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

Vijay Mann, Vincent Matossian, 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: {vijay,vincentm,rajeevdm,parashar}@caip.rutgers.edu

ABSTRACT

This paper presents the design, implementation, and deployment of the DISCOVER (*D*istributed *I*nteractive *S*teering and *Co*llaborative *V*isualization *E*nvi*R*onment) web-based computational collaboratory. The primary goal of this collaboratory is to bring large distributed simulations to the scientists'/engineers' desktop by providing collaborative web-based portals for interaction and control. DISCOVER provides a 3-tier architecture composed of detachable thin-clients at the front-end, a network of web servers in the middle, and a control network of sensors, actuators, interaction agents superimposed on the application at the back-end. The interaction servers build on *servlet* technology and enable clients to connect to, and collaboratively interact with registered applications using a conventional browser. The application control network enables sensors 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. The application interaction gateway manages the overall interaction through the control network of interaction agents and objects. It uses the Java Native Interface to create Java proxy objects that mirror the computational objects and allow them to be directly accessed by the interaction web-server. Security and authentication services are provided using customizable access control lists built on the SSL-based secure server.

I. INTRODUCTION

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 important for the scientists and 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. 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 are critical for transforming simulations into true research modalities.

Enabling seamless interaction and steering of high-performance parallel/distributed applications presents many challenges. A key issue is the definition, deployment and access of *interaction objects* with *sensors* and *actuators* [4] that will be used to monitor and control the applications. These sensors and actuators must be co-located with the computational data-structures in order to monitor and control them at run-time. Defining these (sensor/actuator) 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 need to be exported so that they can be easily accessed by a group of collaborating users to monitor, analyze, and control the application.

This paper presents the design, implementation, and deployment of the DISCOVER (*D*istributed *I*nteractive *S*teering and *Co*llaborative *V*isualization *E*nvi*R*onment) web-based computational

collaboratory. Its primary goal is to bring large distributed simulations to the scientists'/engineers' desktop by providing collaborative web-based portals for interaction and control. DISCOVER has a 3-tier architecture composed of detachable client portals at the front-end, a network of interaction servers in the middle, and a control network of sensors, actuators, and interaction agents superimposed on the application at the back-end. The interaction servers build on *servlet* technology and enable clients to connect to, and collaboratively interact with registered applications using a conventional browser. The application control network enables sensors 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 gateway manages the overall interaction through the control network of interaction agents and objects. It uses the Java Native Interface (JNI) [20] to create Java proxy objects that mirror the computational objects and allow them to be directly accessed by the interaction web-server. Security and authentication services are provided using customizable access control lists built on the SSL-based secure server.

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

II. RELATED WORK

Many interactive computational problem-solving environments are being proposed and developed to address different aspects of application composition, configuration and execution. Similarly, a number of groupware infrastructures that provide collaboration capabilities have separately evolved. Such PSEs have also been termed as distributed laboratories [6]. Existing interactive and collaborative PSE's are classified

and briefly described below. While these systems provide either capability, they do not combine them to provide a collaborative PSE for application interaction and control. A more detailed description and comparison of interactive steering systems can be found in [1], [2] and [3].

1. *Systems for interactive program construction* – Systems in this category, such as *SCIRun* [16], provide support for interactive program construction. SCIRun enables users to graphically connect application components as a data-flow graph. This system primary targeted towards composing and configuring new applications using existing components. It also provides some run-time monitoring capabilities.
2. *Systems for performance optimizations* – These systems aim at interactively optimizing the performance of applications. For example, the *Autopilot* [17] system provides a number of system-level performance sensors with a variety of sensor policies to monitor and tune application performance. These systems typically do not provide access to application-level objects for interaction and steering.
3. *Systems for application remote configuration and deployment* – Systems in this category use existing high performance metacomputing backend resources and provide powerful visual authoring toolkits to configure and deploy distributed applications on these resources. The CoG Kit [12] provides commodity access to the Globus [13] metacomputing environment. Similarly WebFlow [11] and Gateway [10] provide support for configuring, deploying and analyzing distributed applications on Globus. These systems, however, do not target application level run-time interaction and steering.
4. *Systems for run-time interactive steering and control* – Systems providing application-level run-time interactive steering and control capabilities are based on one of the following two approaches:
    a) *Event based steering systems* – In these systems, monitoring and steering actions are based on low-level system "events" that occur during the course of program execution. Application code is instrumented and interaction takes place when the pre-defined events occur. The Progress [15] system provides interaction capability using this approach. The Magellan [5] system extends the Progress approach by incorporating *language directed steering*

capabilities. It also supports simultaneous steering of multiple applications. Both systems require a server process executing in the same address space as the application, to enable interaction. The Computational Steering Environment (CSE) [14] also uses the event based steering approach, but uses a *blackboard* architecture (instead of the server process). A *data manager* acts as a blackboard for communicating data values between the application and the clients.

b) *Systems with high-level abstractions for steering and control* – The Mirror Object Steering System (MOSS) ([7], [8], [24]) provides a high-level model for steering applications. *Mirror objects* are analogues to the objects (data structures) in the application program and are used for monitoring and steering. Application data structure methods are made available to the interactivity system, which are used to perform the steering actions. MOSS is based on CORBA-style objects and has been implemented using a CORBA-compliant object-oriented language. High-level abstractions for interaction and steering provide the most general approach for enabling interaction in applications. The DISCOVER control network presented this paper extends this approach.

5. *Collaboration groupware* – Collaborative groupware environments include *DOVE* [27], *Web Based Collaborative Visualization* [25] system, *NCSA Habenaro* [26], *Tango* [28], *CCASE* [29] and *CEV* [30]. The Tango collaboration framework is web-based and uses centralized server architecture. The Habenaro framework is also Java-based and web-enabled, and uses a centralized server. It however only supports Java applications. The CCASEE provides a (shared) distributed workspace using Java RMI. The CEV system provides collaborative visualization. It uses a central server to perform the computations necessary to generate new collaborative views. The *DOVE* and the *Web Based Collaborative Visualization* systems also provide support for collaborative visualization.

