Engineering an Interoperable Computational Collaboratory on the Grid[1]

Vijay Mann and Manish Parashar
The Applied Software Systems Laboratory
Department of Electrical and Computer Engineering, Rutgers University
94 Brett Road, Piscataway, NJ 08854
{vijay,parashar}@caip.rutgers.edu

Abstract

The growth of the Internet and the advent of the computational Grid have made it possible to develop and deploy advanced computational collaboratories. These systems build on high-end computational resources, communication technologies, and enabling services underlying the Grid, and provide seamless and collaborative access to resources, applications and data. Combining these focused collaboratories and allowing them to interoperate has many advantages and can lead to truly collaborative, multi-disciplinary and multi-institutional problem solving. However, integrating these collaboratories presents significant challenges, as each of these collaboratories has a unique architecture and implementation, and builds on different enabling technologies. This paper investigates the issues involved in integrating collaboratories operating on the Grid. It then presents the design and implementation of a prototype middleware substrate to enable a peer-to-peer integration of and global access to multiple, geographically distributed instances of the DISCOVER computational collaboratory. An experimental evaluation of the middleware substrate is presented.

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 [2]. Computational collaboratories provide uniform (collaborative) access to computational resources, services, applications and/or data. These systems can expand the resources available to researchers, enable multidisciplinary collaborations and problem solving, accelerate the dissemination of knowledge, and increase the efficiency of research.

The growth of the Internet and the advent of the computational "Grid" [3] have made it possible to develop and deploy advanced computational collaboratories[4][5]. Recent efforts include the Upper Atmospheric Research Collaboratory (UARC) [6], Diesel Combustion Collaboratory (DCC) [7], Access Grid [8], Netsolve [9], EMSL [10], the Astrophysics Simulation Collaboratory [11], which builds on Cactus [12], and DISCOVER [1][13][14]. Each of these systems provides a high-level problem-solving environment (PSE) that builds on the underlying Grid technologies to provide seamless access to domain specific resources, services and applications. Together these systems have the potential for enabling truly global scientific investigation through the creation of meta-laboratories spanning many research groups, universities and countries, and transforming computational applications and services into global modalities for research and instruction.

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

Combining these "focused" collaboratories and allowing them to interoperate presents many advantages. The services provided by the different collaboratories can be reused to reduce duplication of effort. At a higher level, the domain specific services provided by the collaboratories can be combined and composed leading to truly collaborative, multi-disciplinary and multi-institutional problem solving. However, integrating these collaboratories presents significant challenges. These 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 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.

This paper has two objectives: (1) To motivate the need for interoperable collaboratories and to identify the issues involved. (2) To present the design, implementation and evaluation of the DISCOVER middleware substrate that enables interoperability between geographically distributed instances of the DISCOVER collaboratory and is a first step towards achieving interoperability on the Grid. In the first part of this paper we discuss the requirements and mechanisms for achieving interoperability among collaboratories on the Grid. In the second part of the paper we present a prototype middleware substrate for enabling a peer-to-peer integration of, and global collaborative access to multiple distributed instances of the DISCOVER computational collaboratory. Note that in a related effort (CORBA CoG [16]) we have extended this substrate to also enable interoperability between DISCOVER and the Grid services provided by the Globus Toolkit [46].

DISCOVER provides collaborative access to high-performance parallel and distributed applications for interaction and steering using web-based portals [1][13][14]. The middleware substrate enables DISCOVER interaction and steering servers to dynamically discover and connect to one another to form a peer-to-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. The principal design challenge is enabling scalable, secure, consistent and controlled access to remote, highly dynamic distributed applications for real-time monitoring, interaction and steering by geographically distributed scientists and engineers in a collaborative environment. The implementation of the DISCOVER middleware substrate builds on existing web servers and leverages commodity technologies and protocols such as CORBA [17] and HTTP [18] to enable rapid deployment, ubiquitous and pervasive access, and easy integration with third party services. An experimental evaluation of the middleware substrate is also presented.

The rest of this paper is organized as follows. Section 2 describes the motivation for interaction among collaboratories. It also discusses issues and related work in interoperability among collaboratories. Section 3 presents the overall design of the prototype middleware substrate for interoperability on the Grid. Section 4

introduces the DISCOVER computational collaboratory for interaction and steering. This section also describes peer-to-peer integration of a network of DISCOVER servers using the middleware substrate, to provide global collaborative access to remote applications. Section 5 presents the implementation and operation of the DISCOVER middleware substrate. Section 6 presents an experimental evaluation of the DISCOVER middleware substrate. Section 7 presents a retrospective evaluation of the design and implementation of DISCOVER and the technology used. This section also presents an evaluation of the commodity distributed technologies and protocols and their ability to support Grid applications. Section 8 presents some conclusions and outlines current and future work.

2 BACKGROUND AND RELATED WORK

2.1 Current Status of Problem Solving Environments and Computational Collaboratories

The growth of the Internet and the advent of the computational "Grid" [3] have resulted in the development and deployment of advanced problem solving environments and computational collaboratories [4][5]. These include the Upper Atmospheric Research Collaboratory (UARC)[6], the Diesel Combustion Collaboratory (DCC) [7], Access Grid [8], Netsolve [9], EMSL [10], the Astrophysics Simulation Collaboratory (ASC) [11] and Cactus[12], Punch [20], WebFlow [21], Gateway [22], HotPage [23] and GridPort [24], GPDK [25], Commodity CoG Kits [26], Nimrod-G [27], JiPang [28], and DISCOVER [1][13][14]. These systems provide specialized services to their user communities and address different issues in wide area resource sharing and the overall Grid computing problem [15][19]. For example, UARC and ASC implement applications specific PSEs, WebFlow provides support for composing, configuring and deploying scientific applications on the Grid, and systems such as GridPort provide support for acquiring and managing Grid resources.

2.2 Motivations for Interoperable Collaboratories

There are several compelling reasons for allowing multiple types of collaboratories to co-exist and interoperate on the Grid [29]. The systems mentioned above are customized to meet the unique requirements of a specific user community, and provide specialized services and user interfaces that best meet the needs of their users. For example, some systems might require ubiquitous web access through web browsers and therefore use HTTP for access. Other systems might require rich collaboration services among clients and build on a multicast protocol for access. Any effort aimed at building collaboratories on the Grid should accommodate all such preferences.

Furthermore, domain specific services provided by the collaboratories can be combined and composed leading to truly collaborative, multi-disciplinary and multi-institutional problem solving. For example, one could combine the physical models provided by ASC and UARC, visually compose and configure an application using WebFlow, allocate resources, deploy and run the application using PUNCH, collaboratively interact and steer the applications using DISCOVER, and if the application generates large amounts of real time data, one could broadcast it to participating clients using the Salamander data dissemination substrate [30] (used in the UARC and the IPMA project [31]). Building each system to provide all required capabilities would not only lead to duplication but is rapidly ceasing to be a viable option. However, as most of these systems are standalone with customized architectures, combining them in the fashion outlined above can be a significant challenge. For example, these systems use different underlying protocols and enabling technologies - WebFlow and DISCOVER use CORBA and

HTTP, PUNCH uses HTML and CGI, while Salamander uses a customized API (application programming interface) written in C/Java/Perl.

The need for an intermediate interoperability layer on top of the Grid has been emphasized earlier [29]. Such a layer will provide basic concepts and mechanisms that can be shared by collaboratories on the Grid, avoiding duplication of effort and core development, and allowing individual systems to focus on domain specific issues and the needs of their user community. Note that the access modes, client-server interaction protocols and user interface designs typically need to be customized for specific domains. Therefore, such a common interoperability layer should only be implemented at the middle tier of a typical 3-tier architecture.

2.3 Approaches to Interoperability

As motivated above, a middleware layer on top of the Grid is one means of achieving interoperability. Such a middleware can be defined as a set of interoperable high-level services providing functionality that is common to these collaboratories, and will enable collaboratory developers to compose these services to develop new and specialized user level services for their specific user community. We believe that without a standard set of high-level services, collaboratories will continue to implement this common functionality in customized ways, resulting in non-reusability and lack of interoperability. Following this approach, we identify three distinct ways to implement interoperable higher-level services – same implementation everywhere, same interface and/or API everywhere, and same protocol everywhere.

