SAND2000-2071C

Evolving the Web-based Distributed SI/PDO Architecture for High-Performance Visualization

Victor P. Holmes, John M. Linebarger, David J. Miller, Ruthe L. Vandewart

Sandia National Laboratories
P.O. Box 5800, Albuquerque, NM 87185 USA
E-mail: {vpholme,jmlineb,djmille,rlvande}@sandia.gov

Charles P. Crowley

Computer Science Department, University of New Mexico
Albuquerque, NM 87131 USA
E-mail: crowley@cs.unm.edu

KEYWORDS

Web-based simulation, scientific visualization, distributed component architecture, Java, CORBA, multi-tiered architectures

ABSTRACT

The Simulation Intranet/Product Database Operator (SI/PDO) project has developed a Web-based distributed object architecture for high performance scientific simulation. A Web-based Java interface guides designers through the design and analysis cycle via solid and analytical modeling, meshing, finite element simulation, and various forms of visualization. The SI/PDO architecture has evolved in steps towards 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. This paper describes the continuing evolution of the architecture to provide high-performance visualization services.

Extensions to the SI/PDO architecture allow web access to visualization tools that run on MP systems. This architecture makes these tools more easily accessible by providing web-based interfaces and by shielding the user from the details of these computing environments. The design is a multi-tier architecture, where the Java-based GUI tier runs on a web browser and provides image display and control functions. The computation tier runs on MP machines. The middle tiers provide custom communication with MP machines, remote file selection, remote launching of services, load balancing, and machine selection. The architecture allows middleware of various types (CORBA, COM, RMI, sockets, etc) to connect the tiers using adapters. The system allows for adding and removing of tiers depending upon the situation. Testing of constantly developing visualization tools can be done in an environment where there are only two tiers which both run on desktop machines. This allows fast testing turnaround and does not use compute cycles on high-performance machines. Once the code and interfaces are tested, they are moved to high-performance machines, and new tiers are added to handle the problems of using these machines. Uniform interfaces are used throughout the tiers to allow this flexibility. Experiments test the appropriate level of interface: either a large set of specific function calls or a small set of generic function calls. This architecture is based on the goals and constraints of our environment: huge data volumes (that cannot be easily moved), use of multiple middleware protocols, MP platform portability, rapid development of the visualization tools, distributed resource management (of MP resources), and the use of existing visualization tools.

1. INTRODUCTION

High fidelity full physics simulations require the fastest available computers. These machines contain protected environments and are often hard to use because of compromises made for performance. Historically at Sandia, users access these machines directly and follow their rules to run top-of-the-line simulations. Recent developments in networking, interfaces, and distributed computing are helping to improve this situation. We are developing a system to deal with these issues. We previously described the SI/PDO architecture [1,2]. In this paper we will

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.

DISCLAIMER

describe continuing development of the architecture to support visualization of simulation results.

We started with the following constraints:

- The visualization tools currently work though a socket interface.
- The tools are under constant development; this means that capabilities and interfaces are subject to change; the architecture must be flexible enough to allow convenient testing of tools and interfaces.
- The tools must be capable of running on large MP machines; these machines vary as to the method of communication with the outside world, how jobs are launched, etc.
- We need to support upcoming developments at Sandia to provide more flexible access to very large data sets.
- We need to work with systems under development that provide distributed resource management.

These are the goals we hope to accomplish:

- Provide a framework for organizing visualization tools.
- Develop an architecture that would be useful for a range of tools; this architecture should be useable in a range of environments, from testing, to the use of large MP machines; it should be flexible and adaptable to changing tasks and network conditions.
- Make tools available on the web.
- Allow persistent sessions.
- Provide services to the visualization tools, such as file selection, display data file meta-data, and automatic server launching.

2. SIMULATION VISUALIZATION PIPELINE

In this section, we give some background that will explain why and how we are distributing the processing steps of the simulation and visualization process. A simulation visualization pipeline consists of five stages, as shown in Figure 1. The first stage is the simulation process in which simulation data is created. The other stages extract visualization-related data from the simulation database, create geometry from that extraction, render the geometry into an image, and allow interaction and navigation through the geometry. Each stage in the pipeline represents a data reduction from the previous stage. The stages are logically separate, such that each could be performed on a different computer. In practice, some or all of the stages are usually combined. Traditional post-processing visualization at Sandia performs the last four stages on a single machine, often a multiprocessor Silicon Graphics workstation with large amounts of shared memory, as in Figure 2.

