An Environment for Web-based Interaction and Steering of High-Performance Scientific Applications

Samian Kaur, Rajeev Muralidhar and Manish Parashar
Department of Electrical and Computer Engineering and CAIP Center, Rutgers University, 94 Brett Road, Piscataway, NJ 08854. Tel: (732) 445-5388 Fax: (732) 445-0593 Email: {samian, rajeevdm, parashar}@caip.rutgers.edu

Abstract

This paper presents an environment for Web-based interrogation, interaction and steering of high-performance parallel/distributed scientific applications. The architecture is composed of detachable thin-clients at the front-end, a network of Java interaction servers in the middle, and [a control network of sensors, actuators, interaction agents, and an application interaction proxy, superimposed on the application data-network, at the back-end. A key innovation of the architecture is the definition of interaction enabled Java proxies, (using the Java Native Interface), for distributed application computational objects. The interaction objects along with the application interaction proxy provide window into application execution, which can be accessed through the interaction web-server. The presented environment is part of an ongoing effort to develop and deploy a web-based computation collaboratory that enables geographically distributed scientists and engineers to collaboratively monitor, and control distributed applications. Its overall aim is to bring large distributed simulations to the scientist/engineers desktop by providing collaborative web-based portals for interaction and control.

I. INTRODUCTION

This paper presents an environment for web-based interrogation, interaction and steering of high-performance parallel/distributed scientific applications. This is a part of an ongoing effort to develop and deploy a web-based computation collaboratory that enables geographically distributed scientists and engineers to collaboratively monitor, and control distributed applications. Its overall aim is to bring large distributed simulations to the scientist/engineers desktop by providing collaborative web-based portals for interaction and control.

Simulations are playing an increasingly critical role in all areas of science and engineering. As the complexity and computational costs of these simulations grows, it has become increasingly important for the scientists/engineers to be able to monitor the progress of these simulations, and to control or steer them at runtime. The utility and cost-effectiveness of these simulations can be greatly increased by transforming the traditional batch simulations into more interactive ones. Closing the loop between the user and the simulations enables the experts to drive the discovery process by observing intermediate results, by changing parameters to lead the simulation to more interesting domains, play what-if games, detect and correct unstable situations, and terminate uninteresting runs early. For example, the scientist can modify a boundary condition based on the current state of the simulation. Similarly in a simulation based on adaptive meshes, additional resolution can be added on the fly in regions that need it. Using checkpoint/restart capabilities, the experts can now stop, rewind, and replay simulation time step(s) with different parameters when it does not correspond to expected values. Furthermore, the increased complexity and multi-disciplinary nature of these simulations necessitates a collaborative effort among multiple, usually geographically distributed, scientists/engineers. As a result, collaboration-enabling tools have become critical for transforming simulations into true research modalities.

Enabling seamless interaction and steering high-performance parallel/distributed applications requires a number of issues to be addressed. A primary issue is the definition and deployment of sensors and actuators that will be used to monitor and control the applications. These sensors and actuators must be co-located with the computational objects and must encapsulate the modes of interaction of the object. Defining these interfaces in a generic manner and deploying them in distributed environments can be non-trivial, as computational objects can span multiple processors and address spaces. The problem is further compounded in the case of adaptive applications (e.g. simulations on adaptive meshes) where computational objects can

be created, deleted, modified and redistributed on the fly. Another issue is the deployment of a control network that interconnects these sensor and actuators so that commands and requests can be routed to the appropriate set of computational objects, and information returned can be collated and coherently presented. Finally, the interaction and steering interfaces presented by the application needs to exported so that it can be simply accessed by a group of collaborating users to monitor, analyze, and control the application.

The objective of this paper is to present the design of a web-based collaborative interaction and steering environment that addresses each of these issues. The system supports a 3-tier architecture composed of detachable thin-clients at the front-end, a network of Java interaction servers in the middle, and a control network of sensors, actuators, interaction agents superimposed on the application data-network at the back-end. The interaction web server enables clients to connect to and collaboratively interact with registered application using a conventional browser. Furthermore, it provides seamless access to computational and visualization servers, and to simulation archives. The application control network enables sensor and actuators to be encapsulated within, and directly deployed with the computational objects. Interaction agents resident at each computational node register the interaction objects and export their interaction interfaces. These agents coordinate interactions with distributed and dynamic computational objects. The application interaction proxy manages the overall interaction through the control network of interaction agents and objects. It uses JNI [9] to create Java proxy objects that mirror the computational objects and allow them to be directly accessed by the interaction web-server. The presented research is part of DISCOVER (*D*istributed *I*nteractive *S*teering and *Co*llaborative *V*isualization *Envi*ronment), an ongoing research initiative aimed at developing and a web-based interactive computational collaboratory. The current implementation enables geographically distributed clients to use the web to simultaneously connect to, monitor and steer multiple applications in a collaborative fashion through a set of dedicated interaction Web Servers. More information about the DISCOVER project can be found at *http://www.caip.rutgers.edu/~parashar/TASSL*.