Shared Implementation: In this approach, the services are built into a toolkit and all users use the same toolkit. An example of this approach is the use of different commercial instant messaging software available from Yahoo, Microsoft, etc. Users of these systems can only share messages with other users with the same software. The scalability, extensibility and the level of interoperability of such an approach leaves much to be desired.

Shared Interfaces and APIs: In this approach, each system publishes a set of APIs and interfaces for its services. This approach is used by CORBA applications where the APIs are specified using its IDL (Interface Definition Language) and these IDLs are shared by all systems. The approach by Fox et al. [32] described below also uses this approach. This solution is feasible for moderate numbers of systems and services, and is most widely used. However, it requires all implementations of a service to conform to a common interface and other systems to use this interface to access the service, and will not provide truly global sharing and interoperability on the Grid.

Shared Protocols: The third approach is to have each system communicate using the same protocol. As identified by Foster et al. in [15][33], true interoperability in a networked environment can only be achieved by using common protocols. A protocol definition specifies how distributed elements interact with one another to achieve a specified behavior, and the structure of the information exchanged during this interaction. It defines the format of the data that is sent between two systems, including the syntax of messages, character sets, and sequencing of messages. The most scalable and interoperable system today is the Internet and the World Wide Web and the major forces behind their success are standard protocols like TCP/IP and HTTP.

Note that true (protocol-based) interoperability can be achieved using CORBA by leveraging the fact that CORBA uses the IIOP (Internet Inter-ORB Protocol) protocol for all communication. The services to be shared can be built into the CORBA ORB as standard CORBA services, and can be accessed in a standard way using IIOP.

This is analogous to the sockets API supported by various implementations of TCP/IP. Although this does involve a standardization process we believe it is feasible.

2.4 Related Work

Although interoperability has been identified as a central issue for Grid based systems in previous work [3][15] [33][34], there has been limited progress towards achieving this goal. This is particularly true in the case of computational collaboratories. Although, there have been specific efforts aimed at bilateral sharing and interoperability such as those between Ninf [35] and NetSolve [9], these have been made possible because of joint development efforts by their respective development teams. These efforts only further highlight the benefits and significance of interoperability and the need for having a general solution for interoperability.

The Collaboratory Interoperability Framework (CIF) Project [34] addresses the interoperability problem by proposing a common communication API that can be used by collaboratory developers to build tools for collaboration such as videoconferencing, text-based collaboration through chat, whiteboard and electronic notebooks. The idea is that, since these tools will use a common communication API that hides the details of the underlying protocol used, they should be able to interoperate with each other. While this approach can provide low-level interoperability at the communication layer, collaboratories will continue to build customized services on top of this layer and cease to be interoperable.

Another approach for enabling interoperability has been recently presented by Fox et al. in [32]. This approach characterizes portals as web based interfaces to applications. In particular it focuses on portals for computational science and web based education. This approach takes the view that interoperable portals should be based on interface standards, which are essentially hierarchical frameworks in the Java approach but are probably best defined in XML. These portal frameworks are based on a 3-tier architecture which uses two interface definitions based on XML. These are the resource markup language (resourceML) that describes the basic learning or computing objects and the portal markup language (portalML) that describes the user view of the portal.

3 Design of a Prototype Middleware Substrate for Grid-based Collaboratories

The overall goal of the middleware substrate (and the related CORBA CoG effort [16]) is to define interfaces and mechanisms for a peer-to-peer integration and interoperation of the services provided by domain specific collaboratories. The middleware design builds on existing web servers and leverages commodity technologies and protocols such as CORBA [17] and HTTP [18]. Its goal is to enable rapid deployment, ubiquitous and pervasive access, and easy integration with third party services, while evaluating the viability of these technologies for advanced Grid applications. Interoperability in the current implementation of the middleware is achieved by sharing interfaces defined in the CORBA IDL.

The overall architecture of our prototype is hybrid rather than pure peer-to-peer or client-server. While the middleware substrate provides a client-server architecture from the users' point of view, 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. Security and manageability are still open issues in true peer-to-peer systems, while one of the reasons for the success of client server systems is the security and manageability associated with having centralized servers. The smaller number of peer servers 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). As the functionality of a centralized server is distributed across multiple peer servers in a peer-to-peer system, it makes them more manageable and lightweight as compared to a centralized server in a client server system, even though each server has an ORB running. A server may be deployed anywhere there is a growing community of users, much like a HTTP Proxy server.

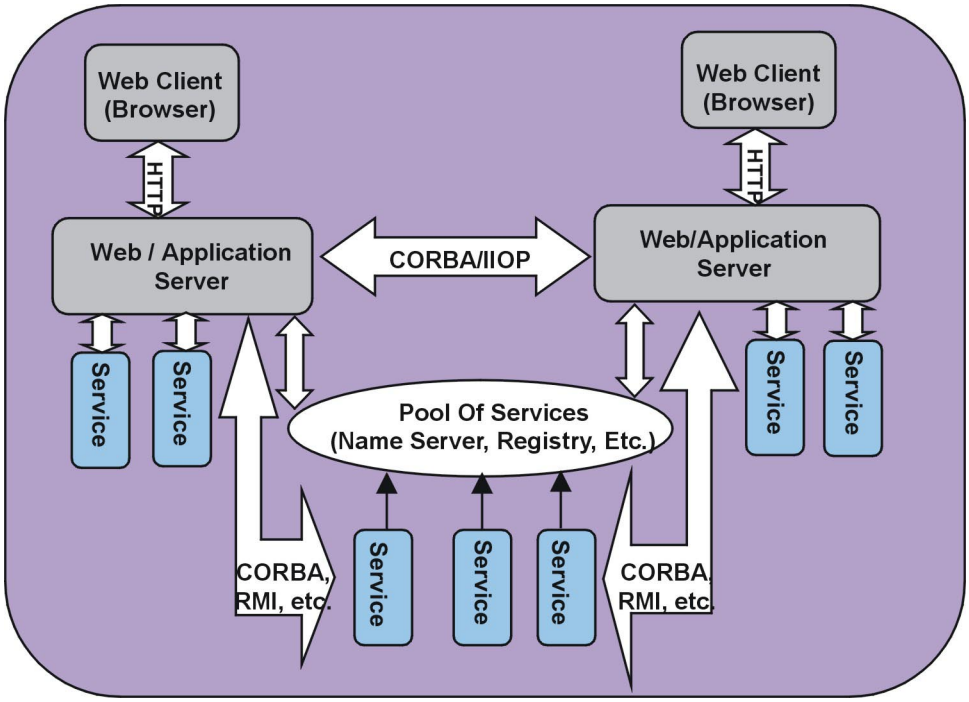

Figure 1. Middleware design for integrating computational collaboratories

A schematic overview of the middle substrate is presented in Figure 1. 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. In order to enable ubiquitous web-based access, clients are kept as simple as possible. The responsibilities of the middle-tier 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 always connects to its "closest" server and has access to all (local and remote) backend services based on its privileges and capabilities. Backend services include resource access and management toolkits, high-performance applications, data archives, and network-monitoring tools. These 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. This is true for many scientific resources and applications. In this case, the local server advertises the service and its interface, and clients and peer servers can discover and access the service through the local server. The server may also wrap this service as a distributed object and bind it to a naming service, registry or a trader service. In either case, the servers and the backend services are accessed using standard distributed object technologies such as CORBA/IIOP, RMI, and DCOM.

The middleware design defines two levels of interfaces for each server. The first level interfaces enable other peer servers to authenticate with a server and query it for active services and users. The second level interfaces are used for authenticating with and accessing a specific service at the server. If a server provides a single application or a service only the second level interfaces may be provided as the server and the service do not have to be identified separately – i.e. the service itself can represent the server. The pool of services model is implemented using the second level interfaces.

The portalML and resourceML interfaces for interoperable web portals presented in [32], are similar to the two levels of interfaces described above. However, the first level interfaces, instead of providing a user view of a portal as in portalML, define a system view of a server, which can be used, by other servers to access its services. The second level interfaces describe a particular service at a server, which is similar to the resourceML. In our current design, these interfaces are defined in the CORBA IDL instead of XML (which is used by portalML and resourceML). The choice between CORBA IDL and XML is a trade-off between speed and loose coupling. XML is self-describing and can provide a greater level of interoperability. However, XML parsing is still an overhead and is significantly slower than CORBA IDL based object marshalling. CORBA also provides more sophisticated services such as discovery and naming.

The DISCOVER middleware substrate presented in the following sections is a prototype implementation of this design to achieve interoperability between multiple, distributed instances of the DISCOVER computational collaboratory.