The DISCOVER computational collaboratory presented in this paper provides an interactive and collaborative PSE for application-level runtime interaction and control using high-level abstractions. It

brings together key technologies in web portals, web servers, collaboration, application interaction and steering, and high performance computing. Key contributions of the DISCOVER system include:

1. Interaction mechanisms for distributed dynamic interactive objects that can span multiple address spaces and can be dynamically created, migrated and destroyed.
2. A scalable hierarchical control network to connect interaction objects, and sensors and actuators distributed over very large parallel systems.
3. Collaborative, web-based, interaction and steering portals for remote access to distributed application using standard distributed object interfaces such as Java RMI [21] and CORBA [19].

III. DISCOVER: AN INTERACTIVE COMPUTATIONAL COLLABORATORY

The DISCOVER computational collaboratory is a virtual, interactive and collaborative PSE that enables geographically distributed scientists and engineers to collaboratively monitor, and control (new and existing) high performance parallel/distributed applications using web-based portals. An architectural overview of the DISCOVER collaboratory is presented in Figure 1. DISCOVER is built using a 3-tier architecture. Its front-end consists of detachable client portals. 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. A network of interaction and collaboration servers forms the middle tier. These servers extend web-servers with interaction and collaboration capabilities. The back-end consists of a control network of sensors, actuators and interaction agents. The DISCOVER interaction model is application initiated, i.e. the application registers with the server exporting an interaction interface composed of "views" and "commands" for different application objects. Views encapsulate sensors and provide information about application and application objects, while commands encapsulate actuators and process steering requests. Some or all of these views/commands may be collaboratively accessed by groups of

client based on the client's capabilities. DISCOVER is currently operational[1] and being used to provide interaction capabilities to a number of scientific and engineering applications, including oil reservoir simulations, computational fluid dynamics and numerical relativity. The DISCOVER components are described in the following sections.

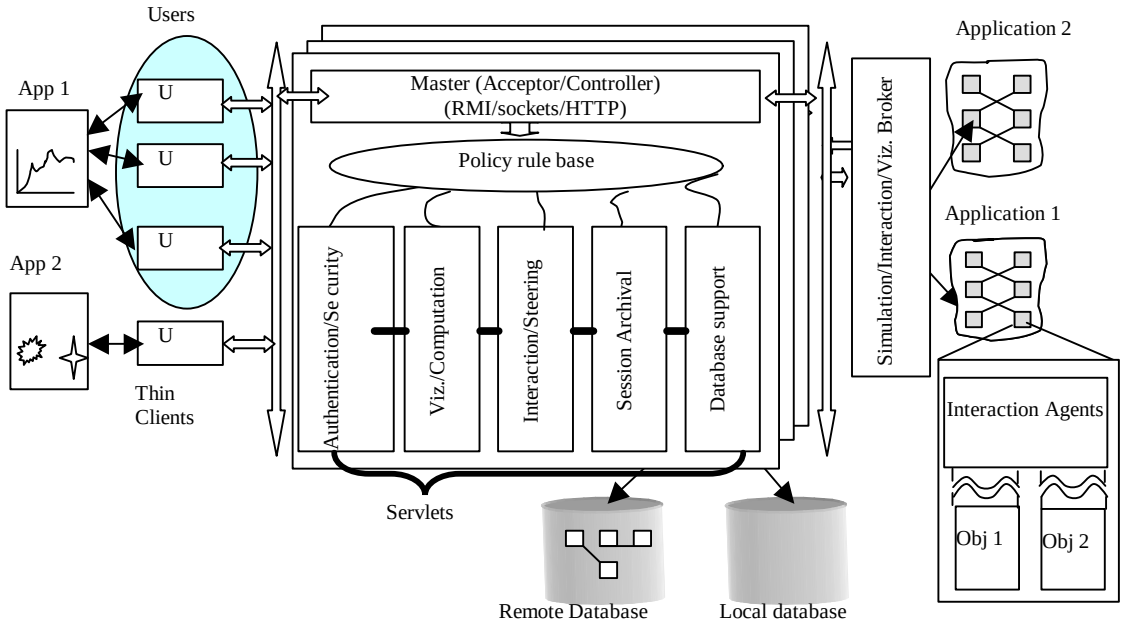

Figure 1 – Architectural Schematic of the DISCOVER Computational Collaboratory

IV. DISCOVER INTERACTION AND COLLABORATION SERVERS

The DISCOVER interaction/collaboration server builds on a traditional (secure) web server and extends its functionality to handle real-time information flow, and serve client requests and application connections. Extension is achieved using Java servlets [22] (server side Java programs). Each server consists of a number of handler servlets running to provide different interaction and collaboration services. Clients connect to the server using standard HTTP, while application-to-server communication is achieved either using distributed object protocols like CORBA [19] and Java RMI [21], or a more optimized, custom protocol using (secure) sockets. The core service handlers provided by each server include, the *Master Handler*, *Collaboration Handler*, *Command Handler*, *Security/Authentication*

[1] See www.discoverportal.org

Handler and a *Daemon Servlet* that listens for application connections. In addition to these core handlers, a number of auxiliary handlers providing services such as session archival, database handling, visualization, request redirection, and remote application proxy invocations (using CORBA). The different services are described in the following paragraphs:

A. Core Discover Services

1. Master Handler

The master (accepter/controller) handler servlet is the client's gateway to the server. It manages client service requests, such as authentication, interaction and steering, collaboration, session archival, and request redirection, and delegates them to the corresponding handler by invoking the corresponding handler servlet. The handler may be invoked on the local server if the service is locally available, or on a remote server using CORBA. Within a local server, it relies on *reflection* to dynamically invoke the handlers, thereby providing an extensible set of services. The master servlet creates a session object for each connecting client and uses it to maintain information about client-server-application sessions. It provides each client with a unique client-id. The client-id along with an application-id (corresponding to the application to which the client is connected) is used to identify each session. Finally, the master is responsible for generating the dynamic HTML required to present application information requested by a clients.

2. Security/Authentication Handler

Security, client authentication and application access control is managed by a dedicated security and authentication handler. The current implementation supports two-level client authentication at startup; the first level is to authorize access to the server and the second level to permit access to a particular application. On successful validation of the primary authorization, the user is presented a list of applications to which s/he has access capabilities. A second level authentication is performed for the application s/he chooses. Once authenticated, the authentication handler servlet builds a customized interaction interface for the client to match his/her access capabilities (i.e. monitor only, monitor and

steer, etc.). This ensures that a client can only access, interact with and steer an application in an authorized way. All communications are encrypted using the *Secure Socket Layer*. On the client side, we are exploring *digital certificates* to validate the server identity before the client downloads views. We are also exploring emerging technologies such as Myproxy [34] to provide secure access to the application.

