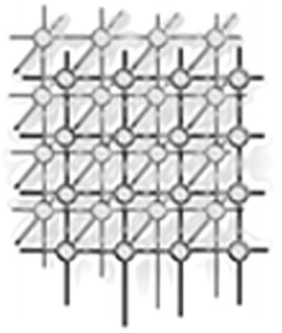
CONCURRENCY—PRACTICE AND EXPERIENCE
Concurrency: Pract. Exper. 2002; **00**:1–15 *Prepared using* cpeauth.cls *[Version: 2001/03/05 v2.01]*



DISCOVER: A Computational Collaboratory for Interactive Grid Applications‡

V. Mann and M. Parashar*, †

The Applied Software Systems Laboratory,
Department of Electrical and Computer Engineering,
Rutgers, The State University of New Jersey,
94 Brett Road, Piscataway, NJ 08854.
Tel: (732) 445-5388 Fax: (732) 445-0593

SUMMARY

The growth of the Internet and the advent of the computational Grid have made it possible to develop and deploy advanced services and computational collaboratories on the Grid. These systems build on high-end computational resources and communication technologies, and enable seamless and collaborative access to resources, applications and data. In this chapter we present an overview of the DISCOVER computational collaboratory for enabling interactive applications on the Grid. Its primary goal is to bring large distributed Grid applications to the scientists'/engineers' desktop and enable collaborative application monitoring, interaction and control. DISCOVER is composed of 3 key components: (1) a middleware substrate that integrates DISCOVER servers and enables interoperability with external Grid services, (2) an application control network consisting of sensors, actuators, and interaction agents that enable monitoring, interaction and steering of distributed applications, and (3) detachable portals for collaborative access to grid applications and services. The design, implementation, operation and evaluation of these components is presented.

KEY WORDS: Computational Collaboratory, Computational Interaction & Steering, Grid Middleware

*Correspondence to: Manish Parashar, Department of Electrical and Computer Engineering, Rutgers, The State University of New Jersey, 94 Brett Road, Piscataway, NJ 08854, USA
†E-mail: {vijay,parashar}@caip.rutgers.edu
‡The DISCOVER collaboratory can be accessed at `http://www.discoverportal.org/`

Contract/grant sponsor: National Science Foundation (CAREERS, NGS, ITR); contract/grant number: ACI9984357, EIA0103674, EIA0120934
Contract/grant sponsor: Department of Energy/California Institute of Technology (ASCI); contract/grant number: PC 295251

Received 07 July 2002

1. INTRODUCTION

A collaboratory is defined as a place where scientists and researchers work together to solve complex interdisciplinary problems, despite geographic and organizational boundaries [18]. The growth of the Internet and the advent of the computational "Grid" [8, 7] have made it possible to develop and deploy advanced computational collaboratories [12, 11] that provide uniform (collaborative) access to computational resources, services, applications and/or data. These systems expand the resources available to researchers, enable multidisciplinary collaborations and problem solving, accelerate the dissemination of knowledge, and increase the efficiency of research.

This chapter presents the design, implementation and deployment of the DISCOVER computational collaboratory that enables interactive applications on the Grid. High performance 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 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.

DISCOVER [6, 17] is a virtual, interactive computational collaboratory that enables geographically distributed scientists and engineers to collaboratively monitor, and control high performance parallel/distributed applications on the Grid. Its primary goal is to bring Grid applications to the scientists'/engineers' desktop, enabling them to collaboratively access, interrogate, interact with and steer these applications using web-based portals. DISCOVER is composed of three key components (see Figure 1):

1. **DISCOVER Middleware Substrate**, that enables global collaborative access to multiple, geographically distributed instances of the DISCOVER computational collaboratory, and provides interoperability between DISCOVER and external Grid services. The middleware substrate enables DISCOVER interaction and collaboration servers to dynamically discover and connect to one another to form a peer network. This allows clients connected to their local servers to have global access to all applications and services across all servers based on their credentials, capabilities and privileges. The DISCOVER middleware substrate and interaction and collaboration servers build on existing web servers and leverage commodity technologies and protocols to enable rapid deployment, ubiquitous and pervasive access, and easy integration with third party services.
2. **DIOS Interactive Object Framework (DIOS)**, that enables the runtime monitoring, interaction and computational steering of parallel and distributed applications on the Grid. DIOS enables application objects to be enhanced with sensors and actuators so that they can be interrogated and controlled. Application objects may be distributed (spanning many processors) and dynamic (be created, deleted, changed or migrated at runtime). A control network connects and manages the

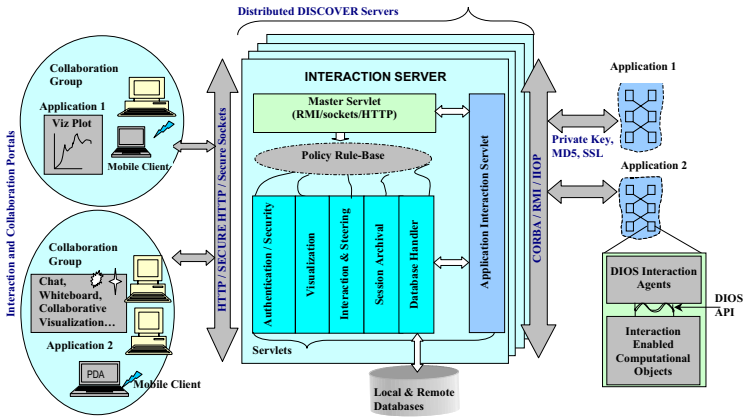


Figure 1. Architectural schematic of the DISCOVER computational collaboratory

distributed sensors and actuators, and enables their external discovery, interrogation, monitoring and manipulation. The DIOS distributed rule engine allows users to remotely define and deploy rules and policies at runtime and enables automated monitoring and steering of Grid applications.