The rest of the paper is organized as follows: Section II outlines the DISCOVER system architecture. Section III presents the design, implementation, and operations of the interaction web-server. Section IV describes the design and implementation of the application interaction substrate and its interface to the interaction server. A brief overview of related research is presented in Section V. Section VI presents conclusions and current and future work.

II. DISCOVER: AN INTERACTIVE COMPUTATIONAL COLLABORATORY

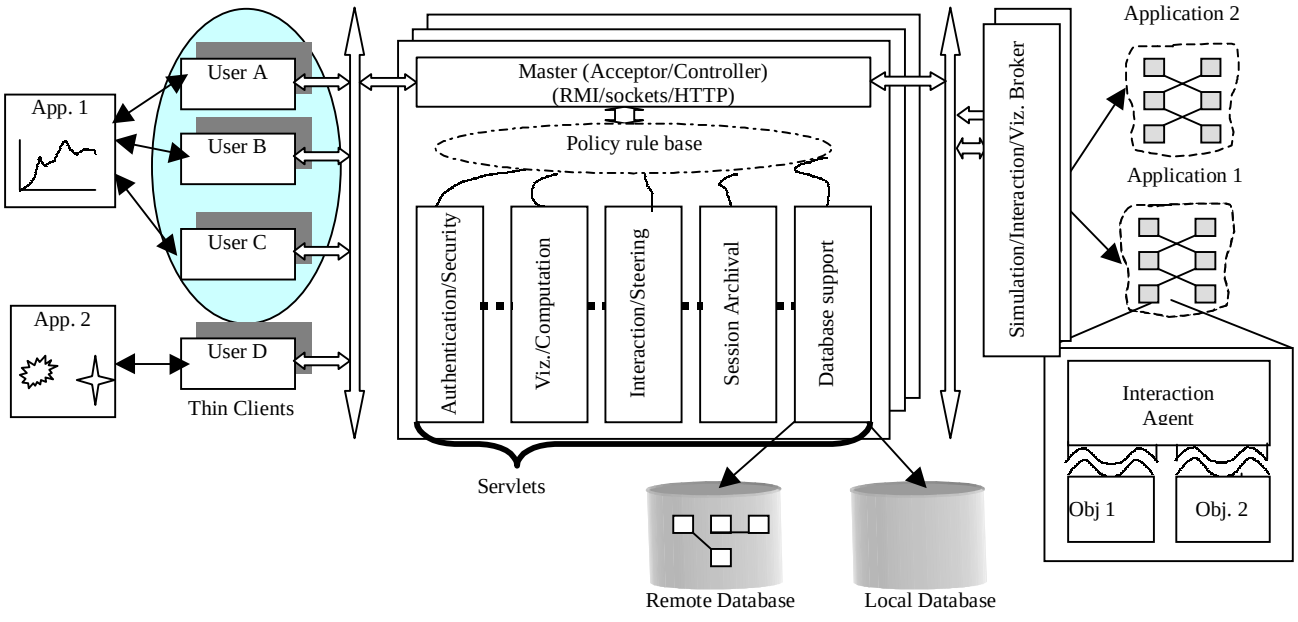

Figure 1 - Architectural Schematic of the DISCOVER Interactive Computational Collaboratory