To control access to the applications, all applications are required to be registered with the server and to provide list of users and their access privileges (e.g. read, modify). The application can also provide access privileges (typically read-only) to the "world". This information is used to create access control lists (ACL) for each user-application pair. At login time a different interaction interface is generated for each client based on his/her ACL. The DISCOVER server provides 3 different access privilege levels – Level 1 where the client is passive and only gets status messages and global updates from the application; Level 2 where the client is able to issue requests for applications views in addition to the Level 1 privileges; and Level 3 where the client is able to issue commands in addition to view requests and global updates.

3. Command / Interaction Handler

The command handler manages all client view and command requests. On receiving these requests from the master handler, the command handler looks up the appropriate application proxy, and forwards them to this proxy. The collaboration handler described below handles the responses to these requests. All requests and responses are Java objects and take advantage of Java's object serialization capability. 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 during collaborative interactions. This ensures that only one client "drives" (issues commands) the application at any time. Lock are typically requested and released explicitly by a user. Preemption occurs only when the driver fails to respond to the server for an extended period of time. Commands issued by the driver are broadcast to all clients logged on the application.

4. Collaboration Handler

DISCOVER enables multiple clients to collaboratively interact with and steer applications. On the server side the collaboration handler manages all collaboration, while on the client side a dedicated thread is used. All clients connected to an application form a collaboration group by default. Global updates (e.g. current application status) are automatically broadcast to this group. Additionally clients can form or join (or leave) collaboration sub-groups with the application group. Once part of a collaboration group, the client can selectively broadcast application information to the group. Clients can also select the type of information it should receive. This allows clients to enable only those views that it can handle, e.g. a client with limited graphics capability may disable all graphical views. Finally, clients can disable all collaboration so that their requests/responses are not broadcast to the entire collaboration group. Individual views can still be explicitly shared in this mode. In addition to view/command collaboration, each application on the client portal is provided with chat and whiteboard tools to further assist collaboration.

5. Daemon Servlet and Application Proxies

The Daemon Servlet forms the bridge between the server and the applications. This servlet opens 3 communication channels with each application that connects to it: (1) A *MainChannel* for application registration and regular updates; (2) A *CommandChannel* for forwarding client interaction requests to the application; and (3) A *ResponseChannel* for receiving application responses to the interaction requests. Each application is authenticated at the server using a pre-assigned unique identifier. The Daemon Servlet creates an *Application Proxy* for each new application that connects to it, and maintains a handle to the proxy object. It also assigns the application with a unique session identifier. The Application Proxy object encapsulates the entire context for an application. It spawns two threads – one for the initial application registration and subsequent updates and a second for receiving responses to view/command queries. All updates and responses from the application are logged on a per-client as well as a per-session basis. This log is used to prevent multiple requests for the same information from being sent to the application. The

Command Channel buffers all requests and sends them to the application only the application is in the "interaction" phase. This ensures that requests are not lost while the application is busy computing.

B. DISCOVER Auxiliary Services

1. Session Archival Handler

The session archival handler maintains two logs. The first logs all interactions between client(s) and the application and enables clients to replay their interactions with the application. It also enables latecomers to a collaboration group to get up-to-speed. The second log maintains all global updates and status messages from each application. This log allows clients to have direct access the entire history of the application. Logging uses standard JDBC interfaces, and local and/or remote databases. All requests, commands, responses, and global updates are stored in memory at the server and synched to the database at regular intervals. At the client end, a customizable log-viewer provides access to the logged information, sorted by interaction epochs. Interaction epochs correspond to each time the application is in its interaction phase. Clients can replay their interactions with application during each such epoch.

2. View Handlers (Plug-Ins)

Application information is presented to the client in the form of application *Views*. Typical views include text strings, plots, contours and iso-surfaces. Associated with each of these views is a view plug-in that is used to present the requested view to the user. The server supports an extendible plug-in repository and allows users to extend, customize or create new views by registering custom mime types and the associated plug-ins with the DISCOVER server. Plug-ins are registered as executable jar files, and can be selectively downloaded from the discover server. For example, in the current implementation plotting views are based on the Java 3D API and use the Ptolemy [33] software package. These plots are of two kinds: iterative or one time. The former shows 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.

V. APPLICATION CONTROL NETWORK FOR INTERACTION AND STEERING

The Distributed Interactive Object Substrate (DIOS) constitutes the back-end of DISCOVER and is composed of two key components: (1) Interaction Objects that are co-located with computational objects and encapsulate sensors and actuators; and (2) A hierarchical control network that connects these objects with different interaction agents.

A. Sensors/Actuators and Interaction Objects

Interaction objects extend application computational objects with interaction and steering capabilities, by providing them with co-located sensors and actuators. Computational objects are the data-structures/objects used by the application. Sensors enable the object to be queried while actuators allow it to be steered. Efficient abstractions are essential for converting computational objects to interaction objects especially when the computational objects are distributed and dynamic. In DISCOVER, this is achieved by deriving the computational objects from a virtual interaction base class provided by the DIOS library. The derived objects define their interaction interfaces as a set of *View*s that they can provide and a set of *Commands* that they can service. *View*s represent sensors and define the type for information that the object can provide. For example, a *Grid* object might export views for its structure and distribution. Similarly, a *GridFunction* (application field defined on a grid) object might export views such as iso-surface plots, norms, and maximum/minimum values. *Commands* represent actuators and define the type of controls that can be applied to the object. Commands for the *Grid* object may include refine, coarsen, and redistribute. Similarly, those for the *GridFunction* may set or reset its data values. Interaction agents, which are part of the DIOS control network described below and are present at each computational node, export this interface to the interaction server using a simple *Interaction IDL* (Interface Definition Language). The Interaction IDL can be easily interfaced to standard distributed object frameworks such as CORBA and Java RMI. 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 layout. DISCOVER interaction objects can be created

or deleted during application execution and can migrate between computational nodes. Furthermore, a distributed interaction object can modify its distribution at any time.