3. **DISCOVER Interaction and Collaboration Portal**, that provides remote, collaborative access to applications, application objects and grid services. The portal provides a replicated shared workspace architecture and integrates collaboration tools such as chat and whiteboard. It also integrates "Collaboration Streams," that maintain a navigable record of all client-client and client-applications interactions and collaboration.

Using the DISCOVER computational collaboratory clients can connect to a local server using the portal, and can use it to discover and access active applications and services on the Grid as long as they have appropriate privileges and capabilities. Furthermore, they can form or join collaboration groups and can securely, consistently and collaboratively interact with and steer applications based on their privileges and capabilities. DISCOVER is currently operational and is being used to provide interaction capabilities to a number of scientific and engineering applications, including oil reservoir simulations, computational fluid dynamics, seismic modeling, and numerical relativity. Furthermore, the DISCOVER middleware substrate provides interoperability between DISCOVER interaction and collaboration services and Globus [9] grid services. The current DISCOVER server network includes deployments at CSM, University of Texas at Austin, and is being expanded to include CACR, California Institute of Technology.

The rest of the chapter is organized as follows. Section 2 presents the DISCOVER middleware substrate. Section 3 describes the DIOS interactive object framework. Section 4 presents the experimental evaluation. Section 5 describes the DISCOVER collaborative portal. Section 6 presents a summary of chapter and the current status of DISCOVER.

2. The DISCOVER Middleware Substrate for Grid-Based Collaboratories

The proliferation of the computational Grid and recent advances in Grid technologies have enabled the development and deployment of a number of advanced problem solving environments and computational collaboratories. These systems provide specialized services to their user communities and/or address specific issues in wide area resource sharing and Grid computing [10]. However, solving real problems on the Grid requires combining these services in a seamless manner. For example, execution of an application on the Grid requires security services to authenticate users and the application, information services for resource discovery, resource management services for resource allocation, data transfer services for staging, and scheduling services for application execution. Once the application is executing on the Grid, interaction, steering, and collaboration services allow geographically distributed users to collectively monitor and control the application allowing the application to be a true research or instructional modality. Once the application terminates data storage and clean up services come into play. Clearly, a seamless integration and interoperability of these services is critical to enable global, collaborative, multi-disciplinary and multi-institutional, problem solving.

Integrating these collaboratories and Grid services presents significant challenges. The collaboratories have evolved in parallel with the Grid computing effort and have been developed to meet unique requirements and support specific user communities. As a result, these systems have customized architectures and implementations, and build on specialized enabling technologies. Furthermore, there are organizational constraints that may prevent such interaction as it involves modifying existing software. A key challenge then, is the design and development of a robust and scalable middleware that addresses interoperability, and provides essential enabling services such as security and access control, discovery, and interaction and collaboration management. Such a middleware should provide loose coupling among systems to accommodate organizational constraints and an option to join or leave this interaction at any time. It should define a minimal set of interfaces and protocols to enable collaboratories to share resources, services, data and applications on the Grid while being able to maintain their architectures and implementations of choice.

The DISCOVER middleware substrate [20, 21] defines interfaces and mechanisms for a peer-to-peer integration and interoperability of services provided by domain specific collaboratories on the Grid. It currently enables interoperability between geographically distributed instances of the DISCOVER collaboratory. Furthermore, it also integrates DISCOVER collaboratory services with the Grid services provided by the Globus Toolkit [9] using the CORBA Commodity Grid (CORBA CoG) Kit [25, 27]. Clients can now use the services provided by the CORBA CoG Kit to discover available resources on the Grid, to allocate required resources and to run applications on these resources, and use DISCOVER to connect to and collaboratively monitor, interact with, and steer the applications. The middleware substrate enables DISCOVER interaction and steering servers as well as Globus servers to dynamically discover and connect to one another to form a peer network. This allows clients connected to their local servers to have global access to all applications and services across all the servers in the network based on their credentials, capabilities and privileges.

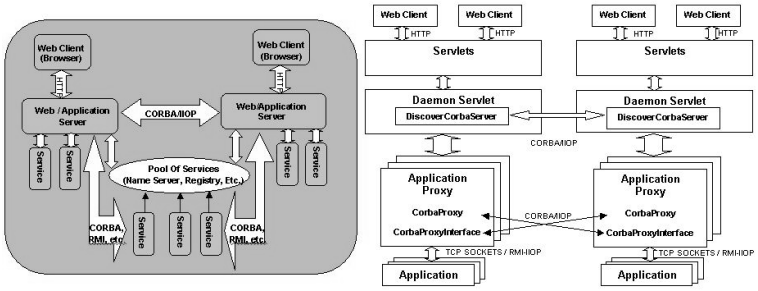


Figure 2. DISCOVER Middleware Substrate: (a) Architecture (b) Implementation

2.1. DISCOVER Middleware Substrate Design

The DISCOVER middleware substrate has a hybrid architecture, i.e. it provides a client-server architecture from the users' point of view, while the middle tier has a peer-to-peer architecture. This approach provides several advantages. The middle-tier peer-to-peer network distributes services across peer-servers and reduces the requirements of a server. As clients connect to the middle-tier using the client-server approach, the number of peers in the system is significantly smaller than a pure peer-to-peer system. The smaller number of peers allows the hybrid architecture to be more secure and better managed as compared to a true peer-to-peer system, and restricts the security and manageability concerns to the middle tier. Furthermore, this approach makes no assumptions about the capabilities of the clients or the bandwidth available to them, and allows for very thin clients. Finally, servers in this model can be lightweight, portable and easily deployable and manageable, instead of being heavy weight (as in pure client-server systems). A server may be deployed anywhere there is a growing community of users, much like a HTTP Proxy server.

