Engineering a Distributed Computational Collaboratory[1]

Samian Kaur, 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: {samian,vijay,vincentm,rajeevdm,parashar}@caip.rutgers.edu

Abstract

This paper presents the design, implementation, and deployment of the DISCOVER 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 monitoring, interaction and control. DISCOVER supports a 3-tier architecture composed of detachable thin-clients at the front-end, a network of interactions servers in the middle, and a control network of sensors, actuators, and interaction agents at the back-end. The interaction servers enable clients to connect 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. The application interaction gateway manages overall interaction. It uses Java Native Interface to create Java proxies that mirror computational objects and allow them to be directly accessed at the interaction server. Security and authentication are provided using customizable access control lists and SSL-based secure servers.

1. 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 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 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 collaborative interaction and steering of high-performance parallel/distributed applications presents many challenges. A key issue is the definition and deployment 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 be able to control individual application data structures. 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 dynamic 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 (and consistently) 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 (Distributed Interactive Steering and Collaborative Visualization EnviRonment) web-based computational collaboratory. Its primary goal is to bring large distributed simulations to the scientists'/engineers' desktop by providing collaborative

[1] The research presented in this paper is sponsored in part by the National Science Foundation via grant number ACI 9984357 (CAREERS) awarded to Manish Parashar.

web-based portals for interaction and control. The DISCOVER architecture consists of detachable thin-clients at the front-end, a network of interactions servers in the middle, and a control network of sensors, actuators, and interaction agents at the back-end. The interaction servers enable clients to connect to and collaboratively interact with registered applications using a 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 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) [19] 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 and SSL-based secure servers.

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 control network, the application interaction substrate and its interface to the interaction server. Section VI describes the client collaborative interaction portal. Section VII presents conclusions and current and future work.

2. 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. Existing interactive and collaborative PSE's are classified and briefly describe below. The primary goal of DISCOVER is to combine the two capabilities to provide a collaborative PSE for application interaction and control.

1. *Systems for interactive program construction* - Systems in this category, e.g. *SCIRun* [15], provide support for interactive program construction. SCIRun allows this by graphically connecting components in a data-flow style graph. It is primarily targeted to developing new applications, and does not support simultaneous steering of multiple applications, or collaborative interaction and steering by multiple users.
2. *Systems for performance optimizations* – These systems are aimed at optimizing performance of applications. In the *Autopilot* [16] system, *sensors* have a variety of sensor policies that optimize application performance. However, these do not provide support for accessing application objects for interactive monitoring and steering.
3. *Systems for application remote configuration and deployment* – These systems use existing high performance metacomputing backend resources and provide powerful visual authoring toolkits to configure and deploy distributed applications. The CoG Kits [11] provide commodity access to the Globus[12] metacomputing environment. The WebFlow [10] and Gateway [9] provide support for configuring, deploying and analyzing distributed applications. These systems, however, do not provide any support for runtime application level interaction and steering.
4. Systems for interactive run-time steering and control
   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 [14] and Magellan [5] system use this approach and require a server process executing in the same address space as the application, to enable interaction. The Computational Steering Environment (CSE) [13] uses a *data manager* 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]) 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 using which steering actions are accomplished. MOSS is based on CORBA-style objects and has been implemented using a CORBA-compliant object-oriented language. The high-level abstractions for interaction and steering provide the most general approach for enabling interaction in applications. The DISCOVER control network extends this approach.
5. *Collaboration groupware*: These environments include *DOVE* [25], *Web Based Collaborative Visualization* [23] system, *NCSA Habanero* [24] system, *Tango* [26], *CCASE* [27] and *CEV* [28].

These systems primarily focus on enabling collaboration; some of them however do provide support for problem solving. The Tango system is based on centralized server, and is web browser enabled. Habanero (developed at NCSA, UIUC) uses a Java based centralized server architecture for web collaboration (and collaborative visualization). The CCASEE provides a distributed workspace using Java RMI. The CEV system provides collaborative visualization using a central server to perform the computations necessary to generate new collaborative views. The *DOVE* and the *Web Based Collaborative Visualization* systems also provide similar support for collaborative visualization.