4 DISCOVER: A COMPUTATIONAL COLLABORATORY FOR INTERACTION AND STEERING

DISCOVER is a virtual, interactive computational collaboratory that enables geographically distributed scientists and engineers to collaboratively monitor, and control high performance parallel/distributed applications. Its primary goal is to bring remote distributed simulations to the scientists'/engineers' desktop by providing collaborative web-based portals for interrogation, interaction and steering. The DISCOVER architecture (see Figure 2) is composed of detachable client portals at the front-end, an interaction server in the middle, and a control network of sensors, actuators, and interaction agents superimposed on the application at the backend. Clients are implemented as Java applets and they communicate with the server over HTTP. They 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 privileges and capabilities. The middle tier consists of Interaction and Collaboration servers, which extend commodity web-servers

with interaction and collaboration capabilities. The backend consists of a control network composed of sensors, actuators and interaction agents. Session management and concurrency control is based on capabilities granted by the server. A locking protocol is used to ensure that the applications remain in a consistent state during collaborative interaction and steering. Security and authentication services are provided using customizable access control lists built on the SSL-based secure server. DISCOVER is currently operational[2] 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. Details about the design and implementation of the DISCOVER collaboratory can be found in [14].

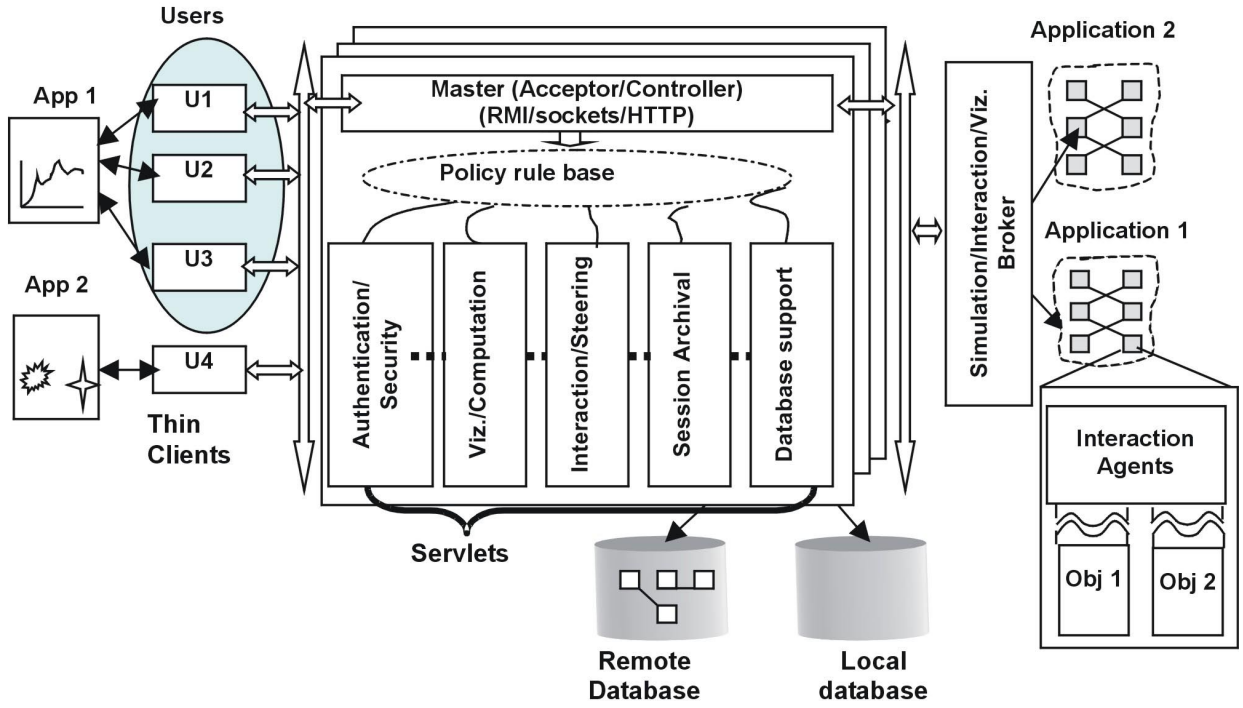

Figure 2. Architectural schematic of the DISCOVER computational collaboratory

4.1 A Middleware Substrate for Peer-to-Peer Integration of DISCOVER Servers

The DISCOVER middleware substrate integrates multiple instances of the DISCOVER computational collaboratory so that a client can access and interact with all the applications for which it has access privileges, regardless of whether they are local or remote. It is motivated by a number of requirements. First, having all applications connect to a single DISCOVER server or having a centralized repository of servers are not scalable options. Furthermore, security constraints often prevent applications from connecting to remote servers outside their domain. This is true for applications executing on most high-end resources. Finally, applications typically do not provide standard access interfaces for interaction and steering, and need to be coupled to their server using a proprietary protocol. The proposed peer-to-peer server architecture with coupled server/application(s) sets is more appropriate for such integration.

[2] See http://www.discoverportal.org

The DISCOVER middleware consists of multiple independent collaboratory domains, each consisting of one or more DISCOVER servers, and applications connected to the server(s). The middleware can be extended to include other servers and services using the "pool of services" model described earlier. For example, the middleware can provide access to a monitoring service, a data archival service or grid services using Java/CORBA CoG Kits. A domain typically consists of different types of servers. All the servers within a domain share a common database with information about users and applications/services. Servers may also share the same security mechanism. Note that the availability of these servers is not guaranteed and must be determined at runtime using a discovery mechanism. A client can connect to a server within its domain (using HTTP), and have secure and authorized access to all applications/services in the entire system. The current configuration of the DISCOVER server network is shown in Figure 3.

The DISCOVER middleware substrate builds on CORBA/IIOP and provides peer-to-peer connectivity between servers within and across domains. Server/service discovery mechanisms are built using the CORBA *Trader Servic*e [38], which allows a server to locate remote servers and to access applications connected to those remote servers. Although CORBA does introduce some overheads, it enables scalability and high availability and provides the services necessary to implement the middleware substrate. It also allows interoperability between servers, while allowing them to maintain their individual architectures and implementations. Moreover, servers are typically connected via link with reasonable bandwidth (~1 Mbps). As no assumptions can be made about client-server connections, having the client connect to the "nearest server", and use CORBA/IIOP to connect the server and the desired application may actually reduce client latencies in some cases. This is because clients (implemented as Java applets) communicate with their "home" server using HTTP and their home server communicates with remote servers on the clients' behalf using IIOP. Since IIOP (unlike HTTP), reuses connections and hence reduces connection overheads, its use over the larger network path helps in reducing client latencies when a large geographical distance separates the two communicating servers, and small chunks of data are transferred (<20Kbytes). This is shown in the experiments presented in Section 6.1.