A schematic overview of the overall architecture is presented in Figure 2a. It consists of (collaborative) client portals at the front end, computational resources, services or applications at the backend, and the network of peer servers in the middle. To enable ubiquitous access, clients are kept as simple as possible. The responsibilities of the middleware include providing a "repository of services" view to the client, providing controlled access to these backend services, interacting with peer servers, and collectively managing and coordinating collaboration. A client connects to its "closest" server and should have access to all (local and remote) backend services and applications defined by its privileges and capabilities.

Backend services can divided into two main classes - (a) resource access and management toolkits (e.g. Globus, CORBA CoG) providing access to Grid services, and (b) collaboratory specific services (e.g high-performance applications, data archives, and network-monitoring tools). Services may be specific to a server or may form a pool of services that can be accessed by any server. A service will be server-specific if direct access to the service is restricted to the local server, possibly due to security, scalability or compatibility constraints. In either case, the servers and the backend services are accessed using standard distributed object technologies such as CORBA/IIOP [2, 4] and RMI [16].

XML based protocols such as SOAP [26] have been designed considering the services model and are ideal candidates.

The middleware architecture defines three levels of interfaces for each server in the substrate. The level-one interfaces enable a server to authenticate with peer servers and query them for active services and users. The level-two interfaces are used for authenticating with and accessing a specific service at a server. The level-three interfaces (Grid Infrastructure Interfaces) are used for communicating with underlying core Grid services (e.g. security, resource access). The implementation and operation of the current DISCOVER middleware substrate is briefly described below. Details can be found in [20, 1].

2.2. DISCOVER Middleware Substrate Implementation

2.2.1. DISCOVER Interaction and Collaboration Server

The DISCOVER interaction/collaboration servers build on commodity web servers, and extend their functionality (using Java Servlets [13]) to provide specialized services for real-time application interaction and steering and for collaboration between client groups. Clients are Java applets and communicate with the server over HTTP using a series of HTTP *GET* and *POST* requests. Application-to-server communication either uses standard distributed object protocols such as CORBA [2] and Java RMI [16], or a more optimized, custom protocol over TCP sockets. An *ApplicationProxy* object is created for each active application/service at the server, and is given a unique identifier. This object encapsulates the entire context for the application. Three communication channels are established between a server and an application: (1) a *MainChannel* for application registration and periodic updates, (2) a *CommandChannel* for forwarding client interaction requests to the application, and (3) a *ResponseChannel* for communicating application responses to interaction requests. At the other end, clients differentiate between the various messages (i.e. Response, Error or Update) using Java's reflection mechanism. Core service handlers provided by each server include the Master Handler, Collaboration Handler, Command Handler, Security/Authentication Handler and the Daemon Servlet that listens for application connections. Details about the design and implementation of the DISCOVER Interaction and Collaboration servers can be found in [17].

2.2.2. DISCOVER Middleware Substrate

The current implementation of the DISCOVER middleware consists of multiple independent collaboratory domains, each consisting of one or more DISCOVER servers, applications/services connected to the server(s) and/or core Grid services. The middleware substrate builds on CORBA/IIOP, which provides peer-to-peer connectivity between servers within and across domains, while allowing them to maintain their individual architectures and implementations. The implementation is illustrated in Figure 2b. It uses the level-one and level-two interfaces to construct a network of DISCOVER servers. A third level of interfaces is used to integrate Globus Grid Services [9] via the CORBA CoG [25, 27]. The different interfaces are described below.

***DiscoverCorbaServer* Interface:** The *DiscoverCorbaServer* is the level-one interface and represents a server in the system. This interface is implemented by each server and defines the methods for interacting with a server. This includes methods for authenticating with the server, querying the server for active applications/services, and obtaining the list of users logged on to the server.

A *DiscoverCorbaServer* object is maintained by each server's Daemon Servlet and publishes its availability using the CORBA trader service. It also maintains a table of references to *CorbaProxy* objects for remote applications/services.

***CorbaProxy* Interface:** The *CorbaProxy* interface is the level-two interface and represents an active application (or service) at a server. This interface defines methods for accessing and interacting with the application/service. The *CorbaProxy* object also binds itself to the CORBA naming service using the application's unique identifier as the name. This allows the application/service to be discovered and remotely accessed from any server. The *DiscoverCorbaServer* objects at servers that have clients interacting with a remote application maintain a reference to the application's *CorbaProxy* object.

Grid Infrastructure Interfaces: The level-three interfaces represent core Globus Grid Services. These include: (1) The *DiscoverGSI* interface that enables the creation and delegation of secure proxy objects using the Globus GSI Grid security service. (2) The *DiscoverMDS* that provides access to the Globus MDS Grid information service using JNDI [14], and enables users to securely connect to and access MDS servers. (3) The *DiscoverGRAM* interface that provides access to the Globus GRAM Grid resource management service and allows users to submit jobs on remote hosts, and to monitor and manage these jobs using the CORBA Event Service [3]. (4) The *DiscoverGASS* interface that provides access to the Globus GASS Grid data access service and enables Grid applications to access and store remote data.