Figure 1 presents an architectural overview of the DISCOVER collaboratory aimed at enabling web-based interaction and steering of high-performance parallel/distributed applications. The system conforms to a 3-tier architecture composed of detachable thin-clients at the front-end, a network of Java interaction servers in the middle, and a control network of sensors, actuators, interaction agents superimposed on the application data-network at the back-end. This paper presents the design and implementation of the latter two. The front-end consists of a range of detachable clients, from palmtops connected through a wireless link to high-end desktops with high-speed links. Clients can connect to a server at any time using a browser to receive information about active applications. Furthermore, they can form or join collaboration groups and can (collaboratively) interact with one or more applications based on their capabilities. Connections to the server could be a dedicated socket connection, via a pre-specified multicast group, or through the default HTTP connection. Information displayed at the client (e.g. surface/line plots, data, text) can be set to be updated on demand or automatically as the applications evolve. The client interface supports two desktops – a local desktop and a virtual desktop. The local desktop represents the users private view, while the virtual desktop is a metaphor for the global virtual space through which multiple users can collaborate with the aid of tools like whiteboards and chats. Application views (e.g. a plot) can be made collaborative by creating them in the virtual desktop or transferring them from the local to the virtual desktop. Session management and concurrency control is based on capabilities granted by the server. A simple locking mechanism is used to ensure that the application remains in a consistent state.

III. THE DISCOVER INTERACTION AND COLLABORATION SERVERS

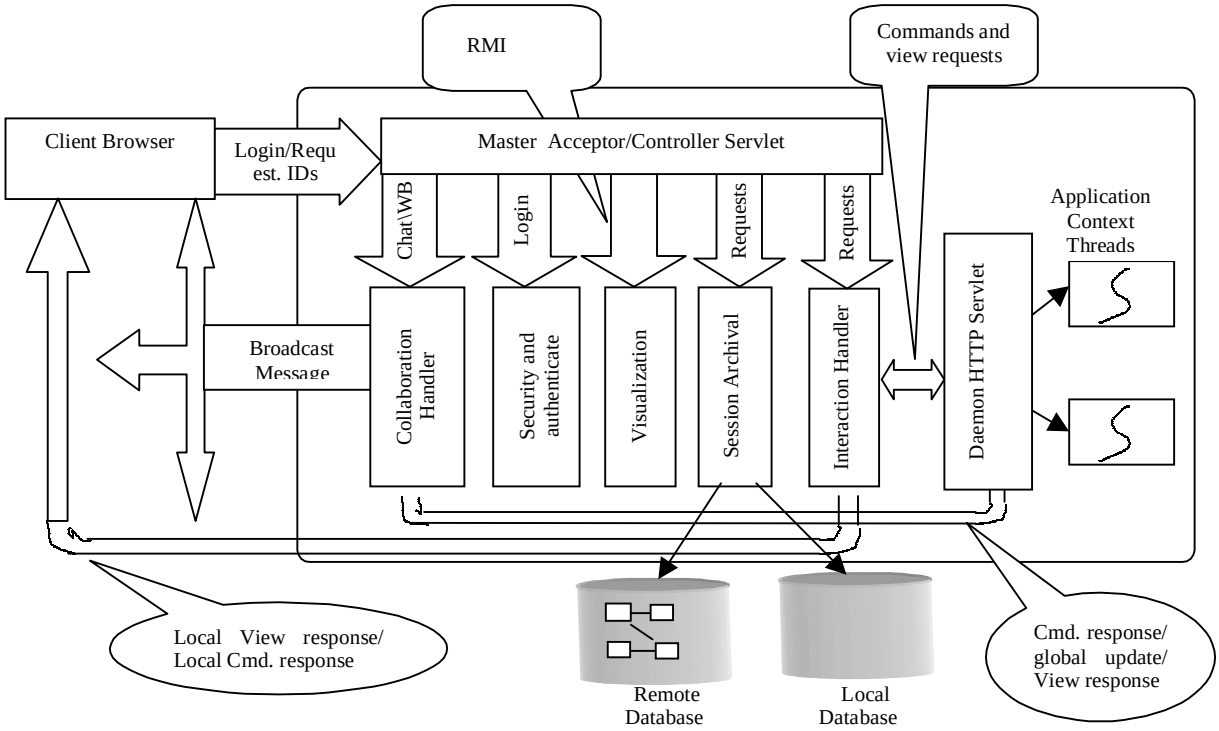

Figure 2 - Interaction Server Architecture

The middle tier of the DISCOVER system consists of a network of interaction and collaboration servers aimed at providing a web-based portal to executing high-performance applications. The servers build on *Servlet* [1] technology to add a range of specialized services to traditional web-servers. The primary innovation of the server architecture is the use of *reflection* to provide an extensible set of services that can be dynamically invoked in an application specific manner.

The overall architecture consists of a master acceptor/controller servlet and a suite of service handler servlets including interaction and steering handler, collaboration handler, security/authentication handler, visualization handler, session archival

handler and database handler. The overall architecture of the interaction server is shown in Figure 2. Key components are described below.