1. Local, Global & Distributed Interaction Objects

Interaction objects can be classified based on the address space(s) they can span during the course of computation as *local*, *global*, and *distributed objects*. Local interaction objects are restricted to the address space of a single computational node at any time. These objects may however migrate to from one node to another during the lifetime of the application. Multiple instances of a local interaction object may exist on different processors at the same time. Global interaction objects are similar to local objects, except that there can be exactly one instance of the object (across all processors) at any time. A distributed interaction object spans multiple processors' address spaces. An example is a distributed array partitioned across available computational nodes. These objects contain an additional *distribution* attribute that maintains its current distribution type (blocked, inverse space filling curve-based, or custom) and layout. This attribute can change during the lifetime of the object if the object is redistributed. Like local and global interaction objects, distributed objects can be dynamically created, deleted, or redistributed. In order to enable interaction with distributed objects, each distributed type is associated with *gather* and *scatter* operations. Gather aggregates information from distributed components of the objects while scatter performs the reverse operation. Canned gather/scatter operations are provided for popular distributions (block-based, Space Filling Curve-based, etc.). Custom distributions can be defined by explicitly providing these operations.

2. Definition and Deployment of Interaction Objects

Transforming an existing computational object into an interaction object is performed in two steps:

1. The computational object is derived from an appropriate virtual interaction class, depending on whether they are local, global or distributed.
2. Views and commands relevant to the computational object are defined and registered. This involves defining and implementing the methods that will perform the desired functionality (generate a view or

execute a command), if they do not already exist. Registering a view/command consists of providing a name for the view/command and a callback that is invoked to process an associated request. For example, computing the desired one-dimensional slice corresponding to a 1-D Plot view; or setting the value of a variable in response to a *SetValue* command.

Non-object-oriented (C/Fortran) data-structures can converted into interaction objects by first defining C++ wrappers to the objects. The resulting computational objects are then converted into interaction objects as described above. Although this requires some application modification, the wrappers are only required for those data-structures that have to be made interactive, and the effort is far less than rewriting the entire application to be interactive. We have successfully applied this technique to enable interactivity within the Fortran-based IPARS parallel oil-reservoir simulator [9] developed at the Center for Subsurface Modeling, University of Texas at Austin.

B. A Control Network for Interaction and Steering

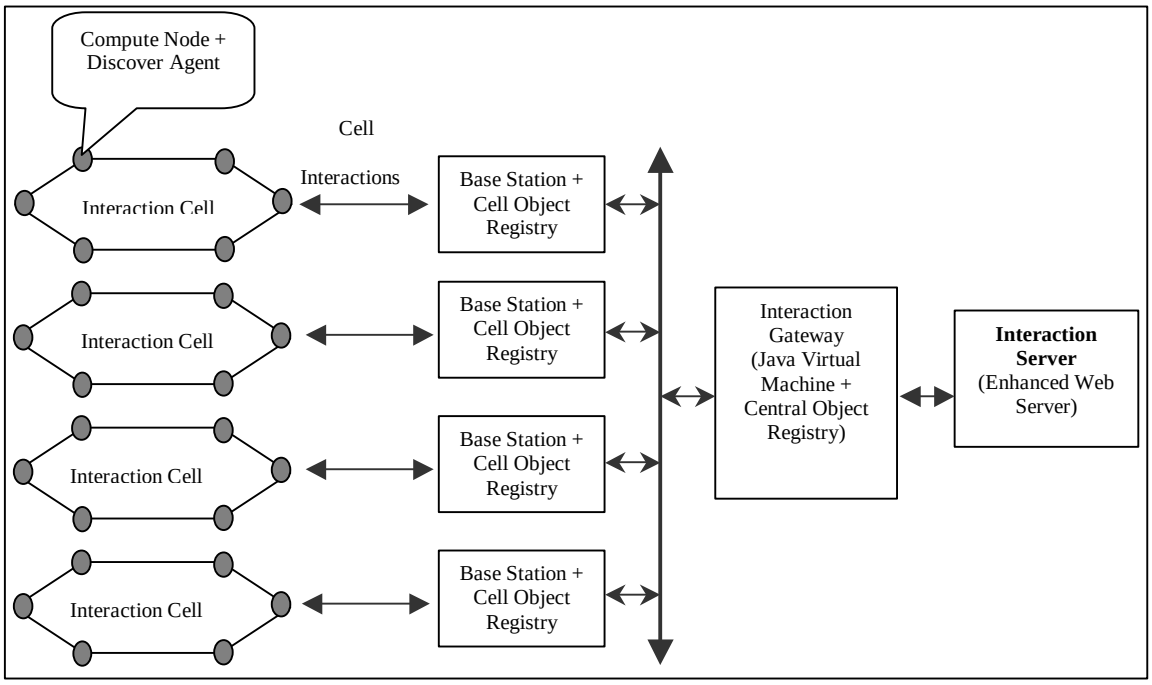

Figure 2 - Control Network of Computation and Interaction Nodes within the

The control network has a hierarchical structure as shown in Figure 2. Computational nodes are partitioned into *interaction cells*, each cell consisting of a set of *Discover Agents* and a *Base Station*. The number of nodes per interaction cell is programmable. At the top of the hierarchy is the *Interaction Gateway* that provides a Java-enabled proxy to the entire application. The cellular control network is automatically configured at run-time using an underlying messaging environment (i.e. MPI [18]) and the available number of processors.

1. Discover Agents, Base Stations and Interaction Gateway

Every computation node houses a *Discover Agent* (DA) that maintains a *local object registry* of all interaction objects currently active and registered by that node and maintains references to them. DA's exports interaction interfaces for all objects in theirs local registry (using the interaction IDL) to their corresponding Base Stations. *Base Stations (BS)* form the next level of control network hierarchy. They maintain interaction object registries containing interaction interfaces only, for an entire interaction cell and export these to the Interaction Gateway. The *Interaction Gateway* (IG) provides an interaction proxy for the entire application. It exports the interaction interfaces provided by the all interaction objects and is responsible for interfacing with external interaction servers or brokers, delegating interaction requests to the appropriate base stations and discover agents, and for combining and collating responses. Object migrations and re-distributions are handled by the respective DAs (and BSs if the migration/re-distribution is across interaction cells) by updating corresponding registries. Interactions between the Server and the IG are achieved using two approaches. In the first approach, the IG connects to the Server and performs object serialization to export all the interaction objects exported by the application to the server. A set of Java classes at the server parses the interaction IDL stream to de-serialize the interaction objects. In the second approach, the Interaction Gateway uses the Java Native Interface [23] to create Java mirrors of registered interaction objects. These mirrors are registered with a RMI (Remote Method Invocation) [21] registry service also executing at the IG. This enables the Server to gain access to and

control the interaction objects using the Java RMI API. We are currently evaluating the performance overheads of using Java RMI and JNI.