2.3. DISCOVER Middleware Operation

This section briefly describes key operations of the DISCOVER middleware. Details can be found in [20, 1].

2.3.1. Security/Authentication

The DISCOVER security model is based on the Globus GSI protocol and builds on the CORBA Security Service. The GSI delegation model is used to create and delegate an intermediary object (the CORBA GSI Server Object) between the client and the service. The process consists of three steps: (1) Client and server objects mutually authenticate using the CORBA Security Service. (2) The client delegates the *DiscoverGSI* server object to create a proxy object that has the authority to communicate with other GSI enabled Grid Services. (3) The client can use this secure proxy object to invoke secure connections to the services.

Each DISCOVER server supports a two-level access control for the collaboratory services: the first level manages access to the server while the second level manages access to a particular application. Applications are required to be registered with a server and to provide a list of users and their access privileges (e.g. read-only, read-write). This information is used to create access control lists (ACL) for each user-application pair.

2.3.2. Discovery of Servers, Applications and Resources

Peer DISCOVER servers locate each other using the CORBA trader services [5]. The CORBA trader service maintains server references as *service-offer* pairs. All DISCOVER servers are identified by the service-id *DISCOVER*. The service offer contains the CORBA object reference and a list of properties

defined as name-value pairs. Thus the object can be identified based on the service it provides or its properties. Applications are located using their globally unique identifiers, which are dynamically assigned by the DISCOVER server and are a combination of the server's IP address and a local count at the server. Resources are discovered using the Globus MDS Grid information service, which is accessed via the *MDSHandler* Servlet and the *DiscoverMDS* interface.

2.3.3. Accessing Globus Grid Services: Job Submission and Remote Data Access

DISCOVER middleware allows users to launch applications on remote resources using the Globus GRAM service. The clients invoke the *GRAMHandler* Servlet in order to submit a job. The *GRAMHandler* Servlet, using the delegated CORBA GSI Server Object, accesses the *DiscoverGRAM* server object to submit jobs to the Globus gatekeeper. The user can monitor jobs using the CORBA Event Service. Similarly, clients can store and access remote data using the Globus GASS service. The *GASSHandler* Servlet, using the delegated CORBA GSI Server Object, accesses the *DiscoverGASS* server object and the corresponding GASS service using the protocol specified by the client.

2.3.4. Distributed Collaboration

The DISCOVER collaboratory enables multiple clients to collaboratively interact with and steer (local and remote) applications. The *collaboration handler* servlet at each server handles the collaboration on the server side, while a dedicated polling thread is used on the client side. All clients connected to an application instance form a collaboration group by default. However, as clients can connect to an application through remote servers, collaboration groups can span multiple servers. In this case, the *CorbaProxy* objects at the servers poll each other for updates and responses.

The peer-to-peer architecture offers two significant advantages for collaboration. First, it reduces the network traffic generated. This is because, instead of sending individual collaboration messages to all the clients connected through a remote server, only one message is sent to that remote server, which then updates its locally connected clients. Since clients always interact through the server closest to them and the broadcast messages for collaboration are generated at this server, these messages don't have to travel large distances across the network. This reduces overall network traffic as well as client latencies, especially when the servers are geographically far away. It also leads to better scalability in terms of the number of clients that can participate in a collaboration session without overloading a server, as the session load now spans multiple servers.

2.3.5. Distributed Locking and Logging for Interactive Steering and Collaboration

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. In the distributed server case, locking information is only maintained at the application's host server i.e. the server to which the application connects directly.

The session archival handler maintains two types of logs. The first log maintains all interactions between a client and an application. For remote applications, the client logs are maintained at the server where the clients are connected. The second log maintains all requests, responses, and status

messages for each application throughout its execution. This log is maintained at the application's host server (the server to which the application is directly connected).

2.4. DISCOVER Middleware Substrate Experimental Evaluation

This section gives a brief summary of the experimental evaluation of the DISCOVER middleware substrate. A more detailed description is presented in [20, 1].

2.4.1. Evaluation of DISCOVER Collaboratory Services

This evaluation compared latencies for indirect (remote) accesses and direct accesses to DISCOVER services over a local area network (LAN) and a wide area network (WAN). The first set of measurements was for a 10Mbps local area network (LAN) and used DISCOVER servers at Rutgers University in New Jersey. The second set of measurements was for a wide area network (WAN) and used DISCOVER servers at Rutgers University and at University of Texas at Austin. The clients were running on the local area network at Rutgers University for both sets of measurements and requested data of different sizes from the application. Response times were measured for both, a direct access to the server where the application was connected and an indirect (remote) access through the middleware substrate. The time taken by the application to compute the response was not included in the measured time. Indirect (remote) access time included the direct access time plus the time taken by the server to forward the request to the remote server and to receive the result back from the remote server over IIOP. An average response time over 10 measurements was calculated for each response size.