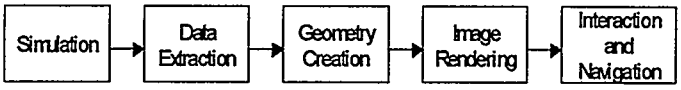

Figure 1: Simulation Visualization Pipeline

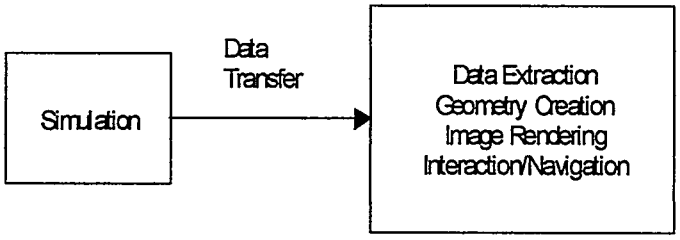

Figure 2: Traditional Post-Processing Simulation Visualization Architecture

The traditional approach breaks down as simulation data volumes increase. The large amount of geometry created swamps the capacity of the client workstation. Figure 3 depicts an alternative architecture used in simulations with large data volumes. Essentially, most of the visualization occurs on a "visualization engine" machine, with only the resulting image being transmitted to the client workstation. This packaging of stages forms the basis for the simulation visualization architecture presented in the body of the paper.

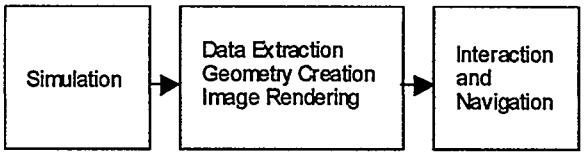

Figure 3: Big Data Visualization Architecture

3. RELATED WORK

Because internet technologies are maturing, there is a large amount of related development occurring in areas that involve bringing distributed computing resources and software tools to the user through the web. The following discussion includes only a few such examples that focus on visualization. An early example of web-based visualization is presented in [3]. The visualization pipeline model is evident in this work, which, through a visualization web server, provides the viewer control of the pipeline through CGI forms, and produces VRML images as output. In

[4], an entire visualization package implemented as a Java applet is discussed, with the benefits of portability, ease of use, and interactivity, but poor performance. In [5], an applet is also utilized, but only as a front-end to a set of environmental vis tools which are linked to a database that can gather real-time information from sensors and feed them to the browser. In [6], the pipeline is implemented as a set of C++ and Java components that are connected using CORBA to provide dynamic updates and a high degree of user interactivity. In [7], Java3D technology is incorporated in applet front-ends to provide a portable viewer and the ability for users to peer into the state of a simulation at a place, perspective, and time of interest. Finally, the work described in [8] also contains a layered model to make existing vis systems web accessible, with layers for the browser, authentication, problem setup, listening daemons, application launchers, and the legacy applications. Many of these concepts are embodied in the architecture discussed in this paper.

4. THE INITIAL VISUALIZATION API

Existing Sandia vis tools have already been modified to use the Model-View-Controller architecture. A GUI (the view) can run on one machine and the central computational core (the model) can run on another machine (as shown in Figure 3 above). The GUIs, however, are implemented with various technologies such as Tcl/Tk and AVS, and they are not easily accessed via a web browser. We standardized on a Java-based web interface.