C. Control Network Initialization and Interaction Sequences

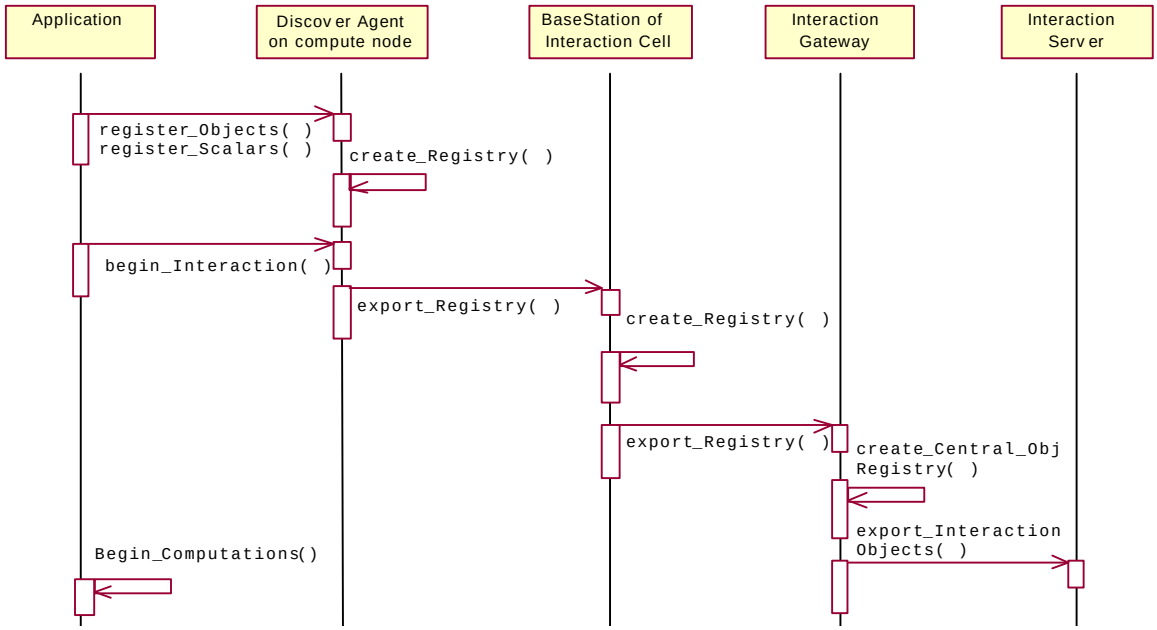

Figure 3 - Sequence of interactions during the control network initialization

The initialization sequence of interactions between the Discover Agents, corresponding Base Stations, Interaction Gateway, and the external Interaction Server (IS) are illustrated in Figure 3. The application, using the DIOS API, creates and registers its interaction objects with its local DAs. The BSs setup interaction cells, and establish communication with their respective DAs to initialize their cell object registries. At the IG, the central object registry is created. The DAs now export their object registries to their respective BSs who in turn forward them to the IG. The IG now communicates with the IS to register the application and export the central object registry to the IS. At the IS, the interaction IDL messages are parsed and interaction objects are recreated. Once the initial object registration process is complete, the application begins its computations.

The application interaction phase is shown in Figure 4. The IG looks for any outstanding interaction requests from the IS. If there are any incoming requests, it parses the request headers to identify the compute node from which the object was exported. In the case of a distributed object, this would be a set

of nodes. The interaction request is now forwarded to the respective compute node(s). The IG then waits until the corresponding response arrives from the DAs. If the responding object is distributed, the IG performs a gather operation on the individual responses. The response then shipped to the IS. At the end of the interaction phase, the IG sends a *go-ahead* message to compute nodes. phase.