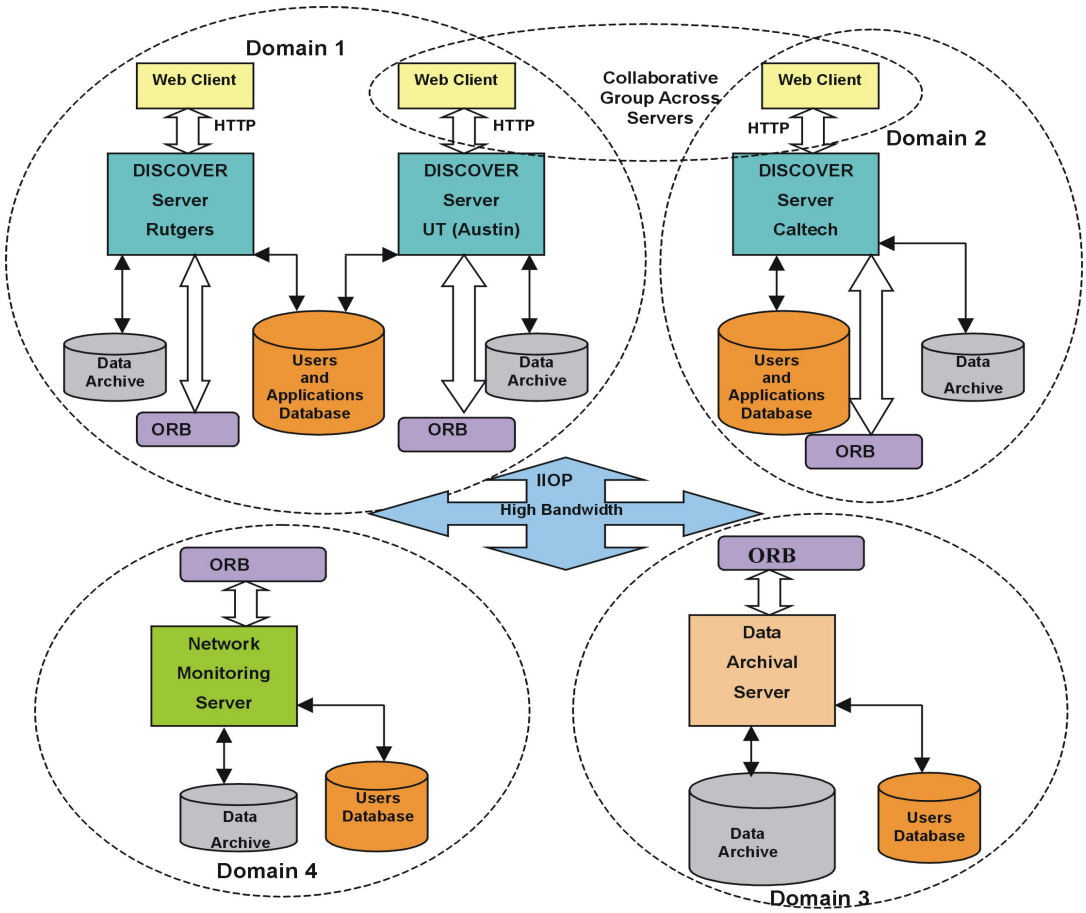

Figure 3. Deployment of DISCOVER servers providing access to a repository of services

The following section describes the implementation and operation of the DISCOVER middleware substrate. The middleware substrate supports the following operations:

- Security, authentication and access control across servers: Since clients will be accessing applications connected to remote servers, the middleware substrate is able to authenticate a client with remote servers and/or remote applications and control access based on privileges and capabilities.
- Collaboration and interaction across servers: In a network of servers, an application might be connected to one server, and clients from different servers might want to collaboratively interact with it. Clients interacting with the same application can form a collaborative group, even if they are connected via different servers.
- Data/State consistency across servers: DISCOVER servers use a simple locking mechanism to ensure that applications remain in a consistent state during collaboration and interaction sessions. The middleware substrate extends this locking mechanism to handle multiple clients from multiple servers.
- Logging capabilities across servers: Since clients from any server can access an application, the application and the client logs are maintained separately.

5 IMPLEMENTATION AND OPERATION OF THE DISCOVER MIDDLEWARE SUBSTRATE

5.1 DISCOVER Interaction and Collaboration Server

The DISCOVER interaction/collaboration servers build on commodity web servers, and extends their functionality (using Java Servlets [36]) to provide specialized services for real-time application interaction and steering and for collaboration between client groups. Clients are Java applet 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 [17] and Java RMI [37], or a more optimized, custom protocol over TCP sockets. An *ApplicationProxy* object is created for each active application at the server, and is given a unique identifier. The application identifier is a combination of the server's IP address and a local count of the applications on the server. 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. These communication channels are abstractions for server-application communication and in the current implementation these channels correspond to three different socket connections. At the other end, clients differentiate between the various messages (i.e. Response, Error or Update) using Java's reflection mechanism (i.e. by querying the received object for its class name). Use of reflection to differentiate messages from the server at the client side has been incorporated to use Java's object serialization for all communication between the client and the server. This eliminates the need for message parsing at the client side as entire Java objects can be sent from the server and their type is determined dynamically through refelction. Although it might be a little slow, object serialization makes the client code much cleaner and simpler.

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. In addition to these core handlers, there can be a number of additional handlers providing auxiliary services such as session archival, database handling, visualization, request redirection, and remote application proxy invocations (using CORBA). These services are optional and need not be provided by every server. Details about the design and implementation of the DISCOVER Interaction and Collaboration servers can be found in [14].

5.2 Implementation of the Middleware Substrate for peer-to-peer Integration of DISCOVER Servers

The DISCOVER middleware substrate builds a peer-to-peer network of DISCOVER servers by implementing the two levels of interfaces described in Section 3. The *DiscoverCorbaServer* interface is the level one interface and represents a server in the system. This interface enables peer servers to discover, authenticate and interact with one another. The *CorbaProxy* interface is the level two interface and represents an application/service at a server. This interface defines the functionality exported by the application/service and allows remote servers to access this functionality. The middleware substrate builds on the DISCOVER interaction/collaboration server architecture described above. Server-server communication uses the three communication channels set up for application-server communication, i.e. *Main Channel*, *Command Channel* and *Response Channel* (see Section 5.1), and establishes an

additional *Control Channel* for error messages and system events. The *Control Channel* implements a notification service similar to the one used in the Salamander substrate [30]. The schematic of the middleware implementation is presented in Figure 4. The two interfaces are described below.

5.2.1 The *DiscoverCorbaServer* Interface

The *DiscoverCorbaServer* interface is implemented by each server and defines the methods for interacting with the 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 represents the server within the peer-to-peer system. The *DiscoverCorbaServer* object publishes its availability using the CORBA trader service. It also maintains a table of references to *CorbaProxy* objects (i.e. *CorbaProxyInterface*) for remote applications. These references are used to provide transparent access to the associated remote applications and to enable local clients to interact with these applications – i.e. all interactions with remote applications from locally connected clients go through the *DiscoverCorbaServer*, which then forwards them to the appropriate *CorbaProxyInterface* reference.

5.2.2 The *CorbaProxy* Interface

The *CorbaProxy* interface represents an active application (or service) at a server. This interface specifies all the methods required for accessing, interacting with and steering the application. This includes methods for querying application status, querying and changing application parameters, requesting steering controls (locks) and issuing commands. The *CorbaProxy* object also binds itself to the CORBA naming service using the application's unique identifier as the name. This allows the application to be discovered and remotely accessed from any server. The *DiscoverCorbaServer* objects at all remote servers that have clients interacting with a remote application maintain a reference to the *CorbaProxy* object for that application. This reference thus serves as the gateway to the applications.

The *CorbaProxy* object is contained within each DISCOVER *ApplicationProxy* created at the server. As described above, the *ApplicationProxy* manages all interactions with the application. In the case of local applications, the *ApplicationProxy* directly communicates with the applications, while in the case of remote applications this communication is through the local reference (i.e. *CorbaProxyInterface*) to the remote *CorbaProxy* object.

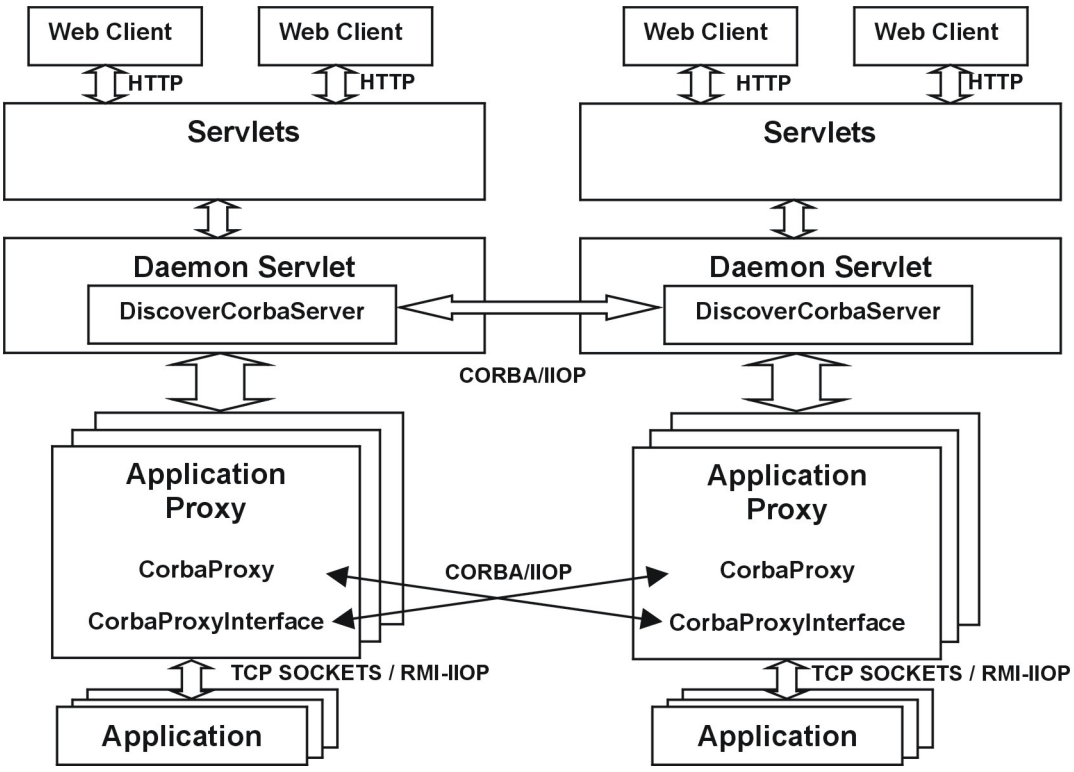

Figure 4. Interaction Model between DISCOVER Servers

5.3 Middleware Operation

This section describes the operation of key mechanisms across multiple instances of the DISCOVER computational collaboratory, namely, server and applications discovery, security/authentication across servers, collaboration across servers, distributed locking, and distributed information logging.