The view and the model communicate via sockets using a home-grown socket library (called the *comm* library). It provides a level of abstraction from sockets that can be taken advantage of in converting to a new architecture. By replacing the library, we can change the communication technology from sockets to something else (like CORBA) without changing any of the source code in the visualization tools. The interface has the following functions:

```
int Comm_ServerInit(int port);
void Comm_WaitForClient();
int Comm_ConnectToServer(char *host, int port);
void Comm_SendFlagMessage(int TYPE);
void Comm_SendStringMessage(int TYPE, char *s);
void Comm_SendByteArrayMessage(int TYPE, int size,
char *byte_array);
void Comm_SendIntMessage(int TYPE, int i);
void Comm_SendIntArrayMessage(int TYPE, int size,
int *int_array);
void Comm_SendFloatMessage(int TYPE, float f);
void Comm_SendFloatArrayMessage(int TYPE, int size,
float *float_array);
void Comm_SendDoubleMessage(int TYPE, double d);
void Comm_SendDoubleArrayMessage(int TYPE, int
size, double *double_array);
void Comm_GetMessageHeader(int *TYPE, int *size);
void Comm_GetMessageData(void *data);
```

Both the client and the server can send and receive data of various types using these functions once the connection is made between them. Each function has a TYPE field and a data object (or array). The receiver receives the message header first which indicates the type and length of the message data.

5. TWO INTERFACE STYLES

Currently the visualization tools use the comm library to implement one of two styles of interface. The first is a specific message protocol that uses the message types to specify the desired function. These interfaces often have 50-60 message types. The messages are not functions but rather are one-way messages. Some messages cause a return value to be sent in a subsequent message. For example, the interface to an isosurface visualization code currently uses this first style and contains about 60 functions. Some example are: send directory contents, choose file, send transformation matrix, processor count, starting time step, ending time step, time step increment, number of isosurfaces, dummy value, number of subsets, variables to isosurface, disambiguate flag, surfaces values, etc.

The second interface is a smaller, more functional interface that uses only five calls. The calls are:

```
Attribute GetAttribute(int attribute);
Void SetAttribute(Attribute attribute);
Void TakeAction(int actionID);
ImageInfo GetImageInfo();
TriMeshInfo GetTriMeshInfo();
```

In this interface we use the model that there are a large number of attributes that can be set and fetched. We have generic set and get functions and the first argument determines the specific attribute to set or get. It is easy to add new attributes without changing the interface. You can also invoke a number of actions. These actions are computations like creating a new image. Using these primitives, you can build up a multi-argument function call by setting an attribute for each argument and then calling TakeAction to invoke the function. Two attributes are important enough to have their own functions: an image and the tri-mesh information. This second style of interface works better with the architecture we are using and so we are converting all the visualization tools to use this interface. We are investigating the idea of converting to a third style of interface. This will be discussed later in the paper.

6. THE ARCHITECTURE

Our Development Path

The first change we made was to provide a consistent web-based interface to the visualization servers. We are implementing the GUI using Java applets on a web browser using the Swing components. The user interface allows you to choose the visualization server you wish to connect to, and panels appropriate to the visualization tool selected are created when you connect to it.

We are adding one or more tiers in between the GUI and the visualization server. These tiers add services such as remote file system browsing, browsing of simulation file meta-data, automatic launching of servers, load management on the server machines, and simplified communication with MP machines.

The N-Tier Model

Our visualization architecture is based on the N-tier model. The architecture will always include two levels, the GUI and the visualization server, but usually it will contain three or more levels. The GUI will run as a Java program in the web browser. In the two-tier version, the two tiers communicate by sending socket-based messages such as `SendIntMessage`, `SendFloatArrayMessage`, etc. In order for these messages to make sense, the sender and the receiver must agree on the meaning of each message type. One of the messages is "Get directory contents" and is used to browse the file system on the server in order to select a file to use. Another message is "Send isosurfaces value" which sets a value to indicate the isosurface of interest. The server will respond to this message by recomputing the model, rendering it, and sending the rendered image back to the GUI level.

This works fine if the server can communicate with sockets. But suppose that the server is running on an MP machine that cannot communicate with the outside world but can only communicate with a service node on the MP machine. In this case we need to add a third level to the architecture.

A Purely Switching Middle Tier

We will call this middle level the *command processor*. Somehow the command processor and the server must be started on the MP machine. This might be done manually. In more advanced versions of our architecture, the command processor will know how to start the visualization server and will do it on command from the GUI.

