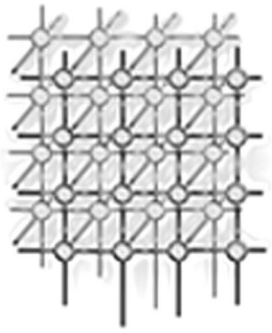
CONCURRENCY—PRACTICE AND EXPERIENCE
Concurrency: Pract. Exper. 2001; **13**:737–754 *Prepared using* cpeauth.cls *[Version: 2000/05/12 v2.0]*



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

V. Mann, V. Matossian, R. Muralidhar and M. Parashar*, †

Department of Electrical and Computer Engineering,
Rutgers University, 94 Brett Road, Piscataway, NJ 08854.
Tel: (732) 445-5388 Fax: (732) 445-0593

SUMMARY

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 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.

KEY WORDS: Web-based Computational Collaboratory, Computational Interaction & Steering

1. INTRODUCTION

Simulations are playing an increasingly critical role in all areas of science and engineering. As the complexity and computational cost 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

*Correspondence to: Manish Parashar CAIP/ECE, Rutgers University, 94 Brett Road, Piscataway, NJ 08854, USA
†E-mail: {vijay,vincentm,rajeevdm,parashar}@caip.rutgers.edu
‡The DISCOVER collaboratory can be accessed at `http://www.discoverportal.org/`

Contract/grant sponsor: National Science Foundation (CAREERS); contract/grant number: ACI 9984357

Received 31 July 2000
Revised 01 December 2000
Accepted 29 January 2001

them at runtime. The utility and cost-effectiveness of these simulations can be greatly increased by transforming 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 [1] [2] that can 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 sensors 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 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 the interaction objects and export their interaction interfaces. These agents coordinate interactions with distributed and dynamic objects. The application interaction gateway manages the overall interaction with the application. It uses the Java Native Interface (JNI) [3] 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 2. Section 3 outlines the DISCOVER system architecture. Section 4 presents the design, implementation, and operation of the interaction and collaboration web-server. Section 5 describes the design and implementation of control network, the application interaction substrate and its interface to the interaction server. Section 6 describes the client collaborative interaction and steering portal. Section 7 presents conclusions and outlines 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 evolved separately. Existing interactive and collaborative PSE's are classified and briefly describe below. Details surveys have been presented in [4] [5]. 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 [6], provide support for interactive program construction. SCIRun allows users to graphically connect program components to create a data-flow style program graph. It is primarily targeted to developing new applications. While SCIRun does provide some steering capabilities, it 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. For example, in the Autopilot [7] system, sensors have a variety of sensor policies that optimize application performance. However, they do not provide support for accessing application objects for interactive monitoring and steering.
3. **Systems for remote application configuration and deployment:** These systems use existing high performance metacomputing back-end resources and provide powerful visual authoring toolkits to configure and deploy distributed applications. The CoG Kits [8] provide commodity access to the Globus[9] metacomputing environment. The WebFlow [10] and Gateway [11] systems 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 [12] and Magellan [13] 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) [14] 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) ([15][16][17]) provides a high-level model for steering applications. Mirror objects are analogues to application objects (data structures) and are used for monitoring and steering. These object export application methods to the interactivity system through which steering actions are accomplished. 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 [18], the Web Based Collaborative Visualization [19] system, the Habanero [20] system, Tango [21], CCASE [22]

and CEV [23]. These systems primarily focus on enabling collaboration; some of them do however provide support for problem solving. The Tango system is based on centralized server, and is browser enabled. Habanero 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 supports a 3-tier architecture. Its front-end is composed of detachable client portals. Clients can connect to a DISCOVER 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. The interaction interfaces are defined and exported using high-level abstractions. Views encapsulate sensors and provide information about the application and the application objects, while commands encapsulate actuators and process steering requests. Some or all of these views/commands may be collaboratively accessed by groups of clients based on the clients' 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 builds on a traditional web server and extends its functionality to handle real-time application information and client requests. Extension is achieved