5.3.1 Discovery of Servers and Applications

Peer DISCOVER servers locate each other using the CORBA trader services. The CORBA trader service maintains all the server references as *service-offer* pairs. In our current system we have implemented a minimal trader service on top of the CORBA naming service. All DISCOVER servers are identified by the service-id 'DISCOVER'. The service offer is a CORBA *CosTrading* module (CORBA Trader service specification), which encapsulates 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 list of properties.

Typically, an application will connect just to its local server. However, in the case of meta-applications, different application components may connect to different servers (without overlap). There is also the possibility that there are multiple instances of an application (run by different groups) on the same or different servers. A scheme was chosen for assigning globally unique identifiers to applications such that the identifiers are unique for both of these cases. These identifiers are dynamically assigned by the Daemon Servlet and each identifier is a combination of the server's IP address and a local count of the applications on each server, guaranteeing that even if the same application is connected to multiple servers or multiple instances of the application are connected to the same server, each instance will have a unique identifier. Furthermore, the server's IP address can be extracted from

this application identifier, making it very easy to determine if the application is a local application or a remote application.

5.3.2 Security/Authentication across Servers

Each DISCOVER server supports a two-level client authentication; the first level authorizes access to the server and the second level permits access to a particular application. To control access, all 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. For access to remote applications, the security handler uses the *DiscoverCorbaServer* interface to authenticate the client with each server in the network, and in return receives the list of all active applications connected to all the servers to which the user has access privileges. Once the client selects a remote application, the second level authentication is performed to get a customized interaction/steering interface for the application based on the client's access privileges. As a result each client can access only those applications that it is authorized to, and can only interact with them in ways defined by its privileges and capabilities. Note that a client has access only to those servers where it is a registered user.

5.3.3 Collaboration across Servers

The DISCOVER collaboratory enables multiple clients to collaboratively interact with and steer (local and remote) applications. The *collaboration handler* servlet within 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 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. They use the *main channel* for retrieving global messages, the *response channel* for retrieving response messages generated in response to a clients' requests, and the *control channel* for retrieving any error messages or steering control (lock) updates. In the case of multiple instances of an application connected to a single server or to multiple servers, all clients (possibly from different servers) connected to a specific application instance form a collaboration group. The clients can prevent invasive communication from other clients by choosing not to participate in the collaboration and disabling all collaboration updates.

The peer-to-peer architecture offers two significant advantages for collaboration. First, it reduces the network traffic generated, by reducing the large number of broadcast messages that would be typically sent by a server to all the participants of the collaboration session. 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. This is illustrated in Figure 5. 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.

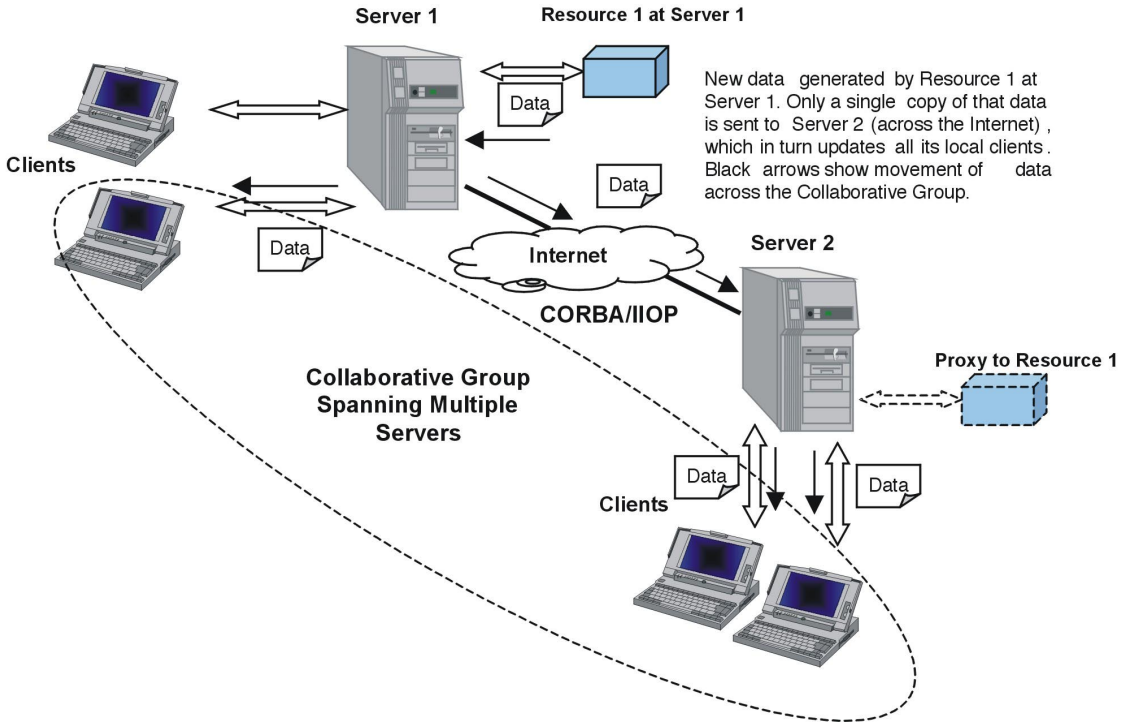

Figure 5. Collaborative Group spanning multiple servers

5.3.4 Distributed Locking 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 a distributed server case, locking information is only maintained at the application's host server i.e. the server to which the application connects directly. Servers providing remote access to the application only relay lock requests to the host server and receive locking information from the host server. Thus using the application's host server as the controller of the session guarantees consistency during interaction and collaboration.

5.3.5 Distributed Logging

The session archival handler maintains two types of logs. The first logs all interactions between a client and an application. This log enables clients to replay their interactions with the applications. It also enables latecomers to a collaboration group to get up to speed. For remote applications, the client logs are maintained at the server where the clients are connected. The peer-to-peer architecture assumes that there is an application running on a remote server, and the application information is sent to all servers that have clients interested in that information. Thus all client logs are handled by the server to which is connected to and this server creates the necessary output files or records under the ownership of the client. The architecture does not allow files to be created on a remote server.

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 and allows clients to have direct access to the entire history of the application.

6 AN EXPERIMENTAL EVALUATION OF THE DISCOVER MIDDLEWARE SUBSTRATE

The DISCOVER collaboratory is currently operational and the current server network includes server deployments Rutgers University and the Center for Subsurface Modeling (CSM), University of Texas at Austin. We are currently expanding the network to include a deployment at the Center for Advanced Computational Research (CARC), California Institute of Technology. Figure 6 shows the setup used for the experimental evaluation presented in this section. The middleware implementation used the Apache Web Server 1.3 [39] with Apache Jserv 1.1.12 [40] as the servlet engine, and Visibroker for Java 4.5.1 [41] as the CORBA ORB. The evaluation consists of three experiments, viz. access latency over local area and wide area networks, effect of multiple clients on access latencies and server memory overheads due to local and remote applications.

