim datasets and products or their re-creation is more appropriate and a strategy for releasing products which are antiquated or when additional storage is needed.

To implement the PDO, the schemas necessary to support the PDO were created in an object database management system (ODBMS). This design effort defines the hierarchy of classes and descendent classes to support instances of the PDO. An ontology of the classes and components of the PDO is being specified utilizing information gained by examination of the needs of users, designers, and application developers. The use of an ODBMS facilitates the evolution of the PDO Schema as more details and requirements are discovered. Database schema evolution is a vital part of the PDO instance's required flexibility. As new entities are defined (new software, data formats, file storage, component types, hardware, etc.) the PDO must be modified to support the new entities without invalidating already existing PDO instances. The new information must be incorporated seamlessly into the existing PDO, preserving useful information with an implementation of versioning.

## Visualization Services

One of the key visualization services provided is the Spyglass capability which was introduced earlier. This enables the user to view an early result of the computation before too much time is invested in an unpromising simulation. It consists simply of the Visualization Spyglass Server, which is started by the corresponding generic Launch Service running on the visualization machine. Launched at the same time as the simulation model server, it awaits a signal from the model server that a consolidated simulation timestep data file has been created. Since the file systems of a service node in a CPlant cluster can be NFS-mounted by the visualization machine, the file generated by the analysis code is simply read in, the geometry of the requested objects is extracted and rendered into an off-line frame buffer (using the Mesa graphics library, which is OpenGL-like), and a JPEG image is created from the framebuffer. This image is sent both to the PDO and to the Webtop (again, if it is active). Note that the decision was made to send an image to the desktop, not the extracted geometry from the data file, in order to reduce both network traffic and computational load on the desktop. Also note that JPEG was chosen as the image format because it is natively recognizable both from a Web browser and from a Java environment, and unlike the GIF format, it is not restricted to 256 colors.

At this point, one of two events can happen. If the Webtop is not active, or does not respond to the receipt of the JPEG image from the simulation timestep data file, the Visualization Spyglass Server simply awaits the signal from the model server that the consolidated data file from a subsequent timestep is available, and repeats the off-line rendering and JPEG image extraction process. If the Webtop is active, it can perform "remote navigation" and request that the image be regenerated from another perspective (i.e., camera position). A Java3D-based object is present on the Webtop that contains a bounding box of the objects in the image (based on the maximum and minimum x, y, and z coordinates of the geometry, which are sent along with the image by the Visualization Spyglass Server), and a set of canonical x, y, and z axis vectors. Six degree-of-freedom navigation is possible within that Java3D object, and the bounding box can be repositioned accordingly. When the desired point-of-view of the bounding box is reached, the viewpoint transformation matrix is extracted and sent to the Visualization Spyglass service, which uses it to reposition the camera and render another JPEG image from that perspective. This remote navigation process can continue until the Webtop user is satisfied that the simulation is progressing properly, or until the image from a subsequent timestep is received.

## CONCLUSIONS

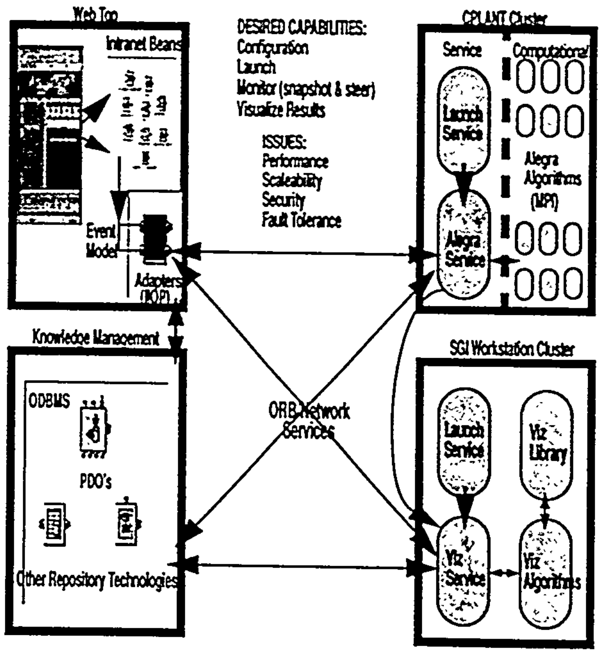
### Current Status of the Framework

The framework is currently in an initial prototype phase. The architecture was formulated after numerous discussions with stakeholders and after performing use case analysis and storyboarding. A focused problem domain for the first implementation was then developed which involves providing desktop access to the Alegra finite element code executing on a Linux-based CPLANT Miata HP cluster. In addition, the framework provides a visualization capability which feeds back snapshots of data generated by Alegra at each time step.