The GUI will start up and the user will tell it to connect to the command processor (using sockets). The command processor will then connect to the visualization server. From then on, the command processor acts only as a switch. Messages from the GUI are passed through to the server and messages from the server are passed through to the GUI. For example, if the command processor receives a message SendIntMessage, it will turn around and use SendIntMessage to pass the message on to the server. The command processor will not interpret the message or even look at the type of the message. It will treat all messages the same. The main advantage of this version is that the command processor encapsulates the knowledge of how to communicate with the server on the compute nodes. The GUI cannot communicate with the server directly so it delegates the task to the command processor.

Adding Functions to the Middle Tier

We have some older MP machines where the compute nodes can only communicate with the service nodes (and hence the outside world) using standard input and standard output. This can be handled using this version of the architecture. First the server must be changed to use standard input and standard output for communication. This is done by replacing the comm library with another version. This version will, for example, implement `SendIntArrayMessage` by converting the message type, the length of the array, and the integers in the array to their ASCII versions and writing these strings to standard output. The standard input of the command processor will be connected to the standard output of the server (and vice versa). The command processor will read this sequence of string on its standard input. It will then transfer the call by converting the integers to internal form and using the usual socket version of `SendIntArrayMessage` to send the array to the GUI.

This does involve changing the server but only slightly. All of the server code uses the comm library to communicate. None of this code needs to be rewritten. The only change is that a special version of the comm library must be written and linked in with the server. No knowledge of how the server works is required to do this. All that is needed is knowledge of the way in which compute nodes can communicate with service nodes.

A Middle Tier with More Semantics

Any form of communication between compute and service nodes can be accommodated using this architecture. But now that we have a middle tier, we have the possibility of transferring some of the functions to the command processor which runs in the middle tier.

We noted that the server has the responsibility of sending the GUI the contents of directories on the server's file system. This is necessary because the user must use the GUI to select a file on the server's file system. The GUI uses this function to implement a remote file browser.

But file browsing is not a visualization function. It has nothing to do with volume visualization, for example. In addition, the GUI will need this function with all visualization servers and it does not make sense to implement it in every one of them. The logical thing to do is to place this function in the command processor. This requires the command processor to examine each message to see if it is the "Get directory contents" message. If it is, the command processor will handle the message itself and not pass it on to the server at all. The command processor gets the directory contents, encodes it as requires and sends it back using `SendStringMessage`.

If every visualization server uses the same message type for the "Get directory contents" message then the command processor can perform this function for all the visualization servers. In addition, the file browsing can be done before the server is even started. This saves the scarce resource of execution time on the MP machine.

There is one more function that the command processor could implement for the visualization servers. All of the visualization servers use the same model file format (called Exodus). An Exodus file contains meta-data that includes the number and names of all the variables as well as several other pieces of information.

A visualization GUI will put up lists of variable names as well as other Exodus file meta-data. Normally the server reads the Exodus file and so it interprets and sends the meta-data to the GUI. But the command processor can also open the Exodus file, read the meta-data, and send it to the GUI for display. This does mean that the meta-data will be read twice since the server will have to read it also. But the meta-data is at the beginning of the file and is fairly small. The rest of the file contains the data values. This part can be very large but it will only be read by the server.

The meta-data function is also required by all visualization servers and can be done even before the server is started. So we have two common functions that are taken on by the command processor. Assuming the ability to handle these functions is left in the servers (they do it now) then we will have the option of using the two-tier or the three-tier architecture. We have the option of leaving out the command processor when it is not necessary. This is true when we are working on the GUI and making sure that the GUI and the server are communicating correctly and that the GUI is displaying information as we would like to see it.

We consider this to be an intermediate step in the architecture. Eventually we will remove these functions from all of the visualization servers and always use a command processor. For debugging, we will host the command processor and the server on a desktop machine and their communication will use local sockets. The advantage of the architecture we have formulated is that we have the possibility of evolving in this manner. At all intermediate stages we have working systems that allow us to continue the development of out visualization servers. This flexibility comes from using the same interface in all tiers.

Changing the Middleware

We have described the system using sockets for communication (except for the case of using standard input and standard output for one MP machine). We used this as an example; the architecture does not depend on the type of middleware used. Again we have used an evolutionary approach to the problem.