The resulting response latencies for direct and indirect accesses measured on the LAN indicated that it is more efficient to directly access an application when it is on the same LAN. In contrast to the results for the LAN experiment, indirect access times measured on the WAN were of comparable order to direct access times. In fact, for small data sizes (1 KB, 10 KB and 20 KB) indirect access times were either equal to or smaller than direct access times. While these results might appear to be contradictory to expectations, the underlying communication for the two accesses provides an explanation. In the direct access measurement, the client was running at Rutgers and accessing the server at Austin over HTTP. Thus in the direct access case, a large network path across the Internet was covered over HTTP, which meant that a new TCP connection was set up over the wide area network for every request. In the indirect access case however, the client at Rutgers accessed the local server at Rutgers over HTTP, which in turn accessed the server at Austin over IIOP. Thus, the path covered over HTTP was short and within the same LAN, while the larger network path (across the Internet) was covered over IIOP, which uses the same TCP connection for multiple requests. Since the time taken to set up a new TCP connection for every request over a wide area network is considerably larger than that over a local area network, the direct access times are significantly larger. As data sizes increase, the overhead of connection set up time becomes a relatively smaller portion of the overall communication time involved. As a result the overall access latency is dominated by the communication time, which is larger for remote accesses involving accesses to two servers. In both the cases, the access latency was less than a second.

2.4.2. Evaluation of DISCOVER Grid Services

This experiment evaluates access to Grid services using the DISCOVER middleware substrate. The setup consisted of two DISCOVER server running on grid1.rutgers.edu and tassl-pc-2.rutgers.edu, connected via a 10Mbps LAN. The Globus Toolkit was installed on grid1.rutgers.edu. The test scenario consisted of: (1) the client logging on to the Portal, (2) the client using the *DiscoverMDS* service to locate an appropriate resource, (3) the client using the *DiscoverGRAM* service to launch an application on the remote resource, (4) the client using the *DiscoverGASS* to transfer the output and error files produced by the application, (5) the client interacting and steering the application using the collaboratory services and (6) the client terminating the application using the *DiscoverGRAM* service. The number of clients was varied up to a maximum of 25. The *DiscoverMDS* access time averaged around 250ms. The total time for finding a resource also depends on the search criterion. We restricted our search criteria to memory size and available memory. The *DiscoverGASS* service was used to transfer files of various sizes. *DiscoverGASS* service performed well for small file sizes (below 10 MB) and deteriorated for larger files. The total time taken for the entire test scenario was measured for two cases: (1) the services were accessed locally at grid1.rutgers.edu and (2) the server at tassl-pc-2.rutgers accessed the grid services provided by grid1.rutgers.edu. This time was further divided into 5 distinct time intervals - a) time taken for resolving services, b) time taken for delegation, c) time taken for event channel creation to receive job updates, d) time taken for unbinding the job and e) time taken for transferring the error file. The time taken was approximately 14.5 seconds in the first case and approximately 18 seconds in the second case. The additional time in the second case was spent in resolving the services not present locally.

3. DIOS: Distributed Interactive Object Substrate

DIOS is a distributed object infrastructure that enables the development and deployment of interactive application. It addresses three key challenges: (1) definition and deployment of interaction objects that extend distributed and dynamic computational objects with sensors and actuators for interaction and steering, (2) definition of a scalable control network that interconnects interaction objects and enables object discovery, interrogation and control and (3) definition of an interaction gateway that enables remote clients to access, monitor and interact with applications. The design, implementation and evaluation of DIOS is presented below. DIOS is composed of 2 key components: (1) interaction objects that encapsulate sensors and actuators, and (2) a hierarchical control network composed of *Discover Agents*, *Base Stations*, and an *Interaction Gateway*.

3.1. Sensors, Actuators and Interaction Objects

Interaction objects extend application computational objects with interaction and steering capabilities, through embedded sensors and actuators. Computational objects are the objects (data-structures,

algorithms) used by the application for its computations[†]. In order to enable application interaction and steering, these objects must export interaction interfaces that enable their state to be externally monitored and changed. Sensors and actuators provide such an interface. Sensors provide an interface for viewing the current state of the object, while actuators provide an interface to process commands to modify the state. Note that the sensors and actuators need to be co-located in memory with the computational objects and have access to their internal state. Transforming computational objects into interaction objects can be a significant challenge. This is especially true when the computational objects are distributed across multiple processors and can be dynamically created, deleted, migrated and redistributed. Multiple sensors and actuators now have to coordinate in order to collectively process interaction requests.

DIOS provides application-level programming abstractions and efficient runtime support to support the definition and deployment of sensors and actuators for distributed and dynamic computational objects. Using our current implementation of DIOS, existing applications can be converted by deriving the computational objects from a DIOS virtual interaction base class. The derived objects can then selectively overload the base class methods to define their interaction interfaces 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 of information that the object can provide. For example, a mesh 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 mesh object may include refine, coarsen, and redistribute. The view and command interfaces may be guarded by access polices that define who can access the interfaces, how they can access them and when they can access them. This process requires minimal modification to original computational objects. Discover agents, which are a part of the DIOS control network, combine the individual interfaces and export them to the interaction server using an Interaction IDL (Interface Definition Language). The Interaction IDL contains metadata for interface discovery and access and is compatible with standard distributed object interfaces like CORBA[2] and RMI[16]. In the case of applications written in languages such as Fortran, proxy objects are first created for the application data structures that require interaction. These proxy objects are then transformed to interaction objects as described above. DIOS 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.

3.2. Local, Global and Distributed Objects

Interaction objects can be classified based on the address space(s) they span during the course of the computation as *local*, *global*, and *distributed objects*. Local interaction objects are created in a processor's local memory. These objects may 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

[†] Note that computational objects do not refer only to objects in an object-oriented implementation of an application, but also to application data structures and operations on these data-structures implemented in languages such as C and Fortran.