A. Master (Acceptor/Controller)

The master (acceptor/controller) is the central hub for all communication between the client and the server. Furthermore, it coordinates all registered applications and interaction sessions. Its implementation is based on traditional object resource brokers (ORB) with extensions to handle requests for distributed computations and interactive analysis and control. The application, on connection, registers with a daemon HTTP servlet at the master controller, which spawns a dedicated *application interaction broker* for each application. All further client requests are suitably processed and forwarded to the corresponding broker. The broker, through an application proxy, deals with interaction agents deployed the computational nodes to locate application objects, discover object interaction/analysis interfaces, forward queries and commands, and aggregate information from distributed objects. On client connection, the master controller also provides the incoming client with a list of registered applications. It accepts, parses and validates client requests and redirects them to the relevant utility servlet. Finally, the master servlet also maintains a *policy rule base*, which used to manage access control as well as to manage client heterogeneity and QoS control.

B. Interaction/Collaboration Handlers

The interaction handler servlet accepts all interaction requests and redirects them to the relevant application brokers. Application responses are forwarded to either the application's collaboration group or the requesting client. In the former case, the response is forwarded to the Collaboration Handler that further broadcasts/multicasts it. The interaction handler maintains references to application brokers in a look-up table indexed by an application ID. This enables the handler to direct requests to the appropriate application broker.

The DISCOVER framework enables clients connected to same application to form collaboration groups. An initially empty *collaborative interaction group* is dynamically created for every application that registers with the server. This group is updated each time a client connects (or disconnects) to (from) the application. The group is then used to replicate events generated by the collaboration tools, i.e. the white board or chat tools, as well as global updates (e.g. timestep or iteration count) generated by the application.

C. Security/Authentication Handler

A special authentication and security handler manages user authentication. For each application, there exists a list of clients and their access control lists (ACL) to determine the permissions assigned to them. The authentication servlet is invoked to validate a client request against its ACL and thus determine whether the request is to be forwarded to the application or not. This feature is of paramount importance as unauthorized tampering of a mission critical simulation is unacceptable. Any application to be considered for this framework should specify the clients and their privileges before any interaction can take place. Our implementation supports a two-level authentication at startup, the first level is to authorize access to the server and the second level is to permit access to a particular application. On successful validation of the primary authorization, the user is shown a list of the applications s/he can register with. A second level authentication is performed for the application s/he chooses at this stage. Further, the use of *digital certificates* validates the server identity before the client downloads the applet and makes a connection. We are currently exploring the *Secure Socket Layer* to provide encryption for all communication between the client and the server.

D. Session Archival Handler

The session archival server logs all interactions between the client(s) and an application to support late-arriving clients and provide replay capabilities. It also supports checkpoint and restart functionality. A session is bound by the time the client connects to the application to the time the client disconnects or switches to another application. The entire session is logged

into a persistent storage to enable rollback and retrospective analysis of the application progress. Furthermore, all the transactions undertaken in a session are log with their timestamps; a transaction being bound by the time a view/command request is issued to the time the response is received.

E. Application View Plug-Ins

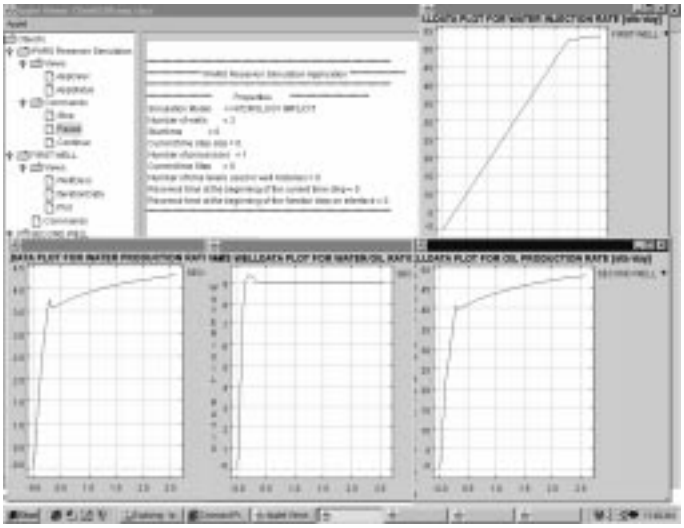

Figure 3 - Views for the IPARS Oil Reservoir Simulation

Application information is presented to the client in the form of application *View*. Typical views include text strings, plots, contours, and isosurfaces. Associated with each of these views is a view plug-in that can be downloaded form the server at runtime and used to present a requested view to the user. The server supports and extendible plug-in repository and allows users to extend, customize or create new views and associated plug-ins. For example, in the current implementation plotting views are based on the Ptolemy [9] software package. These plots are of two kinds: iterative or one time. The iterative plots show the incremental change in the parameter with successive iterations whereas the latter is a response to a user request to show a log of the parameter history from startup or checkpoint. The screen dump in Figure 3 shows iterative plots of well parameters being monitored for the IPARS oil well simulation.