In one experiment we wanted to use CORBA to communicate between the GUI tier and the command processor. The first step was to write a new version of the comm library that used CORBA instead of sockets. We implemented all of the comm functions as CORBA IDL operations. For example, SendIntMessage was made into a CORBA call. The command processor was modified to use CORBA functions also.

Note that the communication is in both directions and so the GUI and the command processor both are CORBA servers and CORBA clients. This is not desirable and one way to get around it is to switch to our second interface style, which we have called the

five-function interface because it comprises five functions (`SetAttribute`, `GetAttribute`, `TakeAction`, `GetImageInfo`, and `GetTriMeshInfo`).

This interface has several advantages. The immediate one is that it is a functional interface, that is, each call is a function. If we convert it to CORBA then the GUI is a CORBA client and the command processor is a CORBA server. The command processor (and hence the server) never sends anything to the GUI without being asked. It only returns values to the "`Get*`" functions. This interface is also smaller and hence easier to implement. Finally this interface allows us to add new attributes and actions without changing the command processor. As long as the GUI and the server understand the attributes and actions, the command processor can pass them through without knowing their meaning. Conversion to Java RMI, COM, or other middleware would be equally easy.

7. ADDING MORE TIERS

The command processor is useful to customize the communication with the compute nodes, to browse the server's file system, and to access data file meta-data. There are other functions that could go between the GUI and the server. Our architecture manages these with additional tiers. We will look at some of the additional tiers we have envisioned. None of these are implemented yet. We will use the volume visualization (VolVis) tool as an example.

We have discussed the possibility of a middle tier to handle the launching of visualization servers. Figure 4 shows how that would fit in. Figure 5 shows the addition of a tier which performs resource management and chooses the most appropriate machine to host the computation. Figure 6 shows a situation where there are multiple visualization users accessing multiple visualization servers. The architecture is repeated three times in this figure although there is sharing of some services.

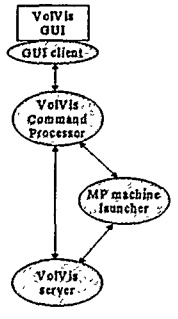

Figure 4: A launching tier

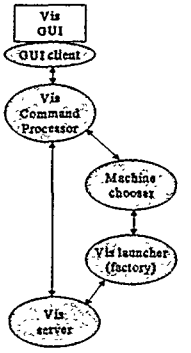

Figure 5: Launching and choosing tiers

We might also add tiers above the command processor. Figure 7 shows a service that combines views from two command processors to allow the integrated use of two visualization tools. Figure 8 shows the addition of a tier to handle session management that allows a user to start a session and then return to it later. Finally figure 9 shows all these new tiers in a single diagram.