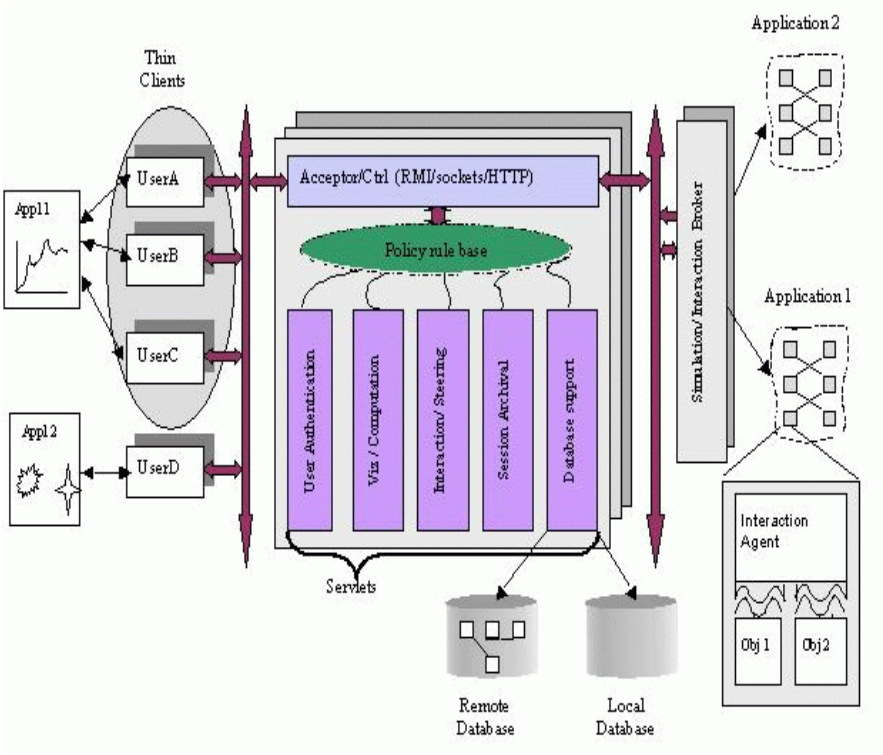


Figure 1. Architectural schematic of the DISCOVER computational collaboratory

using Java servlets [24] (server side Java programs). Each DISCOVER server is a web server with a number of "handler" servlets that provide 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 such as CORBA [25] and Java RMI [26], or a more optimized, custom protocol using TCP sockets. The core service handlers provided by each DISCOVER server include, the Master Handler, Collaboration Handler, Command Handler, Security/Authentication Handler and a Daemon Servlet that listens for application connections. In addition to the core handlers, there may 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 DISCOVER 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 servlet. 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 first level authorization, users are presented with a list of the applications to which they have access privileges. A second level authentication is then performed for the application they choose. Once authenticated, the authentication handler servlet builds a customized interaction interface for the client to match their access capabilities. This ensures that the client can only access, interact with and steer an application in authorized ways. To enable this access control, 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. 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.

4.1.3. Command Handler

The command handler servlet 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 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 to) the application at any time. Locks 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 servlet manages all collaboration, while on the client side a dedicated thread is used. All clients connected to a particular 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 within the application group. Once part of a collaboration group, the client can selectively broadcast application information to the group. A client 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 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 CommandChannel buffers all requests and sends them to the application when the application is in the "interaction" phase. This ensures that requests are not lost while the application is busy computing.

4.2. Auxiliary DISCOVER Services

4.2.1. Session Archival Handler

The session archival handler servlet 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 synchronized 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.

4.2.2. 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 [27] software package. These plots are of two kinds: incremental or one-time. The former shows the incremental change in an application parameter with successive iterations while the latter shows the history of the application parameter from startup or checkpoint to the current time.

5. Application Control Network for Interaction and Steering

The DISCOVER back-end is composed of two components provided by the Distributed Interactive Object Substrate (DIOS): (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 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 are essential for converting computational objects to interaction objects, especially when these computational objects are distributed and dynamic. In DISCOVER, this conversion is achieved by deriving the computational objects from a virtual interaction base class provided by DIOS. The derived objects define an interaction interface as a set of views that they can provide and a set of commands that they can accept and process. 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 (Discover 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.

5.2. Local, Global and 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 created in a processor's local memory. These objects may however migrate to another processor during the lifetime of the application, but always exist in a single processor's address space at any 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 that is replicated on 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, staggered, 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. Application can select from a library of gather/scatter methods for popular distribution types provided by DIOS, or can register customized gather/scatter methods for other distribution types.