F. Interaction Server Implementation Overview

In the current implementation, the server is a Java web server with threads listening to application connections. The implementation relies on the Java servlets to provide the requirements and services outlined earlier. Written to Sun Microsystems' API, Java servlets are embedded in the Web server; servlets access the library of HTTP-specific calls, and make use of all the benefits of the Java language, including portability, reusability, and crash protection. The server consists of three main servlets - the *Master Servlet*, the *Interaction Servlet* and the *Collaboration Handler*. In addition, there could be any number of utility servlets like the login servlet, the authentication servlet, or visualization plug-in servlet.

Each servlet interacts using two objects, ServletRequest and ServletResponse. A new request invokes the Master servlet's `doPost()` method, where the type of the request is determined and the request redirected to the appropriate handler. By invoking the services dynamically, a component-based 'plug and play' design is achieved. Thus, any changes to the underlying classes do not affect the master servlet, provided they adhere to the servlet API and are placed in the appropriate directory.

To implement collaboration, we need to circumvent the *unidirectional* request/response communication paradigm inherent in servlet communication. Our client simulates bi-directional communication by making a series of requests which block until the server decides it's time to return something, who piggybacks it's output on the HTTP response that goes back to the client. Global events can broadcast to all clients register their interest in a particular event or set of events.

```
Class newRequest;
HttpSession session = req.getSession(true);
try
{
//Determine which class to  instantiate
handlerName = Class.forName (req.
getParameter ("handler"));
.
.
//Instantiate the class
handler =handlerName.newInstance();
.
.
//Redirect the request to the new Instance
handler.doPost(req);
}
```

Dynamic Invocation using reflection

IV. A CONTROL NETWORK FOR WEB-BASED APPLICATION INTERACTION AND STEERING

The DISCOVER control network is composed of three components: (1) distributed interaction objects that are derived from simulation computational objects and encapsulate sensors and actuators, (2) a network of interaction agents that monitor and control interaction objects, and (3) an application interaction proxy that provides a web-enabled window to the application and its sensors and actuators. These components are described below.

A. Sensors/Actuators and Interaction Objects

An essential requirement for application interaction and steering is the definition and deployment of appropriate sensors and actuators so as to be co-located with application computational objects. In the DISCOVER control network, sensors and actuators are encapsulated within, and deployed with, the application objects themselves. This is achieved by extending the computational objects with interaction capabilities by deriving them from a virtual interaction base object. The derived interaction objects define a set of *View*s that they can provide and a set of *Commands* that they can accept. These views and commands are exported by the interaction agents using an interaction IDL. Interaction objects can be either local to a single computational node, distributed across multiple nodes, or shared between some or all of the nodes. Distributed objects have an additional *distribution* attribute that describes their layouts. All interaction objects can be created or deleted during application execution and can migrate between computational nodes. A distributed interaction object can modify its distribution at any time. The interaction agents coordinate between the nodes associated with a distributed object and update the distribution attribute when it changes.

In the case of applications written in non-object-oriented languages such as Fortran, application data structures are first converted into computation objects using a C++ wrapper object. These objects are then transformed to interaction objects as described above. IPARS, a Fortran-based oil reservoir simulator, developed at the Center for Subsurface Modeling, University of Texas at Austin, was made interactive using such an approach (see www.caip.rutgers.edu/~parashar/TASSL/DISCOVER/ipars.html).

B. Interaction Agents

Every computation node houses an *interaction agent* that maintains an *object registry* of all interaction objects currently active at that node. Interaction objects register with the interaction agent. The agent maintains handles to these objects in the registry and manages interaction and steering requests to the objects. In the case of distributed interaction objects, the agent coordinates with other agents to generate views or handle commands. Interaction View/Command interfaces are exported by the agents for dynamic discovery using a simple interaction interface description language. The networks of interaction agents are connected to the application interaction proxy described below.