The DISCOVER computational collaboratory brings together key technologies in web portals, web servers, collaboration and interaction and steering to provide (1) interaction mechanisms for distributed dynamic interactive objects that can span multiple address spaces and can be dynamically created and destroyed, (2) a scalable control network to connect distributed interaction objects, sensors and actuators and (3) collaborative, web-based interaction and steering portals for remote access to applications.

3. DISCOVER: An Interactive Computational Collaboratory

The DISCOVER computational collaboratory provides a virtual, interactive and collaborative PSE that enables geographically distributed scientists and engineers to collaboratively monitor and control high performance parallel/distributed applications. An architectural overview of the DISCOVER collaboratory is presented in Figure 1. DISCOVER is built using a 3-tier architecture. Its front-end is composed 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 control network composed 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 using high-level interfaces. 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 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 three DISCOVER components are described in the following sections.

4. Discover Interaction & Collaboration Servers

The DISCOVER interaction/collaboration server

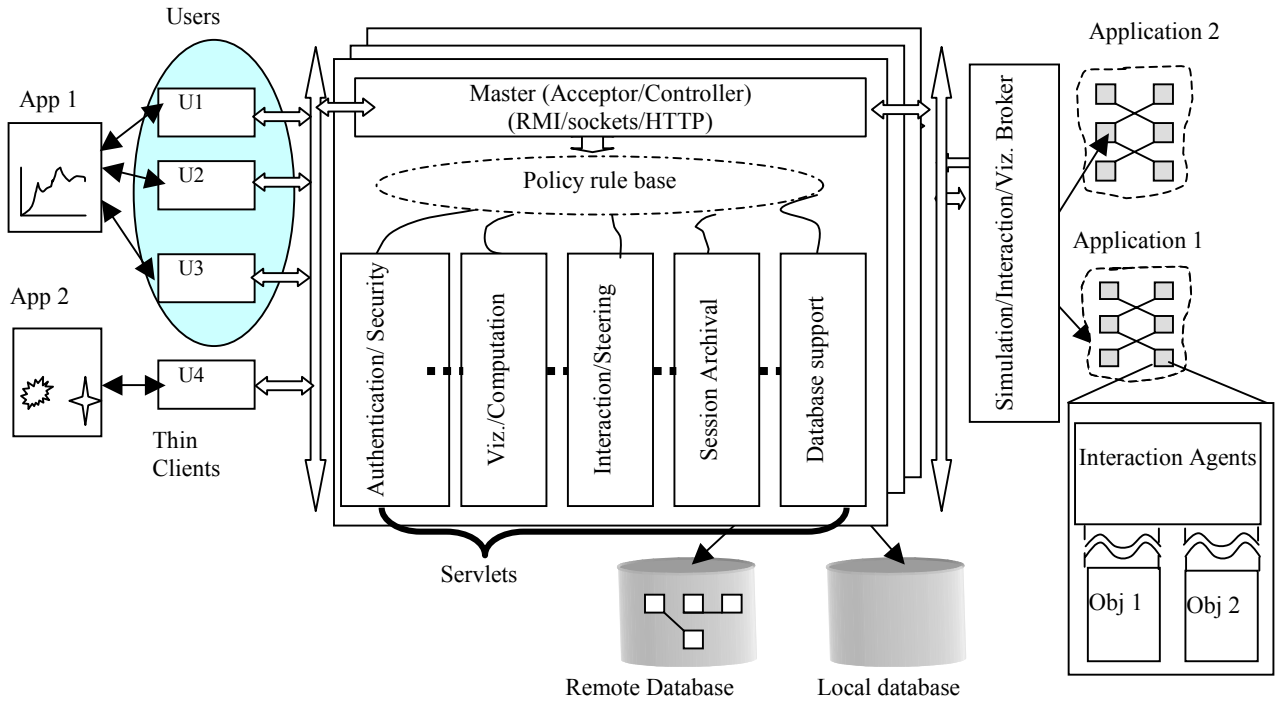

Figure 1. Architectural schematic of the DISCOVER computational collaboratory

builds on a traditional web server and extends its functionality to handle real-time information, and to serve client requests and application connections. Extension is achieved using Java servlets [21] (server side Java programs). Each server is a traditional web server with a number of handler servlets running to provide different interaction and collaboration services. Clients connect to the server using standard HTTP communication using a series of HTTP *GET* and *POST* requests. At the other end, application-to-server communication is achieved either using standard distributed object protocols like CORBA [18] and Java RMI [20], or a more optimized, custom protocol using TCP sockets. The core service handlers provided by each server include, the Master Handler, Collaboration Handler, Command Handler, Security/Authentication Handler and a Daemon Servlet that listens for application connections. In addition to these core handlers, there can be a number of handlers providing auxiliary services such as session archival, database handling, visualization, request redirection, and remote application proxy invocations (using CORBA). These services are optional and need not be provided by every server. The different services are described below:

4.1. Core Discover Services

4.1.1. **Master Handler.** The master (accepter/controller) handler servlet is the client's gateway to the server. It manages client service requests, such as authentication, session archival, request redirection, and view handling, and delegates them to the corresponding handler servlet. For each of these requests it invokes the corresponding handler, either on the local server or on a remote server using CORBA, if the request service handler is not available locally. Within a local server, the master servlet relies on reflection to dynamically invoke handlers, thus 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 to present application information requested by the clients.