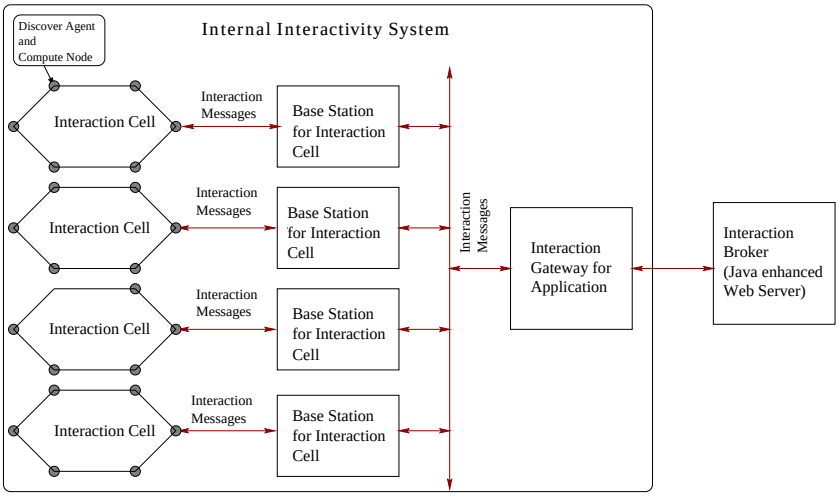


Figure 3. The DIOS Control Network

example is a distributed array partitioned across available computational nodes. These objects contain an additional distribution attribute that maintains its current distribution type (e.g. blocked, staggered, inverse space filling curve-based, or custom) and layout. This attribute can change during the lifetime of the object. Like local and global interaction objects, distributed objects can be dynamically created, deleted, or redistributed.

In order to enable interaction with distributed objects, each distribution type is associated with *gather* and *scatter* operations. Gather aggregates information from the distributed components of the object, while scatter performs the reverse operation. For example, in the case of a distributed array object, the gather operation would collate views generated from sub-blocks of the array while the scatter operator would scatter a query to the relevant sub-blocks. An application can select from a library of gather/scatter methods for popular distribution types provided by DIOS, or can register gather/scatter methods for customized distribution types.

3.3. DIOS Control Network and Interaction Agents

The control network has a hierarchical “cellular” structure with three components - Discover Agents, Base Stations, and Interaction Gateway, as shown in Figure 3. Computational nodes are partitioned into interaction cells, with 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 proxy to the entire application. The control

network is automatically configured at run-time using the underlying messaging environment (e.g. Message Passing Interface (MPI) [22]) and the available number of processors.

3.3.1. Discover Agents, Base Stations and Interaction Gateway

Each computation node in the control network houses a Discover Agent (DA). Each Discover Agent 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 these objects (using the Interaction IDL). Base Stations (BS) form the next level of control network hierarchy. They maintain interaction registries containing the Interaction IDL for all the interaction objects in the interaction cell, and export this information to the Interaction Gateway. The Interaction Gateway (IG) represents an interaction proxy for the entire application. It manages a registry of interaction interfaces for all the interaction objects in the application, and is responsible for interfacing with external interaction servers or brokers. The Interaction Gateway delegates incoming interaction requests to the appropriate Base Stations and Discover Agents, and combines and collates responses. Object migrations and re-distributions are handled by the respective Discover Agents (and Base Stations if the migration/re-distribution is across interaction cells) by updating corresponding registries. The Discover Agent, Base Station and Interaction Gateway are all initialized on the appropriate processors during application start up. They execute in the same address space as the application and communicate using the application messaging environment, e.g. MPI. A recent extension to DIOS allows clients to define and deploy rules to automatically monitor and control applications and/or application objects. The conditions and actions of the rules are composed using the exported view/command interfaces. A distributed rule-engine is built in the control network that authenticates and validates incoming rules, decomposes the rules and distributes components to appropriate application objects, and manages the execution of the rules.

In our implementation, interactions between an interaction server and the interaction gateway are achieved using two approaches. In the first approach, the Interaction Gateway serializes the interaction interfaces and associated meta-data information for all registered interaction objects to the server. A set of Java classes at the server parse the serialized Interaction IDL stream to generate corresponding interaction object proxies. In the second approach, the Interaction Gateway initializes a Java Virtual Machine (JVM) and uses the Java Native Interface [15] to create Java mirrors of registered interaction objects. These mirrors are registered with a RMI [16] registry service executing at the Interaction Gateway. 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. The use of the JVM and JNI in the second approach assumes that the computing environment supports the Java runtime environment.

A more detailed description of the DIOS framework, including examples for converting existing applications into interactive ones, registering them with the DISCOVER interaction server and using web portals for monitoring and controlling them, can be found in [24, 23].