C. Applications Interaction Proxy

The application interaction proxy is representative of the entire application and provides a web-enabled portal to the application. It is responsible for interfacing with the interaction server and for delegating interaction requests to the appropriate interaction agents, and for combining and collating their responses. The application interaction proxy uses the proxy design pattern [24] to create a Java mirror of each application interaction object. It does this using Java Native Interface (JNI) [2]. The proxy pattern provides a placeholder for another object, known as the subject, to control access to it. The Java object can be thought of as a remote proxy, providing a local representative for the application object in the remote (possibly distributed) address space. The primary advantage of this approach is approach is that the interaction objects can now be easily exported to the interaction servers using the Serializable interface of Java without the need for explicit serialization and de-serialization. This way of mirroring interaction objects is similar to the approach of the Mirror Object Steering System [8]. The application interaction proxy hosts a *central object registry* of all interaction objects and their Java mirrors, and a Java Virtual Machine (JVM) to enable it to directly connect to and communicate with the interaction web server. The sequence of interactions between the interaction agents, the application interaction proxy, and the interaction server are illustrated in Figure 4.

Interaction objects are first created and registered with the interaction agent on every computation node. At the application interaction proxy, the JVM is initialized and the central object registry is initialized. The interaction agents now export their object registries of interaction objects to the application interaction proxy, where their corresponding Java mirrors are created. Once the initial object registration process is complete, the application interaction proxy begins the interaction session by registering the application and its interaction objects with the interaction web server and the application now begins its computations. The application now proceeds through the computation and interaction phases alternately. After a sequence of computations, typically after a pre-specified number of iterations, when the application data structures reach stable values, the application enters the interaction phase where it will look for any outstanding interaction requests. By making sure that the interaction phase starts only when the application data have attained stable values, the need for application level synchronization of data structures is circumvented. The application begins an interaction phase by exporting all global computation values like the interaction number, time step, etc. to the interaction Web Server – all clients are automatically updated with these global computation parameters immediately. Now the application interaction proxy looks for any interaction request to view or modify an interaction object. The object is looked up in the registry and the interaction request and parameters, if any are passed to it for processing. The interaction agent collates the response sends it to the application interaction proxy, which then publishes the response to the interaction server and collaborating clients.

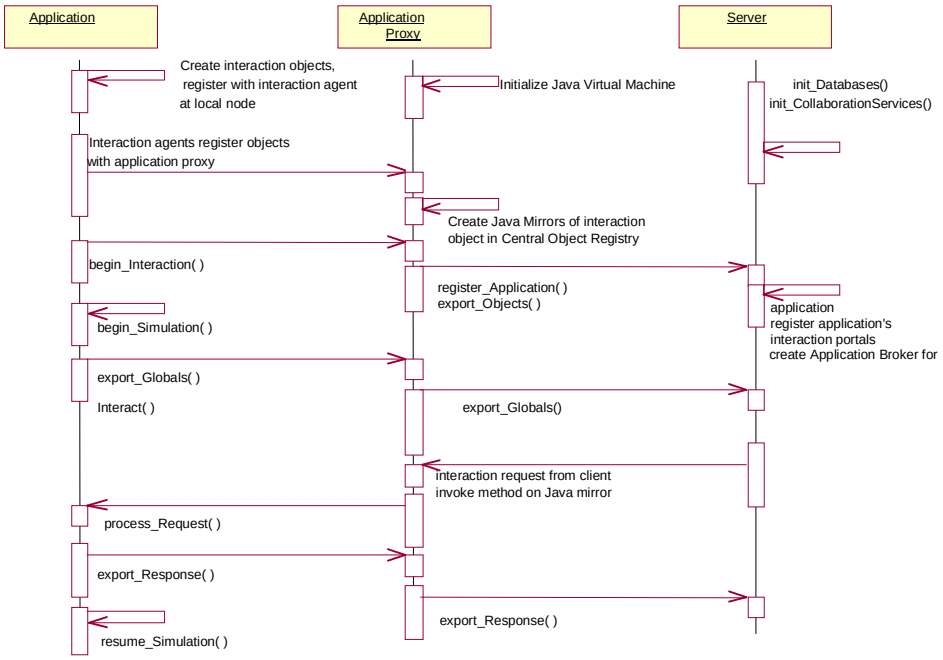

Figure 4 - Application-server interactions

D. Implementation Overview

The `Object` class is the abstract base class of the distributed object library having pure virtual functions that need to be overridden by computation objects in order to enable interaction. The virtual functions allow the derived objects to export their interaction interfaces; by this we mean the interfaces that allow computation data to be exported and those that allow modification of steering parameters. For example, the virtual function `addGetInterface( )` is to be used to exporting an interface that could be used to query the object's data, and the function `addSetInterface( )` is for exporting a function to modify data in the computation object. The virtual function `processMessage( )` has to be overridden by the computation