4.1.2. **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 shown a list of the applications for 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. This ensures that a client can only access, interact with and steer an application in authorized ways. On the client side, digital certificates are used to validate the server identity before the client downloads views. A Secure Socket Layer provides encryption for all communication between the client and the server. To control access, 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. Each interaction request is then validated against the ACL before it is processed.

4.1.3 Command Handler. The command handler manages all client view/command requests. On receiving these requests from the master handler, this handler looks up the appropriate application proxy, and redirects them to this proxy. The collaboration handle 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 to the application.

4.1.4 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.

4.1.5 Daemon Servlet & 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.

4.2. DISCOVER Auxiliary Services

4.2.1. 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.

4.2.1. View Handlers (Plug-Ins). Application information is presented to the client in the form of application *View*s. 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 [31] 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.

5. Application Control Network for Interaction and Steering

The DISCOVER back-end is composed of two components, interaction objects that are co-located with computational objects and encapsulate sensors and actuators, and a hierarchical control network that connects these objects with interaction agents, interaction base stations and the interaction gateway.

5.1. Sensors/Actuators & 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 enables the object to be queried whiles actuators allow it to be steered. Efficient abstractions, essential for converting computational objects to interaction objects especially when the computational objects are distributed and dynamic are achieved by deriving the computational objects from a virtual interaction base class of the DISCOVER Distributed Interaction Object library. The derived objects define its interaction interface as a set of Views that they can provide and a set of Commands that they can accept. Views 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. 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. Interaction agents then export

this interface to the interaction server using a simple Interaction IDL (Interface Definition Language), which is compatible with standard distributed object interfaces like CORBA and RMI. 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.

5.1.1. 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*. A computational node creates local interaction objects locally. These objects may migrate to another processor during the lifetime of the application, but exist in a single processor's address space at any point of time. Multiple instances of a local object could 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 distributed components of the objects while scatter performs the reverse operation.

5.1.2. **Definition and Deployment of Interaction Objects.** In the presented framework, interaction objects are defined by deriving an existing computational object from a library of interaction virtual classes provided. Transforming an existing computational object into an interaction object is performed in two steps: The computational object is derived from an appropriate virtual interaction class, depending on whether they are local, global or distributed. 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. This step is accomplished by overriding specific virtual functions of the interaction base class the computational object is derived from. 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.

