An Architecture for Web-based Interaction and Steering of Adaptive Parallel/Distributed Applications

Rajeev Muralidhar, Samian Kaur 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: {rajeevdm,samian,parashar}@caip.rutgers.edu

Abstract. This paper presents an architecture for web-based interaction and steering of parallel/distributed scientific applications. The architecture is composed of detachable thin-clients at the front-end, a network of web 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 conventional browser. The application control network enables sensors and actuators to be encapsulated within, and directly deployed with the computational objects. Interaction agents resident at each computational node register the interaction objects and export their interaction interfaces. An application interaction gateway manages the overall interaction through the control network of interaction agents and objects. It uses Java proxy objects that mirror computational objects to enable them to be directly accessed by the interaction web-server. The presented architecture is part of an ongoing effort to develop and deploy a web-based computational collaboratory that enables geographically distributed scientists and engineers to collaboratively monitor and control distributed applications.

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 the scientists and engineers to be able to monitor the progress of these simulations, and to control or steer them at runtime. The utility and cost-effectiveness of these simulations can be greatly increased by transforming the traditional batch simulations into more interactive ones. Closing the loop between the user and the simulations enables the experts to drive the discovery process by observing intermediate results, by changing parameters to lead the simulation to more interesting domains, play what-if games, detect and correct unstable situations, and terminate uninteresting runs early. Furthermore, the increased complexity and multi-disciplinary nature of these simulations necessitates a collaborative effort among multiple, usually geographically distributed scientists/engineers. As a result, collaboration-enabling tools have become critical for transforming simulations into true research modalities.

Enabling seamless interaction and steering high-performance parallel/distributed applications presents many challenges. A key issue is the definition and deployment of *interaction objects* with *sensors* and *actuators* [16] 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 adaptive applications (e.g. simulations on adaptive meshes) where computational objects can be created, deleted, modified and redistributed on the fly. Another issue is the deployment of a *control network* that interconnects these sensor and actuators so that commands and requests can be routed to the appropriate set of computational objects, and information returned can be collated and coherently presented. Finally, the interaction and steering interfaces presented by the application need to be exported so that they can be easily accessed by a group of collaborating users to monitor, analyze, and control the application.

The objective of this paper is to present the design of a web-based collaborative interaction and steering environment that addresses each of these issues. The system supports a 3-tier architecture composed of detachable thin-clients at the front-end, a network of Java interaction servers in the middle, and a control network of sensors, actuators, and interaction agents superimposed on the application data-network at the back-end. The interaction web server enables clients to connect to and collaboratively interact with registered applications using a conventional browser. Furthermore, it provides seamless access to computational and visualization servers, and simulation archives. The application control network enables sensor and actuators to be encapsulated within, and directly deployed with the computational objects. Interaction agents resident at each computational node register the interaction objects and export their interaction interfaces. These agents coordinate interactions with distributed and dynamic computational objects. The application interaction proxy manages the overall interaction through the control network of interaction agents and objects. It uses JNI [2] to create Java proxy objects that mirror the computational objects and allow them to be directly accessed by the interaction web-server. The presented research is part of DISCOVER[1], an ongoing research initiative aimed at developing a web-based interactive computational collaboratory. The current implementation enables geographically distributed clients to use the web to simultaneously connect, monitor and steer multiple applications in a collaborative fashion through a set of dedicated interaction Web 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 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 presents conclusions and current and future work.

---

[1] *D*istributed *I*nteractive *S*teering and *Co*llaborative *V*isualization *E*nvi*R*onment (www.caip.rutgers.edu/TASSL/Projects/DISCOVER/)

2 Related Work

Interactive systems for application run-time steering and control can be classified as follows: (1) *Event based steering systems* - These systems are oriented towards the processing of "events" that occur when pieces of code are executed. Instrumentation code is placed in the application statically at compile time at such instrumentation points. When the corresponding events occur, steering actions and decisions are taken. Systems that fall under this category include Progress [18], Magellan [16] and the CSE [5]. (2) *Systems with high-level abstractions* – In order to overcome the shortcomings of event-based steering, systems such as the *Mirror Object Steering System* (MOSS) [4] provide higher-level abstractions for steering by translating application level data structures and parameters into CORBA [13] - style objects. The DISCOVER system presented in this paper falls under this category. Key contributions of the DISCOVER architecture include (a) support for distributed dynamic interactive objects that can span multiple address spaces and can be dynamically created and destroyed, (b) a scalable control network and (c) support for web-based interaction and steering portals.