object to respond to interaction messages; the action to be taken for a query or command message has to be defined by each computation object. Two special functions `scatter( )` and `gather()` need to be overridden for distributed objects to specify the data distribution among the processing nodes. The `gather()` function is used to accumulate data from a distributed object in an application-specific manner in response to a view request from an interacting client.

When an interaction object is received at the application interaction proxy from the interaction agents, its constructor calls from native code into the JVM and creates a Java mirror object. This object has analogous member variables and the same method names with analogous signatures. Both the C++ object and the Java object are entered into the same object registry – the registry has two entries for each Java object/C++ object pair. Each entry is indexed using the C++ object address or the Java object hash code value which is obtained using the `java.lang.Object.getHashCode()` method. Once the registry entries are created for all interaction objects, a Java application is started on the JVM at the application proxy.

V. RELATED WORK

A number of systems have been built for interactive steering and control of high performance applications. Only the systems that similar in terms of architecture and implementation are mentioned in this section. In the *Gateway*[6] system, multiple clients can connect through a CORBA-based middleware to a back end consisting of high-end computing resources managed using the *Globus* [7] metacomputing toolkit. In the *Mirror Object Steering System*[8], a Mirror Object Model is used to translate data structures into CORBA objects that can then be monitored and steered. In *CSE*[9], multiple existing applications can be integrated into the framework by annotation of the application source code. The applications and multiple user interfaces processes can be distributed over different platforms and full customization of the user interface is provided for through graphical editing. In *Falcon*[10] also, existing applications can be integrated by manual annotation of application source code and provide the end user with an abstraction that reveals only the important steering parameters and output data. Also, there is no collaboration support among multiple clients. *SCIRun*[11] uses visual programming to create a steering application and is suited for developing new applications for interactive steering. It also allows for some monitoring of a running application and some limited modification of the data flow network that constitutes the running application. It also provided a visualization interface for observing application data continuously. *Autopilot*[13] presented additional support for automated steering in systems where decisions are generated by fuzzy logic and neural networks. *Cumulvs*[14] provides computational steering in addition to fault-tolerance and visualization by exploiting knowledge of the PVM virtual machine and applications built on PVM. In *VIPER*[15], like in the CSE, the application is regarded as a kind of client process, following the client/server/client architecture, and no support is reported for the simultaneous steering of multiple applications or by multiple users. The *Topos*[16] system is designed to be Web enabled and uses CORBA objects to implement the necessary middle-ware to solve the problems of the interoperability and the software portability. In addition to these systems, there are numerous systems that provide web-based collaborative visualization environments like *DOVE*[17], the *Web Based Collaborative Visualization* system[18], the *NCSA Habenaro* system[19], *Tango*[20], *CCASE*[21] and *CEV*[22].

VI. CONCLUSION AND FUTURE WORK

Simulations are playing an increasingly critical role in all areas of science and engineering. As the complexity and computational costs of these simulations grows, it has become increasingly important for the scientists/engineers to be able to monitor the progress of these simulations, and to control or steer them at runtime. The utility and cost-effectiveness of these simulations can be greatly increased by transforming the traditional batch simulations into more interactive ones. Furthermore, the complexities of the problems being addressed by high performance applications very often require a collaborative effort among multiple scientists and engineers. This paper presented the design of DISCOVER, an environment for web-based interrogation, interaction and steering of high-performance parallel/distributed scientific applications. The overall goal of the system is the development and deployment of a web-based computation collaboratory that enables geographically distributed scientists and engineers to collaboratively monitor, and control distributed applications.

Our current and future work is following two directions. First, the monitoring and steering servers are to be extended to multiple nodes, to enhance scalability and reduce the bottleneck of the centralized system. One of the issues to be addressed here is the mechanism to route interaction requests and responses correctly when the application and the interacting clients are connected to different servers. Further, the set of distributed servers have to maintain a shared state of interaction sessions in order to effectively handle load balancing and fault tolerance in the event of server and/or network failures. Secondly, at the application end, the simple control network in this work needs to be extended to better address the issues of scalability and performance of the interaction system.