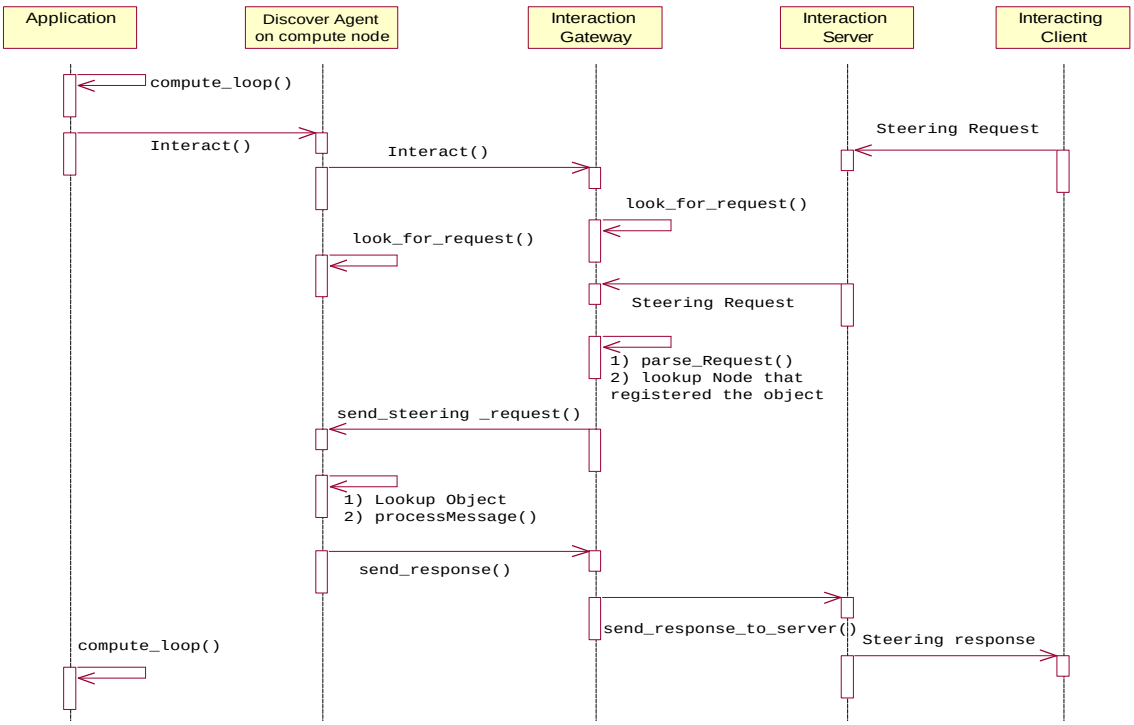

Figure 4 - Sequence of events that occur during an interaction phase

1. Interacting with Local and Distributed Objects

The processing of interaction requests is slightly different for local and distributed objects. In the case of a local object residing on a single computational node, processing is straightforward. On receiving the request from the IS, IG parses the message header to identify the computational node that registered the object. The steering request is then forwarded to the appropriate node. The corresponding DA on the node uses its reference to the associated interaction object to process the request. The response generated is then sent back to the IG, which in turn, exports it to the IS. This process is illustrated in Figure 5.

Processing interaction requests in the case of a distributed object is shown in Figure 6. The IG once again parses the message header to identify the nodes across which the object is distributed. The Gateway then forwards the steering request to these nodes. The corresponding DAs receive the steering request, look up the associated interaction objects and locally process the message. Each DA sends its portion of the response back to the IG. The IG then performs a *gather* operation to collate the responses and forwards them to the IS.

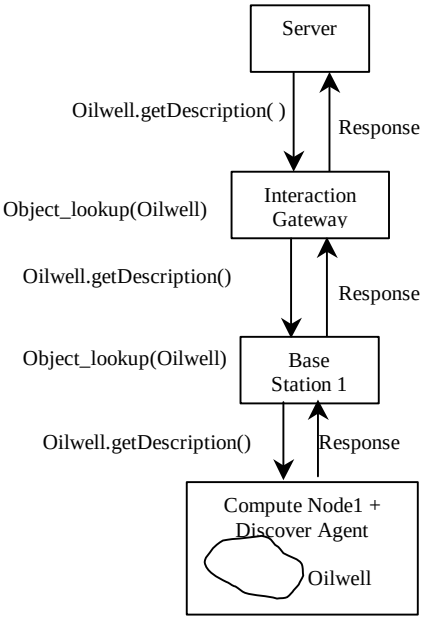

Figure 5 - Processing a View Request for a Non Distributed Object

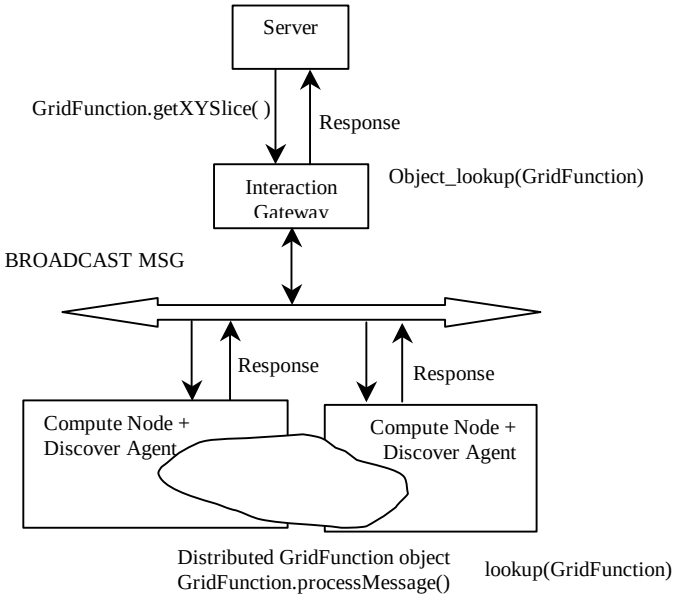

Figure 6 - Processing a View Request for a Distributed Object

D. Experimental Evaluation

This section summarizes the results of an experimental evaluation of the DIOS library using the Sun E10000 cluster. The evaluation consists of 4 experiments.

1. *End-to-end Steering Latency* – The DISCOVER system exhibits latencies varying between 10 - 45 ms for transfer of data sizes ranging from a few bytes to 10KB. This is comparable to steering systems like the MOSS and Autopilot systems, as reported in [7].
2. *Minimum Steering Overhead* – In the minimum steering mode, the application continuously updates the external interactivity system (web server and collaborating clients) with changes in the important

steering parameters of the simulation. The overhead incurred in exporting scalars were measured with respect to the average time spent in a computation iteration and this was found to be within a small fraction of the time spent in computation (ranging from 1% for exporting a single scalar parameter to about 5% for exporting 10 scalars).

3. *Object Registration Overhead* – One of the key sources of overheads was object registration process, including interaction IDL generation and exporting at the Discover Agents (to the Base Station), and IDL processing and exporting at the Base Station and Gateway. These steps are necessary for registering the interaction objects at startup. The different overheads measured were (1) 500 ?sec at each Discover Agent, (2) 10 ms at each Base Station for each compute node in its interaction cell and (3) 10 ms at the Gateway for each Base Station in the control network. We are current working on optimizing the registration process. Note that this is a one-time cost required only at startup.
4. *Query Processing and Steering Overhead* – This cost largely depends to the nature of interaction/steering requested, and the processing required at the application to satisfy the request and generate a response. In the experiments conducted, data sizes generated (for View requests) ranged from a few bytes to about 10 KB and this took between 10-45 ms. Command processing took about 30 ms to refine a grid hierarchy, 1.2 sec to checkpoint execution state to a file and 45 ms to rollback to a previous checkpoint state and resume execution. In this experiment, distributed collaborating clients generated all view and command requests.

VI. THE COLLABORATIVE INTERACTION AND STEERING PORTAL

Web portals, seamlessly bringing multiple services to the user, are becoming more and more common in the Internet development environment since they were first introduced by AOL. The DISCOVER collaborative computational portal can be seen as a working environment for scientists, empowering them with an anytime/anywhere capability of collaboratively (and securely) monitoring and controlling applications, independent of platform architecture or geographic location. Figure 7 shows a screen dump

of the current portal. The portal combines PHP [31], Java and Java servlet technologies and uses mySQL [32] as the back-end database.