Table I. View and command processing times

View Type	Data Size (Bytes)	Time Taken	Command	Time Taken
Text	65	0.4 ms	Stop, Pause or Resume	250 μsec
Text	120	0.7 ms	Refine GridHierarchy	32 ms
Text	760	0.7 ms	Checkpoint	1.2 sec
XSlice Generation	1024	1.7 ms	Rollback	43 ms

4. Experimental Evaluation

DIOS has been implemented as a C++ library and has been ported to a number of operating systems including Linux, Windows NT, Solaris, IRIX, and AIX. This section summarizes an experimental evaluation of the DIOS library using the IPARS reservoir simulator framework on the Sun Starfire E10000 cluster. The E10000 configuration used consists of 64, 400 MHz SPARC processors, a 12.8 GBytes/sec interconnect. IPARS is a Fortran-based framework for developing parallel/distributed oil reservoir simulators. Using DIOS/DISCOVER, engineers can interactively feed in parameters such as water/gas injection rates and well bottom hole pressure, and observe the water/oil ratio or the oil production rate. The transformation of IPARS using DIOS consisted of creating C++ wrappers around the IPARS well data structures and defining the appropriate interaction interfaces in terms of views and commands. The DIOS evaluation consists of 5 experiments:

Interaction Object Registration: Object registration (generating the Interaction IDL at the Discover Agents and exporting it to Base Station/Gateway) took 500 μsec per object at each Discover Agent, 10 ms per Discover Agent in the interaction cell at the Base Station, and 10 ms per Base Station in the control network at the Gateway. Note that this is a one-time cost.

Overhead of Minimal Steering: This experiment measured the runtime overheads introduced due to DIOS monitoring during application execution. In this experiment, the application automatically updated the DISCOVER server and connected clients with the current state of its interactive objects. Explicit command/view requests were disabled during the experiment. The application contained 5 interaction objects, 2 local objects and 3 global objects. The measurements showed that the overheads due to the DIOS runtime are very small and typically within the error of measurement. In some cases, due to system load dynamics, the performance with DIOS was slightly better. Our observations have shown that for most applications, the DIOS overheads are less that 0.2% of the application computation time.

View/Command Processing Time: The query processing time depends on - (a) the nature of interaction/steering requested, (b) the processing required at the application to satisfy the request and generate a response, and (c) type and size of the response. In this experiment we measured time required for generating and exporting different views and commands. A sampling of the measured times for different scenarios is presented in Table 1.

DIOS Control Network Overheads: This experiment consisted of measuring the overheads due to communication between the Discover Agents, Base Stations and the Interaction Gateway while processing interaction requests for local, global and distributed objects. As expected, the measurements

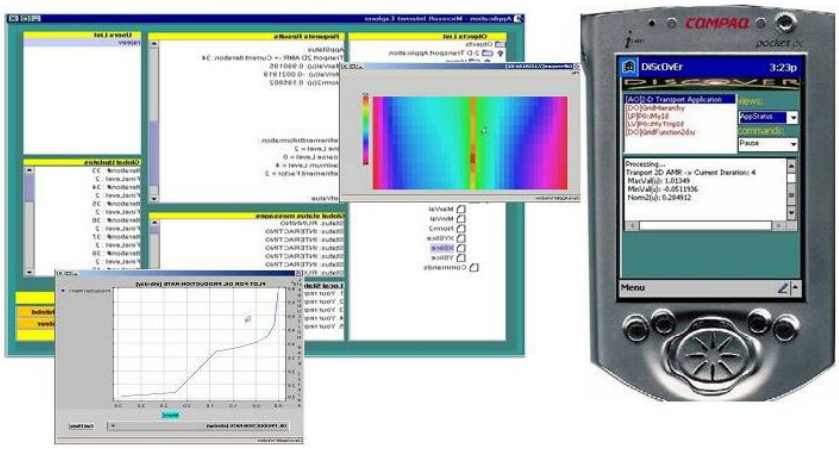


Figure 4. The DISCOVER Collaborative Interaction/Steering Portal

indicated that the interaction request processing time is minimum when the interaction objects are co-located with the Gateway, and is the maximum for distributed objects. This is due to the additional communication between the different Discover Agents and the Gateway, and the `gather` operation performed at the Gateway to collate the responses. Note that for the IPARS application, the average interaction time was within 0.1 to 0.3% of the average time spent in computation during each iteration.
End-to-end steering latency: This measured 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. Remote clients executed within web-browsers on laptops/workstations on different subnets. These measurements of course depend on the state of the client, the server and the network interconnecting them. The DISCOVER system exhibits end-to-end latencies comparable to related steering systems, as reported in [24].

5. The Collaborative Interaction and Steering Portal

The DISCOVER collaborative computational portal provides scientists and engineers with an anytime/anywhere capability of collaboratively (and securely) launching, accessing, monitoring and controlling Grid applications. It integrates access to the collaboratory services and the grid services provided by the DISCOVER middleware. A screen shot of the current DISCOVER portal is presented in Figure 4.

The DISCOVER portal consists of a virtual desktop with local and shared areas. The shared areas implement a replicated shared workspace and enable collaboration among dynamically formed user groups. Locking mechanisms are used to maintain consistency. The base portal, presented to user after authentication and access verification, is a control panel. The control panel provides the user with a

list of services and applications and is customized to match each user's access privileges. Once clients download the control panel they can launch any desired service such as resource discovery, application execution, application interrogation, interaction, collaboration, or application/session archival access. For application access, the desktop 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 once again customized to match the client's access privileges. Chat and whiteboard tools can be launched from the desktop to support collaboration. View requests generate separate (possibly shared) panes using the corresponding view plug-in. All users choosing to steer a particular application form a collaboration group by default with a corresponding shared area on the virtual desktop. New groups can be formed or modified at any time. A separate application registration page is provided to allow super-users to register application, add application users and modify user capabilities.