5.3. Definition and Deployment of Interaction Objects

In DISCOVER, interaction objects are generated using a library of interaction virtual classes provides by DIOS. 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 callback methods that implement 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 the 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 defined by the underlying interaction base class. Non-object-oriented (C/Fortran) data-structures can be converted into interaction objects by first defining C++ wrappers for 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 [28] developed at the Center for Subsurface Modeling, University of Texas at Austin [†].

[†] see: http://www.ticam.utexas.edu/ ut/webipars/

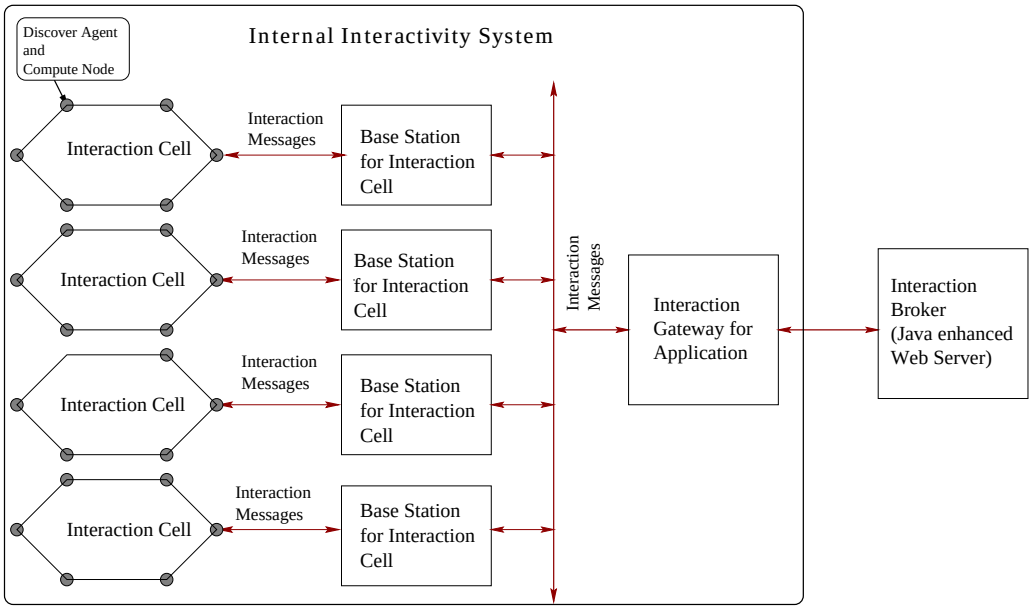


Figure 2. Control Network for Interaction and Steering

5.4. 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 (e.g. MPI [29]) and the available number of processors.

5.4.1. Discover Agents, Base Stations and Interaction Gateway

Every computation node in the control network houses a Discover Agent (DA). Each DA maintains a local interaction object registry containing references to all interaction objects currently active and registered by that node, and exports the interaction interfaces for the objects in its local registry (using the interaction IDL). The Base Stations (BS) form the next level of control network hierarchy. They maintain interaction object registries, containing interaction interfaces only, for the entire interaction cell and export these to the Interaction Gateway. The Interaction Gateway (IG) represents an interaction