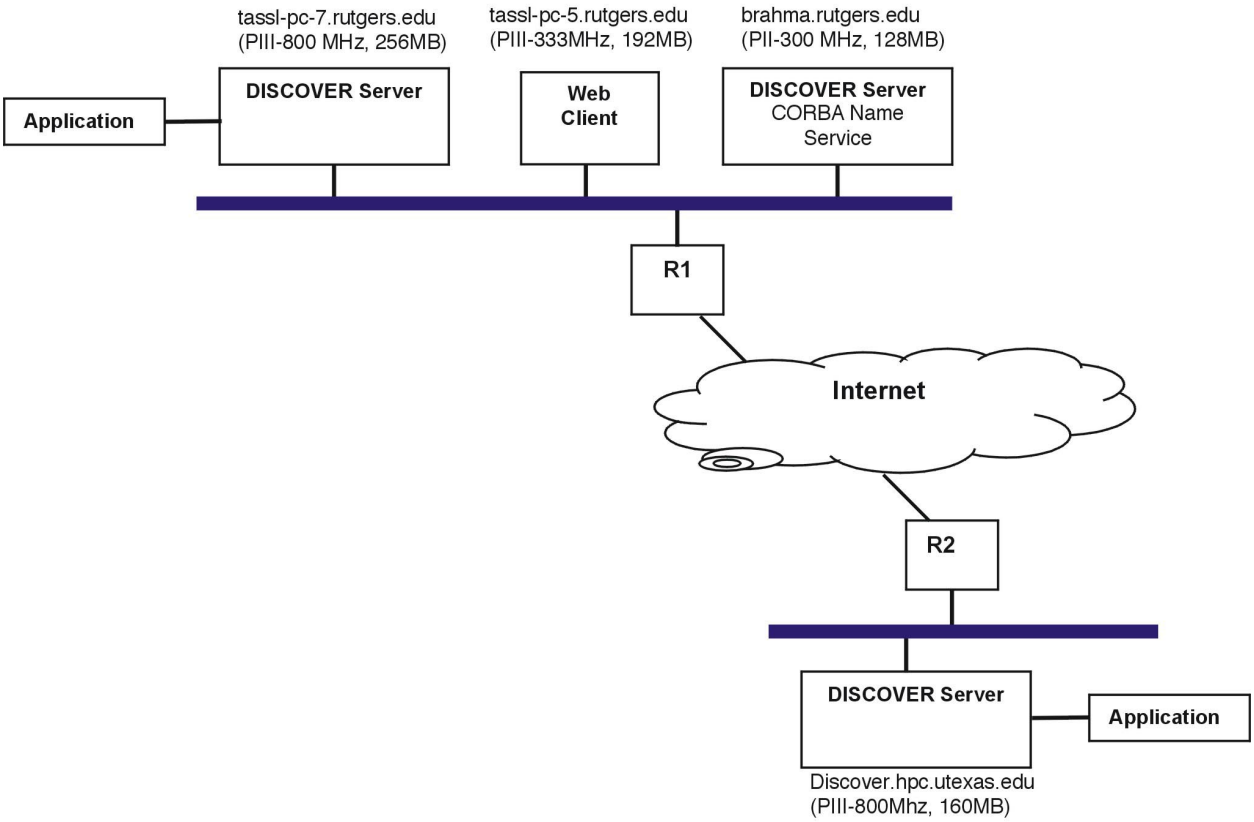

Figure 6. Setup for the experimental evaluation of the DISCOVER middleware

6.1 Experiment 1 –Access Latency over Local Area Networks (LAN) and Wide Area Networks (WAN)

This experiment consisted of two sets of latency measurements. The first set of measurements was for a 10Mbps local area network (LAN) and used DISCOVER servers at Rutgers University. The second set of measurements was for a wide area network (WAN) and used DISCOVER servers at Rutgers University and at CSM, University of Texas at Austin. The clients were running on the local area network at Rutgers University for both sets of measurements.

In this experiment an application was connected to one of the servers, and a minimal client (without any user interface) was used to access and interact with the application. In case of the LAN measurements (set 1) the

application was connected to one of the servers at Rutgers University while in case of the WAN measurements (set 2), the application was connected to the server at CSM, University of Texas at Austin. The client requested data of different sizes from the application, and 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. Direct access times included the time taken for the client's request to be sent to the server over HTTP, the server handling the request and forwarding the request to the application, the server getting and processing the response from the application, and the response being sent back to the client. 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.

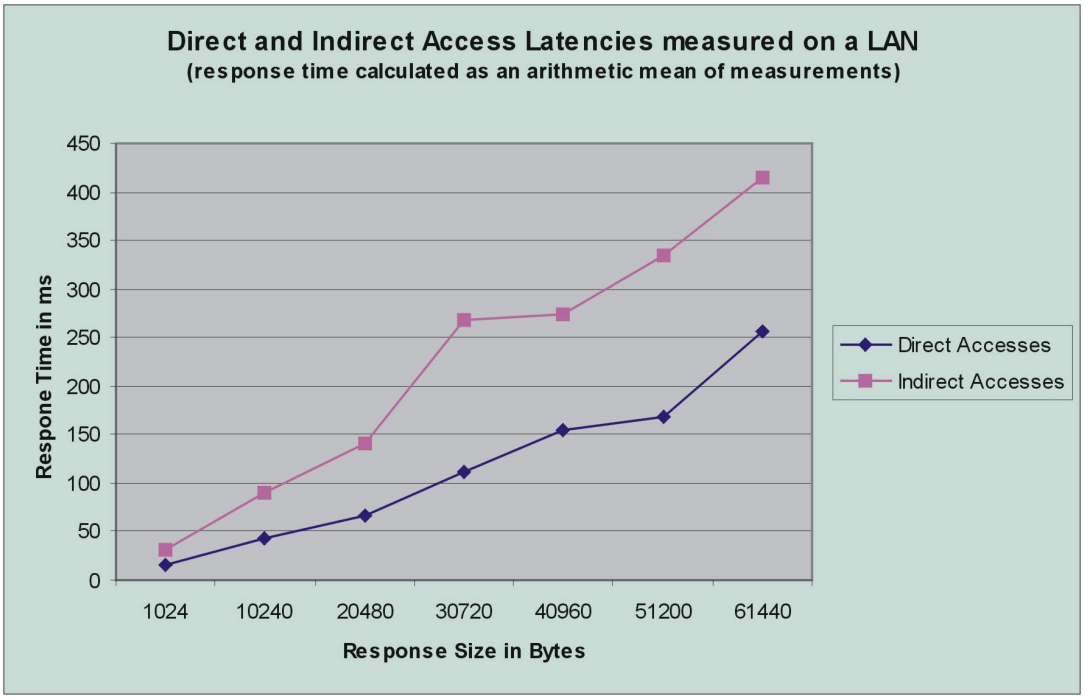

Figure 7. Comparison of latencies for direct and indirect application accesses on a Local Area Network (LAN)

The resulting response latencies for direct and indirect accesses measured on the LAN are plotted in Figure 7. It can be seen that, as expected, the response times for direct accesses to an application at a local server increases with the increase in response size. This increase in latency is primarily due the increased communication times as the server overhead for request-response handling is almost constant. The response times for indirect accesses to an application at a remote server also increase with increase in data size. Indirect access times are almost twice the direct access times, which is not surprising as an indirect access includes the time for a direct access. However, it should be noted that the difference in indirect and direct access times approaches a constant as the data size increases. The obvious conclusion from these results is that it is more efficient to directly access an application when it is on the same LAN.