Other interactive systems include systems for interactive program construction (e.g. SCIRun [6]), systems for interactive performance optimizations (e.g. Autopilot [15]), systems for interactive application configuration and deployment (e.g. WebFlow [17], Gateway [3]). In addition to these systems, there are numerous systems that provide web-based collaborative visualization environments like *DOVE* [7], the *Web Based Collaborative Visualization* [8] system, the *NCSA Habenaro* [9] system, *Tango* [10], *CCASE* [11] and *CEV* [12].

3 DISCOVER: An Interactive Computational Collaboratory

Fig. 1 presents an architectural overview of the DISCOVER collaboratory aimed at enabling web-based interaction and steering of high-performance parallel/distributed applications. The system has 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 data-network at the back-end. The front-end consists of a range of web-based client portals supporting palmtops connected through a wireless link as well high-end desktops with high-speed links. Clients can connect to a server at any time using a browser to receive information about active applications. Furthermore, they can form or join collaboration groups and can (collaboratively) interact with one or more applications based on their capabilities. The client interaction portal supports two desktops – a local desktop represents the clients' private view, while a virtual desktop presents a metaphor for the global virtual space to enable multiple users to collaborate with the aid of tools like whiteboards and chats. Application views (e.g. a plot) can be made collaborative by creating them in the virtual desktop or transferring them from the local to the virtual desktop. Session management and concurrency control is based on capabilities granted by the server. A simple locking mechanism is used to ensure that the application remains in a consistent state. DISCOVER is currently being used to provide interaction capabilities to a number of scientific and engineering

applications[2] including oil reservoir simulations, computational fluid dynamics and numerical relativity.

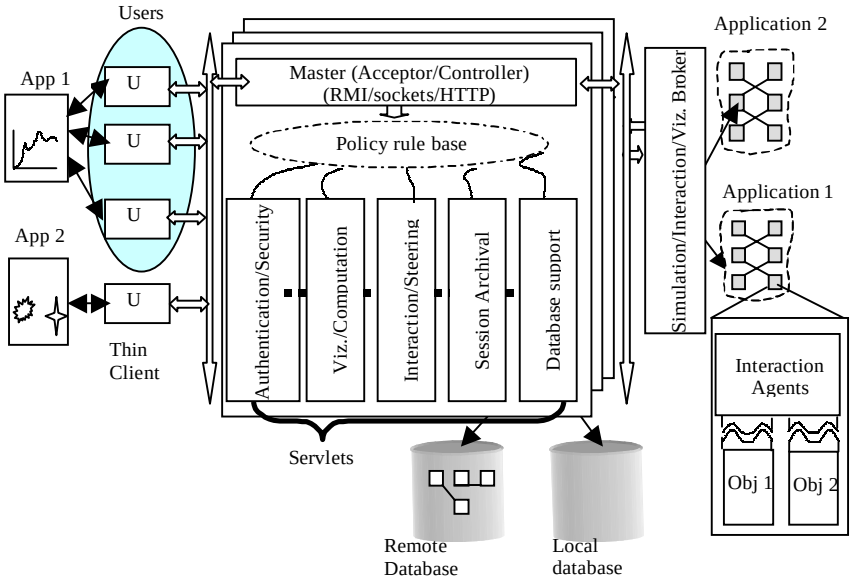

Fig. 1. Architectural Schematic of the DISCOVER Interactive Computational Collaboratory

4 Interaction and Collaboration Servers

The middle tier of the DISCOVER system consists of a network of interaction and collaboration servers aimed at providing a web-based portal to executing high-performance applications. The servers build on *Servlet* [1] technologies to add a range of specialized capabilities to traditional web-servers. A key innovation of the server architecture is the use of *reflection* to provide an extensible set of services that can be dynamically invoked in an application specific manner. The overall architecture consists of a master acceptor/controller servlet and a suite of service handler servlets including interaction and steering handler, collaboration handler, security/authentication handler, visualization handler and session archival handler and database handler. The overall architecture of the interaction server is shown in Fig.1 and the key services are described below.

[2] For example, see www.caip.rutgers.edu/TASSL/DISCOVER/ipars.html.