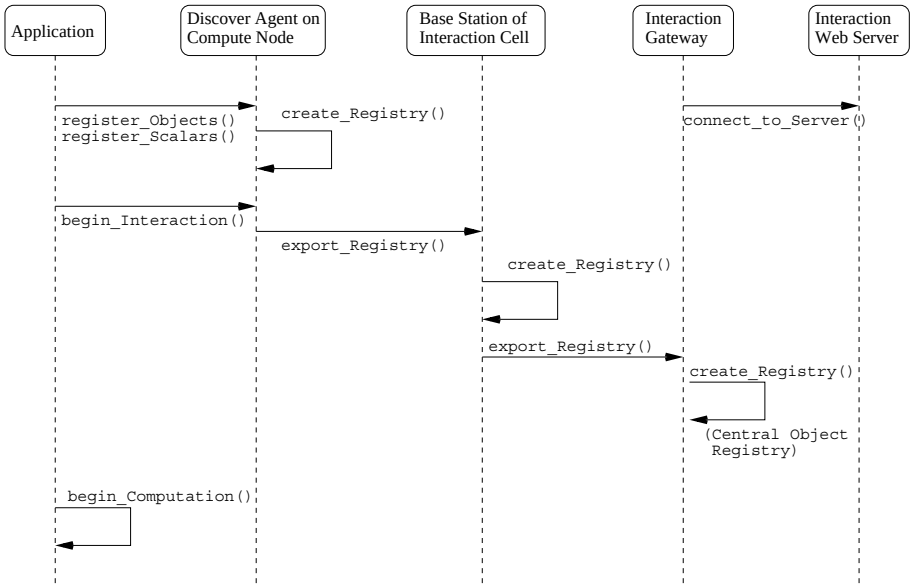


Figure 3. Sequence of interactions during the control network initialization

proxy for the entire application. It manages a registry of the interaction interfaces for all the interaction objects in the application, and is responsible for interfacing with external interaction servers or brokers. The IG delegates incoming interaction requests to the appropriate BSs and DAs, and combines and collates 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 an interaction server and the IG are achieved using two approaches. In the first approach, the IG explicitly serializes the interaction interfaces to the server. A set of Java classes at the server parse the serialized interaction IDL stream to generate the corresponding interaction object proxies. In the second approach, the Interaction Gateway uses the Java Native Interface [3] to create Java mirrors of registered interaction objects. These mirrors are registered with a RMI (Remote Method Invocation) [26] 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.

5.5. Control Network Initialization and Interaction Sequences

During DIOS control network initialization (see Figure 3), the application uses the DIOS API to creates and register its interaction objects with the local DAs. The interaction cells are then set up and the BSs establish communication with their respective DAs to initialize their cell object registries. At the IG, the central interaction object registry is created. The DAs now export local object registries to their

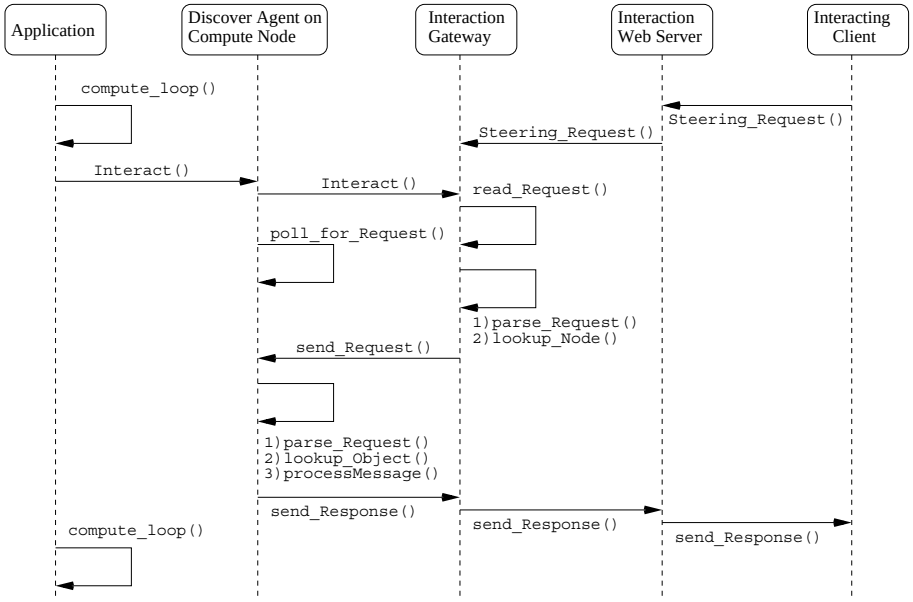


Figure 4. Sequence of events that occur during an interaction phase

respective BSs, who then forwarded them to the IG. The IG now communicates with the DISCOVER server, registering the application and exporting the central object registry. At the server, the incoming interaction IDL streams are parsed and interaction object proxies are recreated. Once the initial object registration process is complete, the application begins its computations.

During the interaction phase (see Figure 4), the IG looks for any outstanding interaction requests from the server. If requests exist, it parses the request headers to identify the compute node(s) where the corresponding object resides and forwards the interaction request to the node(s). All other nodes are sent a go-ahead message indicating that there is no interaction request for any of the objects they registered by the nodes. 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 is then shipped to the server.