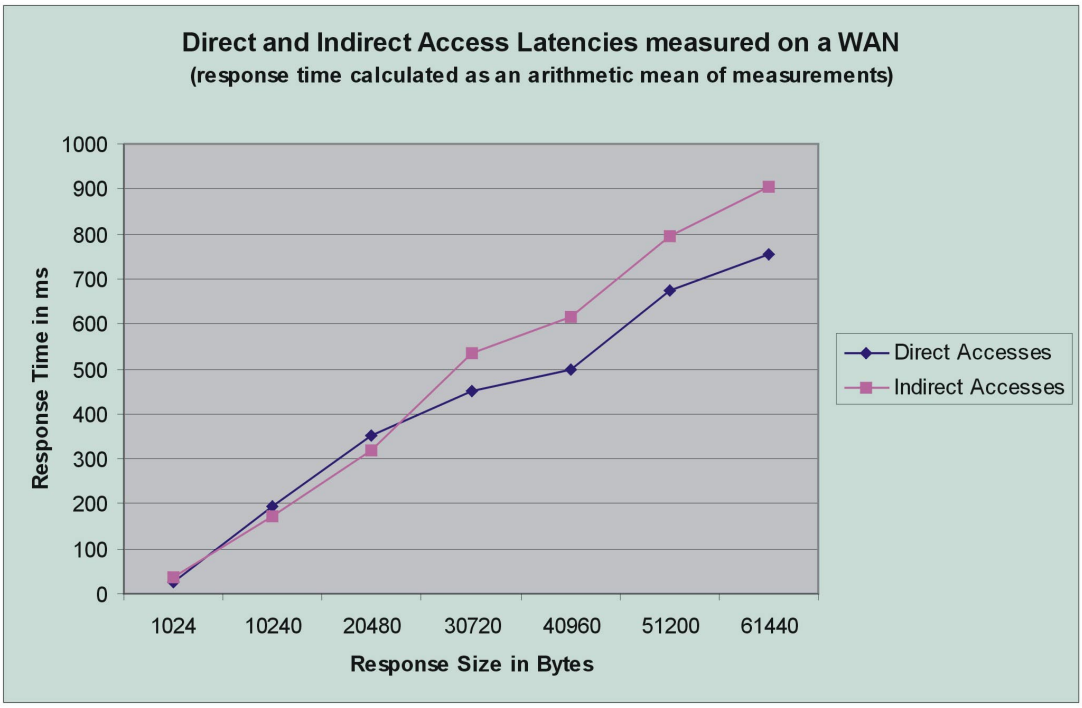

Figure 8. Comparison of latencies for direct and indirect application accesses on a Wide Area Network (WAN)

The response latencies for direct and indirect responses measured on the WAN are plotted in Figure 8. In contrast to the results for the LAN experiment, indirect access times measured on the WAN are of comparable order to direct access times. In fact, for small data sizes (1K, 10 K and 20K) indirect access times are 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 indirect access measurement, the application was connected to the server at Austin, while both the client and locally accessed server were running on machines at Rutgers University. The indirect access consisted of the client at Rutgers accessing the local server at Rutgers over HTTP, which in turn accessed the server at Austin over IIOP. 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 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. Also note that, in both cases, this latency was less than a second.

6.2 Experiment 2 –Access Latency with Multiple Simultaneous Clients

This experiment measured the variation of direct and indirect access latencies in the presence of multiple simultaneously connected clients over a LAN. The setup used for this experiment was the same as that for experiment 1 (Section 6.1) for the LAN. In this experiment, an application was connected to one of the servers and multiple clients simultaneously accessed and interacted with the application. Each client requested data of size 20 Kbytes. Response times for direct access to the server with the application and indirect access through the middleware substrate were measured. These results are plotted in Figure 9. The results show that the response times more or less hover around the average response times for a single client for 20 Kbytes of data (see Figure 7), for both direct and indirect accesses. The 3 points towards the right end of the graph (for 16, 17 and 19 clients) are most probably due to the communication and network irregularities – we are unable to explain these values and are in the process of repeating these experiments.

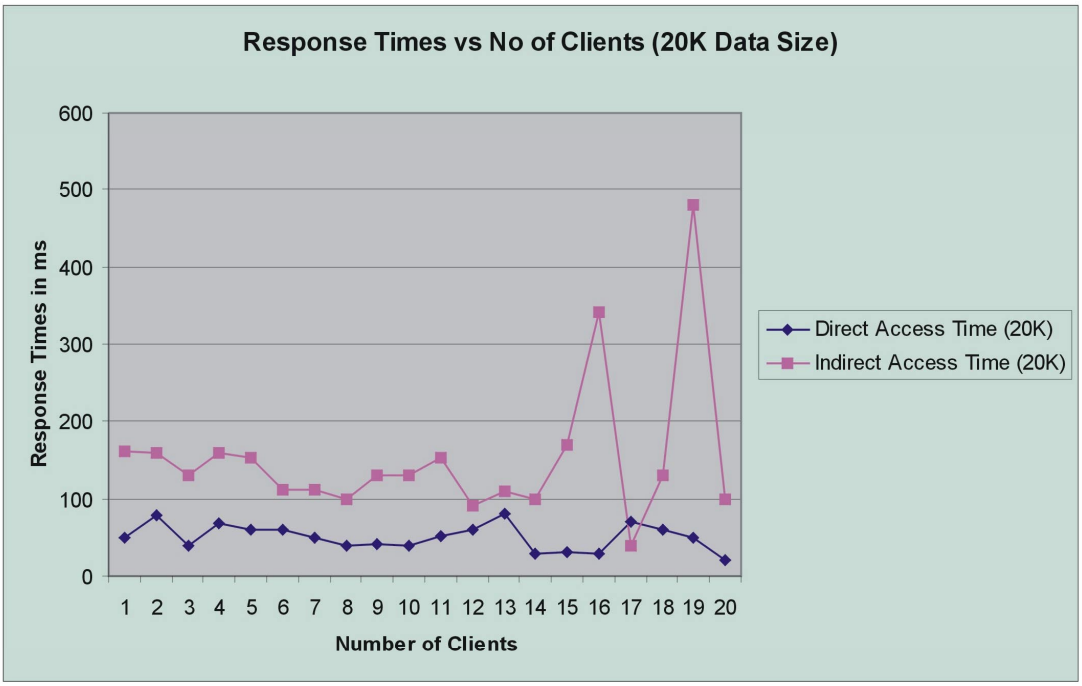

Figure 9. Variation in access latencies with multiple, simultaneous clients over a LAN

6.3 Experiment 3 – An Evaluation of Server Memory Requirements

This set of experiments was conducted to evaluate the server memory requirements for different configurations when multiple applications were connected to it. The motivation for this experiment was our design goal to make the DISCOVER servers lightweight and easily deployable. Memory usage was computed as the difference between the total memory available to the Java Virtual Machine (JVM) for current and future objects and free memory available for future objects. The method calls *freeMemory()* and *totalMemory()* defined in the *java.lang.Runtime* class were used for this purpose. These methods return an approximation of the total bytes of memory currently available for future allocated objects and the total bytes of memory currently available for current and future objects respectively. Since these methods return only an approximate value and this value is for the entire JVM rather than a single process within the JVM, the calculated values for memory usage are also approximates and the actual values will be

slightly lower than those plotted here. However, we made sure that only the server process was operational in the JVM during the experiment, so that changes in the memory usage reflected memory allocated for new applications that connected to the server.

Three different server configurations were used in this experiment. In the single standalone server configuration the server had no ORB or Naming Service running, and made no CORBA invocations. In this case there were no updates generated for remote servers. A single client running on a different machine connected to active applications and generated requests for 1K of data, and memory usage was measured as the number of applications connected to the server was increased. The next 2 configurations consisted of a network of two servers with a local server (the server to which the application was directly connected) and a remote server (the server accessing an application remotely). The goal of this experiment was to measure the memory use at a local server when it publishes its local applications for remote accesses. The applications connected to its local server and this server created the required CORBA objects for the applications (i.e. *CorbaProxy* objects), which then registered themselves with the CORBA Naming Service. CORBA updates for remote servers were generated in this case. A single client running on a different machine accessed one of the applications directly through the application's local server. Memory usage was measured at the local server (the server where the application was connected). In the final experiment, multiple clients were used to access multiple applications remotely through CORBA. As for the previous configuration, the memory utilization at the local server was measured – however in this case there were multiple clients accessing multiple applications remotely instead of a single client accessing one of the many applications directly. In this experiment, the memory utilization at the remote server was also measured. Remote server memory utilization includes the memory required for storing remote CORBA references to applications and invoking IDL methods on them. The measured memory utilizations for these experiments are plotted in Figure 10.

As expected, the memory usage increases for each configuration as the number of applications is increased. However, it should be noted that the overall memory required at the server was not significant, i.e. less than 10 MB. It is interesting to see that the memory requirements are smaller when a server accesses remote applications as compared to when the applications connect directly to the server – i.e. the single standalone server where all applications are local (with no CORBA objects) requires more memory than a remote server having a remote reference to the application. However, publishing a local application for remote access (creating CORBA objects) almost doubles the memory requirements. Thus, local servers (i.e. servers to which the application connects directly) publishing applications have larger memory requirements, while remote servers accessing these applications using the middleware substrate have reduced memory requirements. Note that the memory required for maintaining client state at the server is not significant. This can be inferred from the local server memory requirements for a single direct client and multiple remote clients.