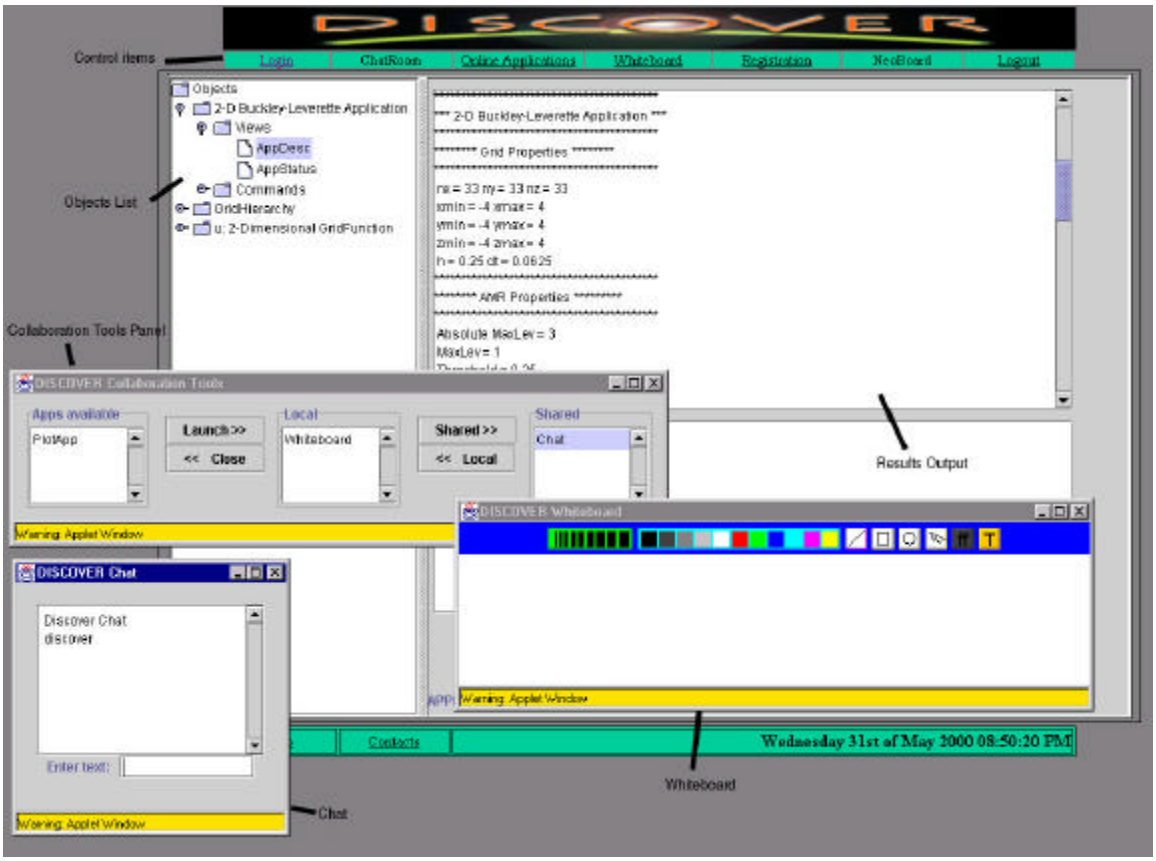

Figure 7 – The DISCOVER Collaborative Interaction/Steering Portal

A. Portal Elements and Architecture

The portal integrates access to DISCOVER services. The base portal, presented to the user after authentication and application selection, is a control panel. The control panel is designed to be lightweight as all clients irrespective of their capabilities must be able to download it. Once the client has the control panel s/he can launch any desired service such as view interrogation, interaction, collaboration, or application/session archival access. The application control panel is a Java swing applet and consists of: (1) a list of interaction objects and their exported interaction capabilities (views and/or commands), (2) an information pane that displays global updates from the selected application, (3) a list of all users logged on to a particular application, (4) a response pane for all textual responses and (5) a status bar that displays the current mode of the application (computing, interacting) and the status of command/view

request. The list of interaction objects is customized to match the client's access privileges. Clients having privilege to issue commands have to acquire a lock from the server for steering the application. Locks are maintained by the DISCOVER server and are assigned on a consensus basis. Chat and whiteboard tools enable collaboration. View requests for graphs/plots generate separate panes using the corresponding view plug-in. A separate application registration page is provided to allow super-users to register applications, add users and modify user capabilities.

All communication between the server and the client applet is based on Java's capability of sending serialized objects. All status messages, responses, chat messages, whiteboard events, error messages and the object lists are shipped out from the server as objects. The main portal applet is multithreaded - one thread polls for global updates from the application while another polls for responses from the server to requests issued by either the client (no collaboration mode) or by other clients (collaboration mode). A separate thread handles chat and whiteboard events. The main thread manages the interaction object list and sends command/view request to the server.

VII. DEPLOYING THE DISCOVER COMPUTATIONAL COLLABORATORY

DISCOVER has current been deployed at www.discoverportal.org and can be used by any scientific/engineering simulation to enable collaborative interaction and steering. The steps involved in integrating an application with DISCOVER include:

1. **Registration**: Each application must be registered with the DISCOVER system using the web registration form to receive a unique application identifier and the name of the server it should connect to. Application clients and their capabilities and privileges are specified during registration.
2. **Creation of Interaction Objects**: Use the DIOS library (DIOS is currently supported on SGI, Linux/Beowulf, SP and Sun E10000 systems) to construct interaction objects from the computational data-structures, and to export their interaction interfaces to the server.

3. **Application Execution**: Use the acquired application identifier to authenticate the application with the DISCOVER interaction web server at runtime (using a DIOS API call). During execution, application information is automatically exported to the server by the interaction objects.
4. **Application Interaction and Steering**: Distributed clients can now point their web browsers to the DISCOVER interaction web server, browse for running applications and join interaction and/or collaboration sessions. The application is now enabled for use with the DISCOVER computational collaboratory.