5.5.1. 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 server, the IG parses the message header to identify the computational node where the object resides. 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

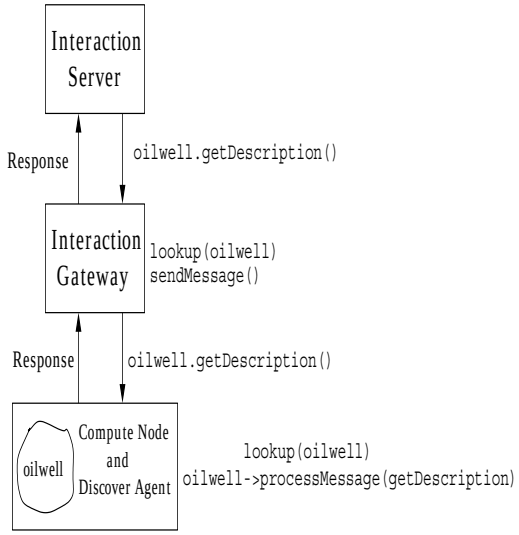


Figure 5. Processing interaction requests for local objects

response generated is then sent back to the IG, which in turn, exports it to the server. The 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 IG then forwards the 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 server.

5.6. Experimental Evaluation

This section summarizes an experimental evaluation of the DIOS library using a Sun Starfire E10000 cluster. The E10000 configuration used consists of 64, 400 MHz SPARC processors and an 12.8 Gbytes/sec interconnect. The E10000 provides hardware supported distributed shared memory. Applications used includes the IPARS reservoir simulator and the RM3d compressible turbulence kernel. A detailed discussion of the experimental evaluation is presented in [30]. The evaluation consisted of 4 experiments.

1. **Object Registration Overhead:** One of the key sources of overheads in DIOS is the object registration process during initialization. This includes generating and exporting (to the Base Station) the interaction IDL for each interaction object at the Discover Agents, and processing and the exporting the IDL at the Base Station and Gateway. The different overheads measured

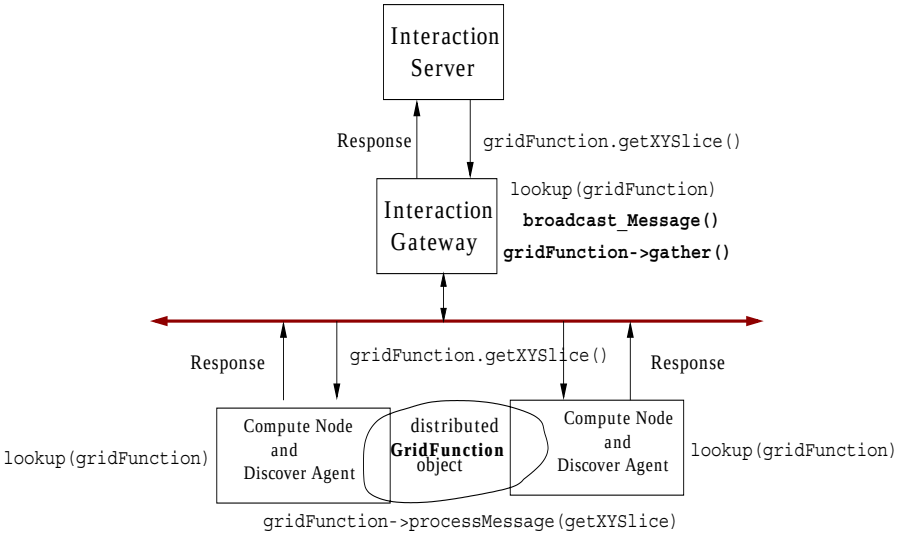


Figure 6. Processing interaction requests for distributed objects

were (1) 500 μs per object at each Discover Agent, (2) 10 ms per Discover Agent in the interaction cell at the Base Station and (3) 10 ms per Base Station in the control network at the Gateway. We are current working on optimizing the registration process to reduce this overhead. Note that this is a one-time cost required only at startup.

2. **Minimum Steering Overhead:** In the minimum steering overhead experiment, the application automatically updated the external interactivity system (web server and collaborating clients) with changes in a set of objects. In this experiment the explicit command/view requests (other than those automatically provided by the application) were disabled. The percentage overhead for generating and exporting the views with respect to the time spent on actual computation was measured. This overhead is plotted in Figure 7 for IPARS. This overhead was typically found to be less than 0.1% for most real applications.
3. **View/Command Processing Overhead:** The query processing and steering overheads largely depends on the nature of interaction/steering requested, and the processing required at the application to satisfy the request and generate a response. In this experiment we measured the view/commands processing (and exporting) overhead at a node. The view processing overheads for data sizes ranging from a few bytes to a KByte was between 1 and 2 ms. Similarly command processing overheads depended on the nature of the command. Commands such as stop, pause and continue required only about 250 μs to process while checkpoint and rollback commands requiring file I/O required more time. In this experiment, all distributed collaborating clients generated the same view and command requests. A sampling of this overhead for different view generations and command command executions is presented in Table I.