6. Summary and Current Status

In this chapter we presented an overview of the DISCOVER computational collaboratory for enabling interactive applications on the Grid. Its primary goal is to bring large distributed Grid applications to the scientists'/engineers' desktop and enable collaborative application monitoring, interaction and control. DISCOVER is composed of 3 key components: (1) a middleware substrate that integrates DISCOVER servers and enables interoperability with external Grid services, (2) an application control network consisting of sensors, actuators, and interaction agents that enable monitoring, interaction and steering of distributed applications, and (3) detachable portals for collaborative access to grid applications and services. The design, implementation, operation and evaluation of these components was presented. DISCOVER is currently operational and is being used to provide these capabilities to a number of application specific PSEs including the IPARS oil-reservoir simulator system at the Center for Subsurface Modeling, University of Texas at Austin, the virtual test facility at the ASCI/ASAP Center, California Institute of Technology, and the Astrophysical Simulation Collaboratory. Furthermore, the DISCOVER middleware integrates access to Globus Grid services. Additional information and an online demonstration are available at http://www.discoverportal.org.

Acknowledgments

We would like to thank the members of the DISCOVER team, V. Bhat, M. Dhillon, S. Kaur, H. Liu, V. Matossian, R. Muralidhar, A. Swaminathan and S. Verma for their contributions to this project. We would also like to thank J. Gawor and G. von Laszewski for their help with the CORBA CoG Kit.

REFERENCES

1. V. Bhat, V. Mann and M. Parashar. "Integrating Grid Services using the DISCOVER Middleware Substrate," *Technical Report*, The Applied Software Systems Laboratory, http://www.caip.rutgers.edu/TASSL, June 2002.

2. "CORBA: Common Object Request Broker Architecture", http://www.corba.org.
3. "CORBA's Event Service Version 1.1," http://www.omg.org/technology/documents/formal/event_ service.htm.
4. "CORBA/IIOP Specification", http://www.omg.org/technology/documents/formal/corbaiiop.html.
5. "CORBA Trader Service Specification," ftp://ftp.omg.org/pub/docs/formal/97-07-26.pdf.
6. The DISCOVER Computational Collaboratory, http://www.discoverportal.org.
7. I. Foster, "Internet Computing and the Emerging Grid", Nature, Web Matters, (http://www.nature.com /nature/webmatters/grid/grid.html), Dec. 2000.
8. I. Foster and C. Kesselman. The Grid: Blueprint for a New Computing Infrastructure, *Morgan Kaufmann*, San Francisco, 1998.
9. I. Foster and C. Kesselman. "Globus: A Metacomputing Infrastructure Toolkit," *International Journal of Supercomputing Applications*, 11(2): 115-128, 1997.
10. I. Foster, C. Kesselman, and S. Tuecke. "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", *International Journal of Supercomputing Applications*, 2001.
11. Grid Computing Environments Working Group, Global Grid Forum, http://www.computingportals.org.
12. "The Global Grid Forum, http://www.gridforum.org.
13. J. Hunter. "Java Servlet Programming," 1st edition, O'Reilly, California, 1998.
14. "Java Naming and Directory Interface," http://java.sun.com/products/jndi/libraries.
15. "Java Native Interface Specification," http://web2.java.sun.com/products/jdk/1.1/docs/guide/jni.
16. "Java Remote Method Invocation," http://java.sun.com/products/jdk/rmi.
17. S. Kaur, V. Mann, V. Matossian, R. Muralidhar, and M. Parashar. "Engineering a Distributed Computational Collaboratory", *34th Hawaii Conference on System Sciences*, January 2001
18. R.T. Kouzes, J.D. Myers, and W.A. Wulf. "Collaboratories: Doing science on the Internet", *IEEE Computer*, Vol.29, No.8, August 1996.
19. 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, USA, http://www.mcs.anl.gov/ laszewsk/cog.
20. V. Mann and M. Parashar. "Engineering an Interoperable Computational Collaboratory on the Grid", *Special Issue on Grid Computing Environments, Concurrency and Computation: Practice and Experience*, John Wiley and Sons, 2002 (to appear).
21. V. Mann and M. Parashar. "Middleware Support for Global Access to Integrated Computational Collaboratories", *Proceedings of 10th IEEE International Symposium on High Performance Distributed Computing*, San Francisco, CA, USA, pp 35-46, IEEE Computer Society Press, August 2001.
22. MPI Forum. "MPI: Message Passing Interface," www.mcs.anl.gov/mpi.
23. R. Muralidhar. "A Distributed Object Framework for the Interactive Steering of High-Performance Applications," MS thesis, Department of Electrical and Computer Engineering, Rutgers, The State University of New Jersey, October 2000.
24. R. Muralidhar and M. Parashar. "A Distributed Object Infrastructure for Interaction and Steering," *Proceedings of the 7th International Euro-Par Conference (Euro-Par 2001)*, Lecture Notes in Computer Science, Editors: R. Sakellariou, J. Keane, J. Gurd and L. Freeman, Springer-Verlag, Manchester, UK, Vol. 2150, pp 67 - 74, August 2001.
25. M. Parashar, G. von Laszewski, S. Verma, J. Gawor, K. Keahey, and N. Rehn. "A CORBA Commodity Grid Kit," *Special Issue on Grid Computing Environments, Concurrency and Computation: Practice and Experience*, John Wiley and Sons, 2002 (to appear).
26. Simple Object Access Protocol (SOAP), http://www.w3.org/TR/SOAP.
27. S. Verma, M. Parashar, J. Gawor and G. von Laszewski. "Design and Implementation of a CORBA Commodity Grid Kit", *Second International Workshop on Grid Computing - GRID 2001*, Denver, CO, USA, pp. 2-12, Springer LNCS 2242, November 2001.