VIII. CONCLUSIONS, CURRENT STATUS AND FUTURE WORK

This paper presented the design and implementation of the DISCOVER computation collaboratory, a collaborative PSE for interaction and steering parallel/distributed applications. DISCOVER 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, and interaction agents superimposed on the application at the back-end. The DIOS interactive object framework enables easy deployment of sensors and actuators in existing applications. This framework can handle both distributed and dynamic objects. The architecture of the control network interconnecting these sensors and actuators is designed to be hierarchical so that it can scale to large parallel and distributed systems. The interaction gateway provides an interaction "proxy" to the application and enables web-based access to the application via the interaction server. An experimental evaluation of DIOS framework was also presented. To further reduce the end-to-end application response latency, a model for multithreaded interactive steering is being developed. DISCOVER is currently operational and is being used to provide these capabilities to a number of application specific PSEs including (1) the IPARS oil-reservoir simulator system at the Center for Subsurface Modeling, University of Texas at Austin, (2) The virtual test facility at the ASCI/ASAP Center, California Institute of Technology, and (3) Astrophysical Simulation Collaboratory at Washington University. We are currently working on extending the current single server to a network of interconnected interaction/collaboration servers, where an application can connect to any server, and

clients connected to a server can access applications connected to local/remote servers. As the servers are typically interconnected through a high bandwidth link, clients can connect to the closest server and have access to remote applications. Server-server interactions are designed to use CORBA, and application proxies can now refer to an application executing on a remote server. The key advantage provided by CORBA is scalability. Since we assume high bandwidth links between the servers, and caching mechanisms are used for client requests and application response objects, the overheads of using CORBA are greatly reduced. We are also evaluating the benefit of mirroring interaction objects using JNI and using Java RMI-based interaction between the gateway and the server. Finally, we exploring integration of the DISCOVER portals with other portal efforts such as the Globus CoG [12] and the Grid Portal Collaboration [35].

IX. REFERENCES

[1]. Gu. W., Vetter J., Schwan K. "*Computational steering annotated bibliography*", Sigplan notices, 32 (6): 40-4 (June 1997).

[2]. Weiming Gu, Jeffrey Vetter, and Karsten Schwan. "*An Annotated Bibliography of Interactive Program Steering*", GIT-CC-94-15, also in ACM SIGPLAN Notices, July 1994.

[3]. Mulder J., Jack van Wijk and Robert van Liere. "*A Survey for Computational Steering Environments*", Future Generation Computer Systems, Vol. 15, nr. 2, 1999.

[4]. Jeffrey Vetter and Karsten Schwan. "*Models for Computational Steering*", Third International Conference on Configurable Distributed Systems, IEEE, May 1996.

[5]. Jeffrey Vetter and Karsten Schwan. "*High Performance Computational Steering of Physical Simulations*", International Parallel Processing Symposium (IPPS), IEEE, Geneva, April 1997.

[6]. B. Schroeder, G. Eisenhauer, K. Schwan, J. Heiner, P. Highnam, V. Martin and J. Vetter. (1997), "*From Interactive Applications to Distributed Laboratories*".

[7]. Greg Eisenhauer. "*An Object Infrastructure for High-Performance Interactive Applications*", PhD thesis, Department of Computer Science, Georgia Institute of Technology, May 1998

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

[9]. 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.

[10]. 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.

[11]. Erol Arkarsu, Geoffrey Fox, Wojtek Furmanski, Tom Haupt, Hasan Ozdemir, Zeynep Odcikin Ozdemir, Tom Haupt. "*Building Web / Commodity Based Visual Authoring Environments for Distributed Object / Component Applications - A Case Study Using NPAC WebFlow Systems*". http://www.npac.syr.edu/Projects/WebSimulation/WebFlow.

[12]. Gregor von Laszewski, Ian Foster, Jarek Gawor, Warren Smith and Steven Tuecke. "*CoG Kits: A Bridge between Commodity Distributed Computing and High Performance Grids*". Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439, USA. http://www.mcs.anl.gov/~laszewsk/cog.

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

[14]. 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.

[15]. J. Vetter and K. Schwan. "*Progress: A Toolkit for Interactive Program Steering*", Proceedings of the 1995 International Conference on Parallel Processing, pp. 139-149. 1995.

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

[17]. Randy L. Ribler, Jeffrey S. Vetter, Huseyin Simitci, and Daniel A. Reed. "*Autopilot: Adaptive Control of Distributed Applications*", Proceedings of the 7th IEEE Symposium on High-Performance Distributed Computing, Chicago, IL, July 1998.

[18]. MPI Forum. "*MPI: Message Passing Interface*". Technical Report CS/E 94-013, Department of Computer Science, Oregon Graduate Institute, March 1994.

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

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

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

[22]. Hunter J. *Java Servlet Programming*. 1st edition, O'Reilly, California (1998).

[23]. Gordon R. *Essential JNI: Java Native Interface*. 1st edn. Prentice Hall, New Jersey (1998).

[24]. Eisenhauer G., Schwan K. "*An Object-Based Infrastructure for Program Monitoring and Steering*". 2nd SIGMETRICS Symposium on Parallel and Distributed Tools (1998).

[25]. Bajaj C., Cutchin S. "*Web based Collaborative Visualization of Distributed and Parallel Simulation*", IEEE Parallel Symposium on Visualization (1999).

[26]. 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.

[27]. Jain L.K. "*A Distributed, Component-Based Solution for Scientific Information Management*". MS Report, Oregon State University (1998).

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

[29]. Raje R.R., Teal A., Coulson J, Yao S., Winn W., Guy III E. "*CCASEE – A Collaborative Computer Assisted Software Engineering Environment* ". Proceedings of the International Association of Science and Technology for Development (IASTED) Conference (1997).

[30]. Boyles M., Raje R., Fang S. "*CEV: Collaboration Environment for Visualization Using Java RMI* ". Proceedings of the ACM Workshop on Java for High-Performance Network Computing (1998).

[31]. PHP Hyper Processor, http://www.php.net

[32]. MySQL, http://www.mysql.com

[33]. 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.

[34]. Myproxy (v 1.0), National Laboratory for Applied Network Research, http://dast.nlanr.net/Features/MyProxy/, July 2000.

[35]. Grid Portal Collaboration, https://palomar.extreme.indiana.edu/.