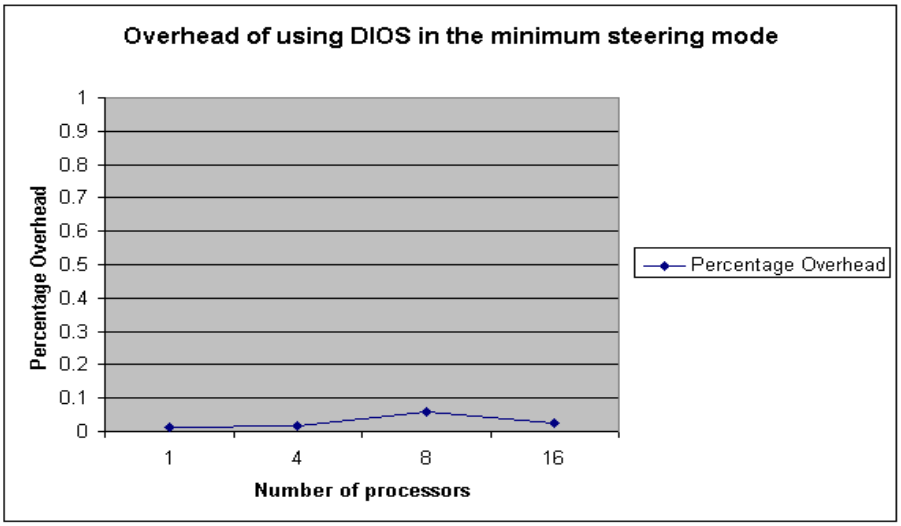


Figure 7. Minimum steering overhead (IPARS Reservoir Simulator)

Table I. View/command processing overheads

View Overheads			**Command Overheads**	
View Type	*Data Size*	*Time Taken*	*Command*	*Time Taken*
Text	65	1.4 ms	Stop, Pause or Resume application	250 μsec
Text	120	0.7 ms	Refine GridHierarchy	32 ms
Text	760	0.7 ms	Checkpoint to file	1.2 sec
XSlice Generation	1024	1.7 ms	Rollback to a previous Checkpoint	43 ms

4. **End-to-end Steering Latency:** This experiment measures the time to complete a round-trip steering operation starting with a request from a remote client and ending with the response delivered to that client. These measurements of course depend on the specification and state of the client, the server and the interconnect. The DISCOVER system exhibits end-to-end latencies between 10 - 45 ms for data sizes ranging from a few bytes to 10 KBytes. This is comparable to steering systems like the MOSS and Autopilot systems, as reported in [17] (see Figure 8) - these systems however do not support interactions with distributed or dynamic objects.

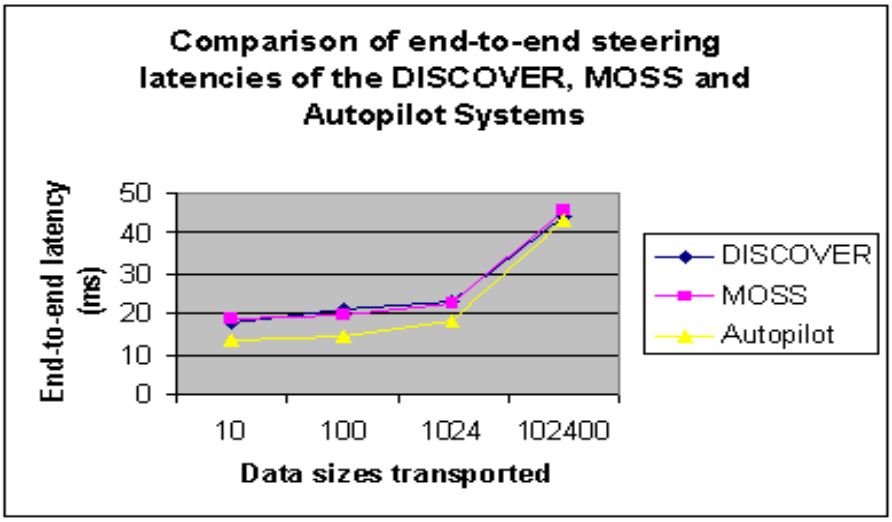


Figure 8. End-to-end steering latencies

6. The Collaborative Interaction and Steering Portal

Web portals, that seamlessly bring multiple services to the user using conventional web-browsers, are becoming increasingly 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 9 present a screen dump of the current DISCOVER portal.