5.2. Control Network for Interaction and Steering

The control network has a hierarchical "cellular" structure with three components 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. Discover Agents are present on each computational node and manage run-time references to the interaction objects on the node. The Base Station maintains information about interaction objects for the entire interaction cell. The highest level 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 and the available number of processors.

5.2.1 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. Each DA exports the interaction interfaces for the objects in the local registry (using the interaction IDL). *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) represents an interaction proxy for the entire application. It exports the interaction interfaces provided by the all interaction objects of the application and is responsible for interfacing with external interaction server or brokers, delegating interaction requests to the appropriate base stations and discover agents, and for combining and collating responses. Object migrations and re-distributions

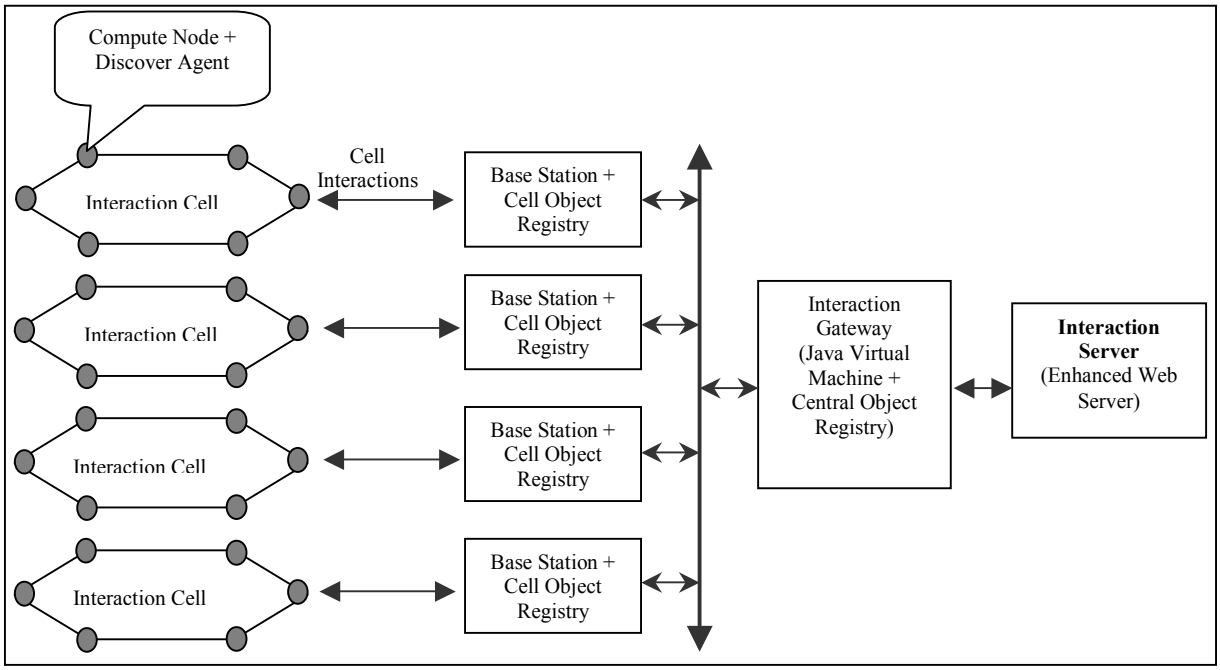

Figure 2. Control Network for Interaction and Steering

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 [19] to create Java mirrors of registered interaction objects. These mirrors are registered with a RMI (Remote Method Invocation) [20] 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.

5.3. Control Network Initialization and Interaction Sequences

In the initialization sequence, 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 object registries to their respective BSs that are forwarded to the IG. The IG now communicates with the Server to registers the application and exports the central object registry to the Server. At the Server, the interaction IDL messages are parsed and interaction objects are recreated. Once the initial object registration process is complete, the application begins its computations.