VII. REFERENCES

[1]. Java Servlet API, *http://java.sun.com/products/servlet*.

[2]. Java Native Interface Specification, *http://web2.java.sun.com/products/jdk/1.1/docs/guide/jni*.

[3]. Remote Method Invocation, *http://java.sun.com/products/jdk/rmi*.

[4]. Jospeh Buck, Soonhoi Ha, Edward A. Lee, David G. Messerschmitt, "*Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems*", International Journal of Computer Simulation, 1992.

[5]. John A. Wheeler et al., "*IPARS: Integrated Parallel Reservoir Simulator*", Center for Subsurface Modeling, University of Texas at Austin, *http://www.ticam.utexas.edu/CSM*.

[6]. Bill Asbury, Geoffrey Fox, Tom Haupt, Ken Flurchick "*The Gateway Project: An Interoperable Problem Solving Environments Framework For High Performance Computing*", *http://www.osc.edu/~kenf/theGateway*.

[7]. I. Foster, C. Kesselman, "*Globus: A Metacomputing Infrastructure Toolkit*," International Journal of Supercomputer Applications, 11(2): 115-128, 1997.

[8]. "*Mirror Object Steering System*", *http://www.cc.gatech.edu/systems/projects/MOSS*.

[9]. Robert van Liere, Jan Harkes, Wim de Leeuw. *A Distributed Blackboard Architecture for Interactive Data Visualization*. Proceedings of IEEE Visualization'98 Conference, D. Ebert, H. Rushmeier and H. Hagen (eds.), IEEE Computer Society Press, 1998.

[10]. "*The Falcon Monitoring and Steering System*", *http://www.cs.gatech.edu/projects/FALCON*.

[11]. S.G. Parker, C.R. Johnson. *SCIRun : a scientific Programming Environment for computational steering*. In Proceedings of Supercomputing '95, 1995.

[12]. D.J. Jablonowski, J.D. Bruner, B.Bliss, R.B Haber. *VASE : The Visualization and application steering environment*. In Proceedings of Supercomputing '93, pages 560-569, 1993.

[13]. D.A. Reed, C.L. Elford, S.E. Lamm, T.M. Madhyastha, E. Smirni, H.J. Siegel. *The Next Frontier: interactive and closed loop performance steering.* In Proc. 1996

[14]. G.A. Geist II, J.A. Kohl, P.M. Papadopoulos. *CUMULVS : Providing fault tolerance, visualization, and steering of parallel applications.* The International Journal of Supercomputer Applications and High Performance Computing, 11(3):224-235, 1997.

[15]. S. Rathmayer, M. Lenke. *A tool for on-line visualization and interactive steering of parallel hpc applications.* In Proceedings of the 11th International Parallel Processing Symposium, IPPS 97, pages 181-186, 1997.

[16]. Sergey S. Kosyakov, Serge A. Krashakov, Lev N. Shchur. *Topos: Manager for Distributed Computing Media*. Proceedings of Advances in Databases and Information Systems (ADBIS) '97.

[17]. Lalit Kumar Jain. "*A Distributed, Component-Based Solution for Scientific Information Management*". MS Report, Oregon State University, July 1998.

[18]. C. Bajaj, S. Cutchin. *Web based Collaborative Visualization of Distributed and Parallel Simulation***,** accepted for presentation in IEEE Parallel Symposium on Visualization '99.

[19]. Brent Driggers, Jay Alameda, Ken Bishop, "Distributed Collaboration for Engineering and Scientific Applications Implemented in Habanero, a Java-Based Environment", *http://union.ncsa.uiuc.edu/habenaro*.

[20]. Geoffrey Fox et al., "*Tango - A Collaborative Environment for the World Wide Web*," Technical Report, NPAC Syracuse University, Syracuse NY.

[21]. Rajeev R. Raje, Antony Teal, Joeseph Coulson ,Sihai Yao, William Winn, "*CCASE-A Collaborative Computer Assisted Software Engineering Environment*", Purdue University, Indianapolis.

[22]. Michael Boyles, Rajeev Raje, Shiaofen Fang, "CEV: Collaboration Environment for Visualization Using Java RMI," Purdue University, Indianapolis.

[23]. Common Object Request Broker Architecture, *http://www.omg.org*.

[24]. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, "*Design Patterns*", Addison Wesley Professional Computing Series, 1994.