To implement the focused problem, it was first necessary to perform a capability analysis on Linux-based ORBs because Linux is the CPLANT Miata cluster operating system, and most mainstream commercial ORB vendors do not currently support Linux as a platform. An evaluation criteria was established, the best candidates were tested, and the results produced the "best" ORB for framework development. The ORB selected for initial use is called ORBacus from Object-Oriented Concepts, Inc.

It was also necessary to make some decisions on environment and tools. The desktop is currently implemented with Netscape. Java and Java Beans (JDK1.2 Beta4, Swing Set 1.1Beta3, Java3D), and CORBA (Java ORBacus 3.0.1). The PDO environment utilizes the Versant ODBMS version 5.0.8 with a CORBA server acting as a front-end. The Linux-based CPLANT service nodes are where the CORBA wrappers reside for applications which run on the massively parallel clusters. Finally, an SGI workstation cluster is used for visualization, including the use of the Mesa software for offline rendering.

The following figure illustrates the capabilities provided by the first framework prototype.



Again, the three-tiered architecture is evident, consisting of the Web Top, computational resources or business model, and knowledge management. On the Web Top are the components using Java applets and Intranet Beans to configure, execute, monitor, and display results of an Alegra run. Coupled to the beans through the bean event model are the adapters which handle the underlying distributed network protocol. For this prototype, IIOP/CORBA is used to communicate with application wrappers and the PDO. For long distance computing, these adapters could be replaced with some other technology for handling wide area networks or ATM switches. The PDO repository is used to build the navigation tree on the desktop which depicts the products and services available to the user. The PDO database is also used to coordinate all of the activities associated with a particular product development cycle. On the CPLANT side, a launch service is used to automatically launch application wrappers, and wrappers have been developed for the Alegra and Visualization services. The Alegra service is actually a front-end to the native Alegra code and has the capability to call CPLANT services to load the code into the computational nodes, start execution, and monitor status. The Visualization service is a CORBA-based server which encapsulates visualization algorithms for offline rendering and generation of JPEG images. Because the capability does not yet exist to allow direct communication of data in parallel from applications to visualization codes, Alegra generates files on shared file systems, and then a CORBA-based Alegra status task is used to notify the Visualization service when data is available for rendering. When the image is completed, CORBA is again used to ship it back to the desktop where it is displayed.

## Future Work

There is obviously much work remaining in the development of the framework and the overall Simulation Intranet architecture. The framework needs to be extended and productionized for actual mission-based use. To do this more easily, reuseable design patterns are being documented which can be applied to extrapolating the framework for other applications, services, or problem domains. In addition, further research and development will involve incorporating fault tolerant features, investigating impacts of distance computing on some of the framework's architectural components such as the adapters and network protocols, looking at performance, scalability, security, and other distributed resource management (DRM) issues, and continuing the work on the PDO concept.

## REFERENCES

Baldeschwieler, J. E., R. D. Blumofe, and E. A. Brewer, ATLAS: An Infrastructure for Global Computing, *Proceedings of the Seventh ACM SIGOPS European Workshop on System Support for Worldwide Applications*, 1996.

Baratloo, A., M. Karaul, Z. Kedem, and P. Wyckoff, Charlotte: Metacomputing on the Web, *Proceedings of the 9th Conference on Parallel and Distributed Computing Systems*, 1996.

Brecht, T., H. Sandhu, M. Shan, and J. Talbot. ParaWeb: Towards World-Wide Supercomputing, *Proceedings of the Seventh ACM SIGOPS European Workshop on System Support for Worldwide Applications*. 1996.

Camiel, N., S. London, N. Nisan, and O. Regev. The POPCORN Project: Distributed Computation over the Internet in Java, *6th International World Wide Web Conference*, April 1997.

Christiansen, B. O., P. Cappello, M. Ionescu, M. O. Neary, K. E. Schauser, and D. Wu, Javelin: Internet-Based Parallel Computing Using Java, Department of Computer Science, University of California, Santa Barbara, 1998.

Foster, I., and C. Kesselman, Globus: A Metacomputing Infrastructure Toolkit, *International Journal of Supercomputer Applications*, 1997.

Grimshaw, A. S., W. A. Wulf, and the Legion team, The Legion Vision of a Worldwide Virtual Computer, *Communications of the ACM*, 40 (1), January 1997.

Litzkow, M., M. Livny, and M. W. Mutka, Condor - A Hunter of Idle Workstations, *Proceedings of the 8th International Conference of Distributed Computing Systems*, June 1988.

Vahdat, A., P. Eastham, C. Yoshikawa, E. Belani, T. Anderson, D. Culler, and M. Dahlin, WebOS: Operating System Services For Wide Area Applications, *Technical Report CSD-97-938*, UC Berkeley, 1997.

http://www.cs.sandia.gov/discom/