6.1. Portal Elements and Architecture

The DISCOVER 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 clients download the control panel they can launch any desired service such as application interrogation, interaction, collaboration, or application/session archival access. The application control panel consists of: (1) a list of interaction objects and their exported interaction interfaces (views and/or commands), (2) an information pane that displays global updates (current timestep of a simulation) from the application, and (3) a status bar that displays the current mode of the application (computing, interacting) and the status of issued command/view requests. The list of interaction objects is customized to match the client's access privileges. Chat and whiteboard tools can be launched from the

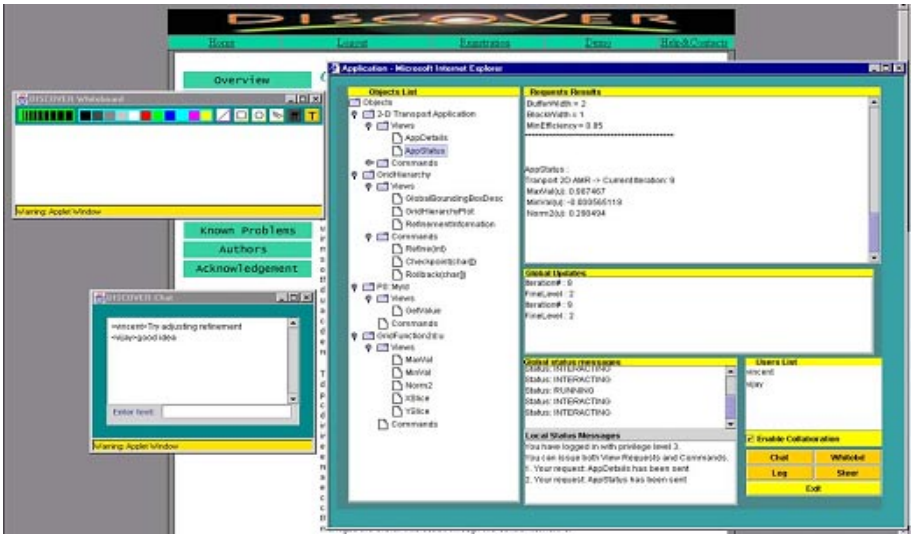


Figure 9. The DISCOVER Collaborative Interaction/Steering Portal

control panel to enable collaboration. View requests generate separate panes using the corresponding view plug-in. A separate application registration page is provided to allow super-users to register application, add application users and modify user capabilities.

The portal is implemented using a combination of PHP[31], Java and Java servlet technologies. It uses mySQL [32] as its back-end database. All communication between the server and the portal use Java object serialization. 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 requests to the server.

7. 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 interaction and collaboration servers in the middle, and a control network consisting of sensors, actuators, and interaction agents at the back-end. The DIOS interactive object framework enables high-level definition and deployment of sensors and actuators into existing application objects. DIOS can handle both distributed and dynamic objects. The control network interconnecting these sensors and actuators is hierarchical to ensure scalability to large parallel and distributed systems. The interaction gateway

provides an java-enabled interaction "proxy" to the application and provides web-based access to the application. An experimental evaluation of the DIOS framework was 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 local network interaction/collaboration servers to a widely distributed network. This will enable applications connected to local servers to be globally accessible. Clients connected to a server can access applications connected locally as well as remotely. As the servers are typically interconnected through a more reliable and higher 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. 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. Finally, we exploring integration of the DISCOVER portals with other portal efforts such as the Globus CoG [9] and the Grid Portal Collaboration [34].