4.1 Collaborative Interaction and Steering

Collaborative interaction is managed by the master servlet along with the interaction and collaboration handler servlets. The master is the central hub for all communication between the client and a server. Furthermore, it coordinates all registered applications and interaction sessions. Each application, on connection, registers with this servlet, which in turn spawns a dedicated *application interaction broker*. All client requests are classified and forwarded to the corresponding broker by the interaction handler. The broker uses an application proxy to locate application objects, discover object interaction/analysis interfaces, forward queries and commands, and aggregate information from distributed objects. Application responses and updates are multicast to the interested client group by the collaboration handler. Clients can form/join/leave collaboration groups at any time. On a client connection, the master provides the incoming client with a list of registered applications. The master also accepts, parses and validates client requests and redirects them to the relevant utility servlet.

4.2 Security, Authentication and Access Control

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. 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 enable access control all applications are required 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). Each interaction request is then validated against the ACL before it is processed.

4.3 Application View 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 can be downloaded from the server at runtime and 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 and associated plug-ins.

5 Application Control Network for Interaction and Steering

The DISCOVER control network is composed of two components: (1) Interaction Objects that encapsulate interaction sensors and actuators, and (2) a Control Network consisting of distributed Interaction Objects and Interaction Agents. These components are described below.

5.1 Sensors/Actuators and Interaction Object

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 the computational objects are distributed and dynamic. In the DISCOVER system, this is achieved by deriving the computational objects from a virtual interaction base class of the DISCOVER Distributed Interaction Object library. The derived objects define a set of *View*s that they can provide and a set of *Commands* that they can accept. Interaction agents then export these views and commands to the interaction server using a simple *Interaction IDL* (Interface Definition Language). Interaction objects can be either local to a single computational node, distributed across multiple nodes, or shared between some or all of the nodes. Distributed objects have an additional *distribution* attribute that describes their layouts. 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.

In the case of applications written in non-object-oriented languages such as Fortran, application data structures are first converted into computation objects using a C++ wrapper object. These objects are then transformed to interaction objects as described above.

5.2 The Control Network and Interaction Agents

The DISCOVER control network (see Fig. 2) has a hierarchical *cellular* structure and partitions the processing nodes into a number *interaction cells*. The network is composed of (1) Discover Agents on each node, (2) Base Stations on each interaction cell and (3) an Interaction Gateway that connects to the interaction server and provides a proxy to the entire application. The number of nodes per interaction cell is programmable. The cellular control network is automatically configured at run-time using an underlying messaging environment and the available number of processors.

Discover Agents present on each node maintain run-time references to all registered interaction objects on that node. The object references can change dynamically during program execution if data is migrated to handle load balancing. The control network ensures that object references are valid and refer to consistent data. At startup, each Discover Agent exports the registered objects' interaction information to its corresponding Base Station. Base Stations maintain a Cell Object

Registry containing information about interaction objects for its interaction cell. Similarly, the Interaction Gateway maintains a Central Object Registry of interaction objects exported by all Base Stations. It is responsible for interfacing with the interaction server, delegating interaction requests to the appropriate interaction agents (Discover Agents and/or Base Stations), and collecting their responses. In the case of distributed objects, the Gateway additionally performs a *gather* operation for collating the responses arriving from the corresponding nodes.