In the interaction phase, the IG looks for any outstanding interaction requests from the server. If there are any incoming requests, it parses the header of the request to identify the compute node from which the object was exported and the interaction request is now forwarded to the destination node(s) identified. All other nodes are sent a *go-ahead* message indicating that there is no interaction request for any of the objects they registered during this interaction phase. 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 server.

5.3.1 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.

Processing interaction requests in the case of a distributed object is similar. 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.

5.4. Experimental Evaluation

This section summarizes 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 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 [8][16].
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 μsec 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.

6. The collaborative interaction and steering portal

Web portals, the seamlessly bring multiple services to the user using conventional web-browsers, are becoming more and more common in the Internet development environment. 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 3 shows a dump of the current DISCOVER portal. The DISOCVER portal design combines PHP [30], Java and Java servlet technologies and uses mySQL [32] as the back-end database.

6.1. Portal Elements and Architecture

The portal integrates access to DISCOVER services. The base portal, presented to 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 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, and (3) a status bar the 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. Chat and whiteboard tools enable collaboration. View requests generate separate panes using the corresponding the view plug-in. A separate application registration page is provided to allow super-users to register application, 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.

7. 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.

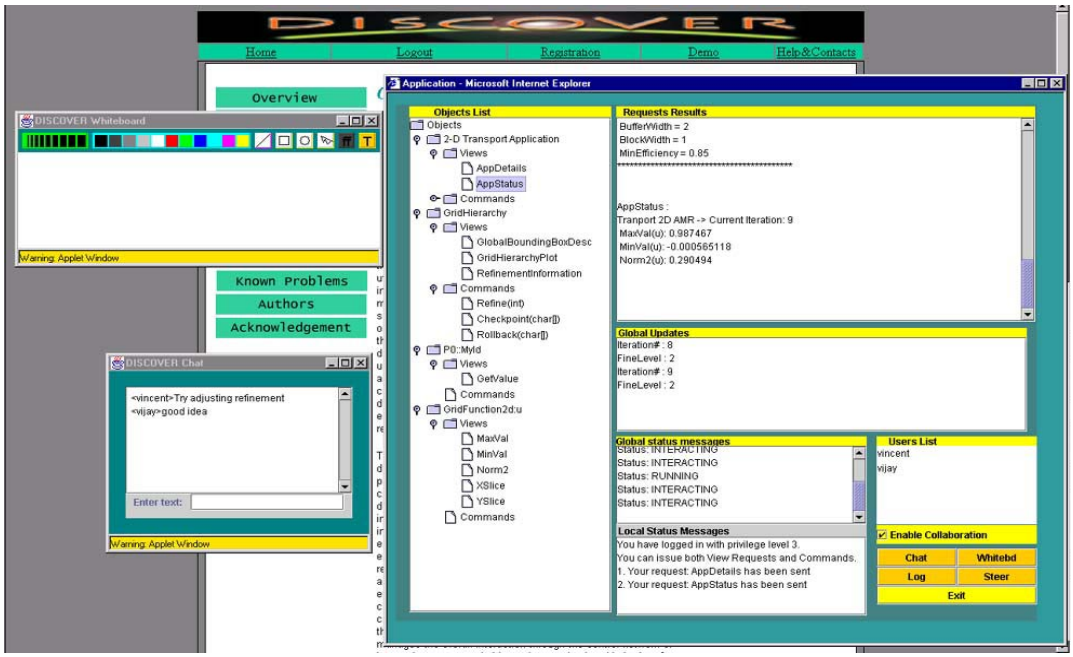

Figure 3. The DISCOVER Collaborative Interaction/Steering Portal

8. Reference

[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] 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.

[10] 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.

[11] 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.

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

[13] 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.

[14] 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.

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

[16] 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.

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

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

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

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

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

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

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

[24] 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.

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

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

[27] 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).

[28] 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).

[29] Gamma E., Helm R., Johnson R., Vlissides J.: Design Patterns, Addison Wesley Professional Computing Series (1994).

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

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

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