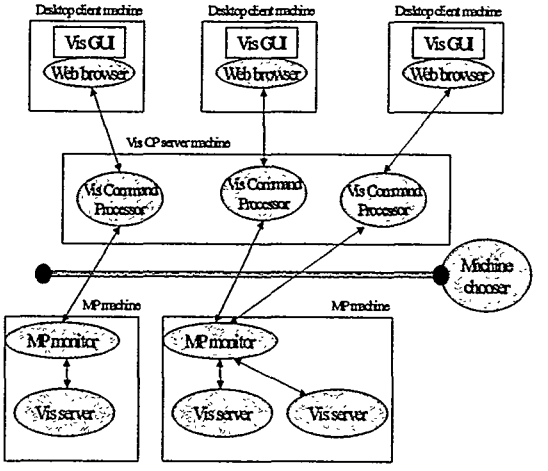

Figure 6: Multiple versions of the architecture

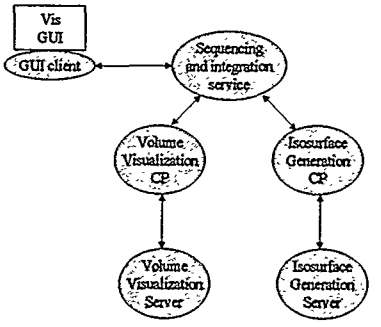

Figure 7: Combining two servers

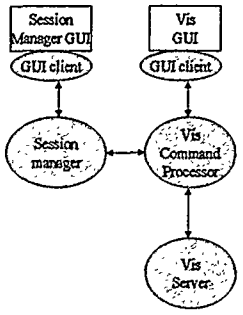

Figure 8: A session manager

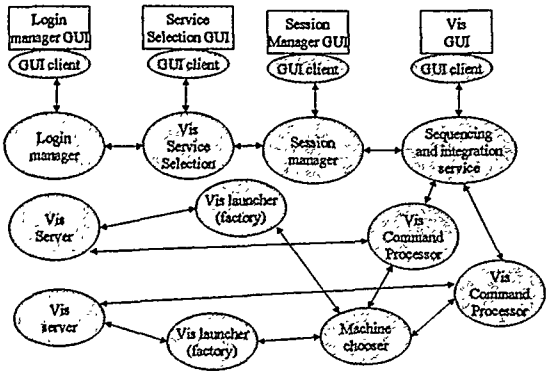

Figure 9: All the services together

8. CONCLUSIONS

Our architecture is based on the standard N-tier model. We choose this model because it is logical and flexible and has been shown to be an effective architecture for web applications. We have been investigating how to migrate our existing visualization servers to this architecture. We have also been investigating how a common interface at all tiers improves the flexibility of the architecture.

We started with a simple separation of the GUI from the visualization engine and then added new levels as needed to meet the goals we have and deal with problems in running the code on high-performance machines. We will continue to evolve the architecture as we gain experience with it. The main future task before us is to integrate this architecture with current developments at Sandia to handle the problem of very large data files. This will add a new tier to the system below the visualization server.

REFERENCES

1. V. P. Holmes, J. M. Linebarger, D. J. Miller, and R. L. Vandewart. "The Simulation Intranet Architecture." *Proceedings of the 1999 International Conference on Web-Based Modeling and Simulation*, SCS Simulation Series, vol. 31, no. 3, pp. 95–104.
2. V. P. Holmes, J. M. Linebarger, D. J. Miller, and R. L. Vandewart. "An Object-Based Metasystem for Distributed High Performance Simulation and Product Realization." *Proceedings of the 1999 Workshop on Parallel/High-Performance Object-Oriented Scientific Computing (POOSC'99)*, held at the 1999 European Conference on Object-Oriented Programming (ECOOP'99), Lisbon, Portugal, 14-18 June 1999, pp. 31–40.
3. Jason Wood, Ken Brodlie and Helen Wright. "Visualization over the World Wide Web and its application to environmental data." *Proceedings of the conference on Visualization '96*, 1996, pp. 81–86.
4. Cherilyn Michaels and Michael Bailey. "VizWiz: a Java applet for interactive 3D scientific visualization on the web." *Proceedings of the conference on Visualization '97*, 1997, pp. 261–267.
5. Suzana Djurcilov and Alex Pang. "Visualization products on-demand through the Web." *Proceedings of the third symposium on Virtual reality modeling language*, 1998, pp. 7–13.
6. Bastiaan Schönhage and Anton Eliëns. "A flexible architecture for user-adaptable visualization." *Proceedings of the workshop on New paradigms in information visualization and manipulation*, 1998, pp. 8–10.
7. Chad F. Salisbury, Steven D. Farr and Jason A. Moore. "Web-based simulation visualization using Java3D" *Proceedings of the winter simulation conference on Winter simulation (Vol. 2): simulation—a bridge to the future*, 1999, pp. 1425–1429.
8. Adel S. Elmaghraby, Sherif A. Elfayoumy, Irfan S. Karachiwala, James H. Graham, Ahmed Z. Emam and AlaaEldin Sleem. "Web-based performance visualization of distributed discrete event simulation." *Proceedings of the winter simulation conference on Winter simulation (Vol. 2): simulation—a bridge to the future*, 1999, pp. 1618–1623.
9. Klaus Engel, Rüdiger Westermann and Thomas Ertl. "Isosurface extraction techniques for Web-based volume visualization." *Proceedings of the conference on Visualization '99: Celebrating ten years*, 1999, pp. 139–146.