REFERENCES

1. B. Plale, G. Eisenhauer, K. Schwan, J. Heiner, V. Martin, and J. Vetter. "From Interactive Applications to Distributed Laboratories," IEEE Concurrency, IEEE Computer Society Press, pp. 78 - 90. April-June 1998.
2. J. Vetter and K. Schwan. "High Performance Computational Steering of Physical Simulations," International Parallel Processing Symposium (IPPS), IEEE, Geneva, April 1997.
3. "Java Native Interface Specification," http://web2.java.sun.com/products/jdk/1.1/docs/guide/jni.
4. W. Gu, J. Vetter, and K. Schwan. "Computational steering annotated bibliography," Sigplan notices, 32 (6): 40-4 (June 1997).
5. J. Mulder, J. van Wijk and R. van Liere. "A Survey for Computational Steering Environments," Future Generation Computer Systems, Vol. 15, nr. 2, 1999.
6. S.G. Parker, C.R. Johnson. "SCIRun: A scientific Programming Environment for computational steering," In Proceedings of Supercomputing '95, 1995.
7. R.L. Ribler, J.S. Vetter, H. Simitci, and D.A. Reed. "Autopilot: Adaptive Control of Distributed Applications," Proceedings of the 7th IEEE Symposium on High-Performance Distributed Computing, Chicago, IL, July 1998.
8. G. von Laszewski, I. Foster, J. Gawor, W. Smith and S. 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.
9. I. Foster and C. Kesselman. "Globus: A Metacomputing Infrastructure Toolkit," International Journal of Supercomputer Applications, 11(2): 115-128, 1997.
10. E. Arkarsu, G. Fox, W. Furmanski, T. Haupt, H. Ozdemir, and Z. Odcikin. "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. B. Asbury, G. Fox, T. Haupt, K. Flurchick. "The Gateway Project: An Interoperable Problem Solving Environments Framework for High Performance Computing," http://www.osc.edu/ kenf/theGateway.
12. 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.
13. J. Vetter and K. Schwan. "Models for Computational Steering," Third International Conference on Configurable Distributed Systems, IEEE, May 1996.

14. R. van Liere, J. Harkes and W. 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. G. Eisenhauer and K. Schwan. “An Object-Based Infrastructure for Program Monitoring and Steering,” 2nd SIGMETRICS Symposium on Parallel and Distributed Tools (1998).
16. G. Eisenhauer. “An Object Infrastructure for High-Performance Interactive Applications,” PhD thesis, Department of Computer Science, Georgia Institute of Technology, May 1998
17. G. Eisenhauer and K. Schwan. “Mirror Object Steering System,” http://www.cc.gatech.edu/systems/projects/MOSS.
18. L.K. Jain. “A Distributed, Component-Based Solution for Scientific Information Management,” MS Report, Oregon State University (1998).
19. C. Bajaj and S. Cutchin. “Web based Collaborative Visualization of Distributed and Parallel Simulation,” IEEE Parallel Symposium on Visualization (1999).
20. B. Driggers, J. Alameda and K. Bishop. “Distributed Collaboration for Engineering and Scientific Applications Implemented in Habanero, a Java-Based Environment,” http://union.ncsa.uiuc.edu/habenaro.
21. G. Fox et al. “Tango - A Collaborative Environment for the World Wide Web,” Technical Report, NPAC Syracuse University, Syracuse NY.
22. R.R. Raje, A. Teal, J Coulson, S. Yao, W. Winn and E. Guy III. “CCASEE - A Collaborative Computer Assisted Software Engineering Environment,” Proceedings of the International Association of Science and Technology for Development (IASTED) Conference (1997).
23. M. Boyles, R.R. Raje and S. Fang. “CEV: Collaboration Environment for Visualization Using Java RMI,” Proceedings of the ACM Workshop on Java for High-Performance Network Computing (1998).
24. J. Hunter. “Java Servlet Programming,” 1st edition, O’Reilly, California (1998).
25. “CORBA: Common Object Request Broker Architecture,” http://www.omg.org.
26. “Java Remote Method Invocation,” http://java.sun.com/products/jdk/rmi.
27. J. Buck, S. Ha, E.A. Lee and D.G. Messerschmitt, “Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems,” International Journal of Computer Simulation, 1992.
28. J.A. Wheeler et al. “IPARS: Integrated Parallel Reservoir Simulator,” Center for Subsurface Modeling, University of Texas at Austin, http://www.ticam.utexas.edu/CSM.
29. MPI Forum. “MPI: Message Passing Interface,” www.mcs.anl.gov/mpi.
30. R. Muralidhar and M. Parashar. “A Distributed Object Framework for Web-based Interaction and Steering of Distributed Applications,” www.caip.rutgers.edu/TASSL/Projects/DISCOVER.
31. “PHP Hyper Processor,” http://www.php.net
32. “MySQL,” http://www.mysql.com
33. ”Myproxy (v 1.0),” National Laboratory for Applied Network Research, http://dast.nlanr.net/Features/MyProxy/, July 2000.
34. “Grid Portal Collaboration,” https://palomar.extreme.indiana.edu/.