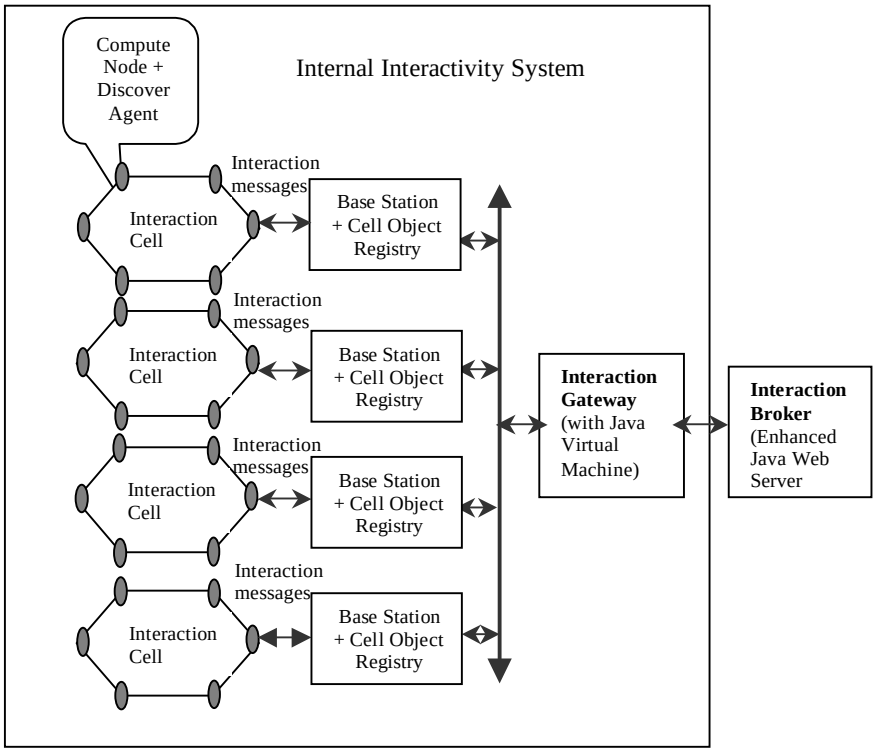

Fig. 2. DISCOVER Control Network for Application Interaction and Steering

The Interaction Gateway creates a Java mirror of each registered interaction object interface using the Java Native Interface (JNI) [2]. It uses the proxy object pattern [14] to create a placeholder for the remote (possible distributed) interaction objects (the subject) and controls access to it. This innovative feature enables interaction objects to be easily exported to an interaction server using the *Serializable* interface of Java to the interaction web server.

6 Conclusion and Future Work

This paper presented the architecture of the DISCOVER web-based computational collaboratory. The current implementation allows thin-clients at the front-end, uses enhanced Java Servlets-based interaction web servers for the middle-tier and a control network of interaction agents and interaction objects at the back-end. The interaction servers enable clients to connect to, and collaboratively interact with registered applications using a conventional browser. Current work includes implementing the Java mirroring capability using JNI at the application's interaction gateway and extending the middle tier to consist of a network of CORBA connected servers.

References

1. Hunter J.: Java Servlet Programming. 1st edition, O'Reilly, California (1998).
2. Gordon R.: Essential JNI: Java Native Interface. 1st edn. Prentice Hall, New Jersey (1998).
3. Asbury B., Fox G., Haupt T., Flurchick K.: The Gateway Project: An Interoperable Problem Solving Environments Framework for High Performance Computing. http://www.osc.edu/~kenf/theGateway.
4. Eisenhauer G., Schwan K: An Object-Based Infrastructure for Program Monitoring and Steering. 2nd SIGMETRICS Symposium on Parallel and Distributed Tools (1998).
5. van Liere R., Harkes J., de Leeuw W: A Distributed Blackboard Architecture for Interactive Data Visualization. IEEE Viz. Conference (1998).
6. Parker S.G., Johnson S.G.: SCIRun: A scientific Programming Environment for computational steering. Proceedings of Supercomputing (1995).
7. Jain L.K.: A Distributed, Component-Based Solution for Scientific Information Management. MS Report, Oregon State University (1998).
8. Bajaj C., Cutchin S.: Web based Collaborative Visualization of Distributed and Parallel Simulation**,** IEEE Parallel Symposium on Visualization (1999).
9. NCSA Habenaro Home Page.: http://havefun.ncsa.uiuc.edu/habenaro. NCSA Software Development Division (1998).
10. Tango Interactive.: http://www.webwisdom.com/tangointeractive.
11. 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).
12. 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).
13. Common Object Request Broker Architecture, http://www.omg.org.
14. Gamma E., Helm R., Johnson R., Vlissides J.: Design Patterns, Addison Wesley Professional Computing Series (1994).
15. Ribler R.R., Vetter J., Simitci H., and Reed D.: AutoPilot: Adaptive Control of Distributed Applications. 7th IEEE Symp. On High Performance Distributed Computing (1998).
16. Vetter J., Schwan K: High Performance Computational Steering of Physical Simulations. IEEE International Parallel Processing Symposium (1997).
17. Arkarsu E., Fox G., Furmanski W., Haupt T., Osdemir H., Oademir Z. O.: Building Web/Commodity base Visual Authoring Environments for Distributed Object/Component Applications – A Case Study Using NPAC WebFlow Systems.
18. Vetter J., Schwan K: Progress: A Toolkit for Interactive Program Steering. Proceedings of the 1995 International Conference on Parallel Processing (1995), 139-149.