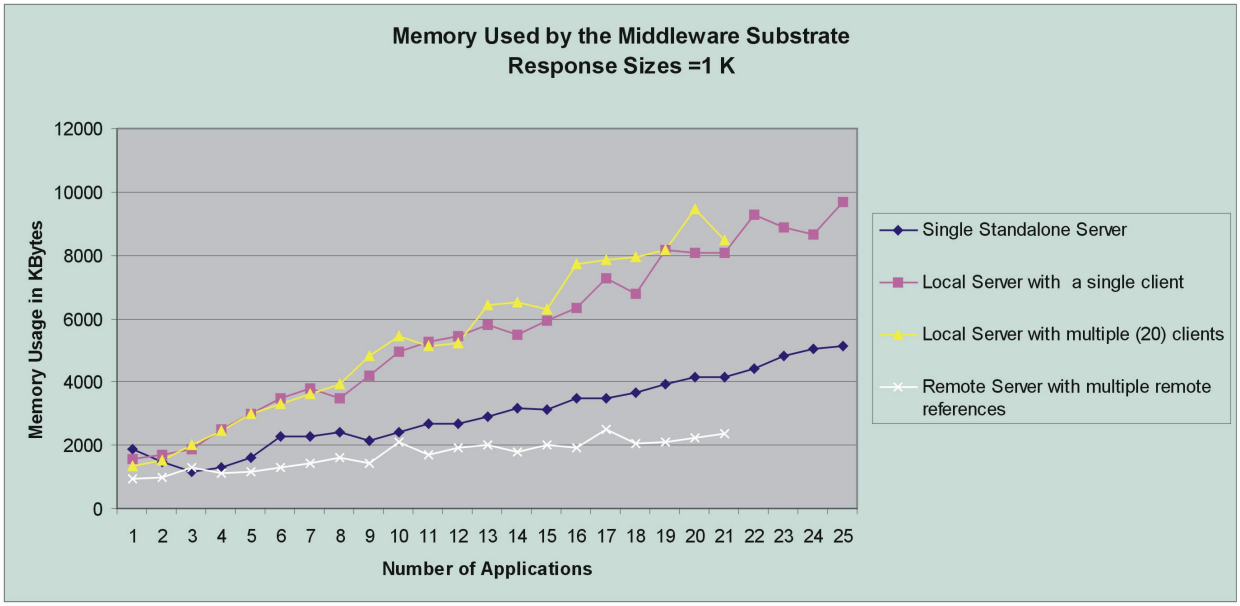

Figure 10. Sever memory utilization for different configurations

7 RETROSPECTIVE EVALUATION OF THE DESIGN AND TECHNOLOGIES USED

The primary goal of the solutions presented in this paper is to support wide deployment and global access; as a result we build on widely used commodity distributed technologies. For example, access to DISCOVER is provided using thin web browsers and the ubiquitous HTTP protocol. Our implementation builds on existing HTTP servers and adds new services, rather than building customized servers from scratch. The choice of CORBA as the middleware substrate is motivated by its inherent support for peer-to-peer interactions. It enables seamless integration with 3rd party custom servers, thereby achieving interoperability through shared IDLs. Integration of these IDLs as standard CORBA services into an ORB is the next step towards true interoperability. The use of IIOP for communication among peer servers produces latencies that are comparable to directly accesses to remote servers using HTTP over a wide area network, while enabling scalability, higher availability and an enhanced set of services and applications for the users.

The use of these commodity technologies however is not without its disadvantages and limitations. While the use of HTTP for client-server interactions provides ubiquitous pervasive access through standard web browsers, it necessitates a poll and pull mechanism for fetching the data from the server instead of a push mechanism (as HTTP is a request-response protocol). The poll and pull mechanism makes it necessary to maintain buffers for each client at the server in order to support slower clients. Such a poll and pull mechanism may be unsuitable for large virtual reality collaborative environments where 3D data is involved, as it presents both memory and performance overheads. Similarly the use of CORBA as the middleware technology causes the middleware to give up control over its transport and communication policies and reduces performance when compared to a lower level socket based system. Furthermore, in our experience, the current commercial CORBA ORBs leave much to be desired, especially in the areas of high performance and interoperability.

8 CONCLUSION AND FUTURE WORK

This paper presented the design implementation, and operation of a middleware substrate that enables a peer-to-peer integration of and global collaborative access to multiple, geographically distributed instances of the DISCOVER collaboratory. The substrate builds on the CORBA distributed object technology and enables dynamic application/service discovery, remote authentication and access control, coordinated interactions for collaborative interaction and steering. An experimental evaluation of the middleware substrate and a retrospective evaluation of the design were also presented. The performance of the middleware substrate over a wide area network validated the middleware design and justified the use of CORBA/IIOP for inter-server communication. The DISCOVER middleware architecture is currently operational and provides collaborative interaction and steering capabilities to remote distributed scientific and engineering simulations, including oil reservoir simulations, computational fluid dynamics and numerical relativity. The DISCOVER server network currently includes deployments at CSM, University of Texas at Austin, and is being expanded to include CACR, California Institute of Technology.

The key contribution of this paper is the design of a middleware substrate that enables interoperability between multiple, geographically distributed and independently administered and managed instances of an entire collaboratory. The primary aim of the substrate was to provide global access to and enable sharing of services/applications across these instances with acceptable performance. We believe that this is the first step towards enabling overall interoperability among collaboratories on the Grid. The DISCOVER middleware is a prototype implementation of this design providing services such as resource discovery, request dispatching, status monitoring, and remote authentication. Interoperability in this implementation is achieved by sharing interfaces defined in CORBA IDL. We are currently working on integrating these IDLs into an open source ORB (such as JacORB [42] or MICO[43]) as standard CORBA services to enable true interoperability. We are also experimenting with interoperability using other protocols and technologies and are investigating XML based protocols such as SOAP and peer-to-peer initiatives such as JXTA [44]. In a related effort we used the middleware substrate to enable interoperability between DISCOVER and the CORBA CoG Kit[3][16] that provides access to Globus [46] Grid services. As a result users can combine and compose the services provided by the two collaboratories. For example, a user can now use services provided by the CORBA CoG Kit to discover available resources on the Grid, to allocate required resources and to run an applications on these resources, and use DISCOVER to connect to and collaboratively monitor, interact with, and steering the application. We are currently evaluating this implementation.

REFERENCES

[1]. 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.

[2]. R. T. Kouzes, J. D. Myers, and W. A. Wulf, “Collaboratories: Doing science on the Internet”, IEEE Computer, Vol.29, No.8, August 1996.

[3] http://www.caip.rutgers.edu/TASSL/Projects/CorbaCoG/

[3]. I. Foster and C. Kesselman, "The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann", San Francisco, 1998.

[4]. The 1st Global Grid Forum, March 2001, Amsterdam, Netherland, http://www.ggf1.nl

[5]. Grid Computing Environments Working Group, Global Grid Forum, http://www.computingportals.org.

[6]. S. Subramanian, G.R. Malan, H.S. Shim, J.H.Lee, P. Knoop, T. Weymouth, F. Jahanian, A. Prakash, and J. Hardin, "The UARC web-based collaboratory: Software architecture and experiences", IEEE Internet Computing, Vol.3, No.2, pp.46-54, 1999. See also: http://intel.si.umich.edu/sparc/.

[7]. C. M. Pancerella, L. A. Rahn, and C. L. Yang, "The diesel combustion collaboratory: Combustion researchers collaborating over the Internet", Proc. of IEEE Conference on High Performance Computing and Networking, Portland, OR, November 1999.

[8]. Access Grid, Argonne National Laboratory, Online at: http://www-fp.mcs.anl.gov/fl/accessgrid/

[9]. H. Casanova and J. Dongarra, "NetSolve: A Network Server for Solving Computational Science Problems", The International Journal of Supercomputer Applications and High Performance Computing, Volume 11, Number 3, pp 212-223, Fall 1997, http://www.cs.utk.edu/netsolve/

[10]. The EMSL Collaboratory. http://www.emsl.pnl.gov:2080/docs/collab/.

[11]. M. Russell, G. Allen, G. Daues, I. Foster, T. Goodale, E. Seidel, J. Novotny, J. Shalf, W. Suen, and G. von Laszewski, "The Astrophysics Simulation Simulation Collaboratory: A Science Portal Enabling Community Software Development". Proceedings of 10th IEEE International Symposium on High Performance Distributed Computing, San Francisco, CA, USA, pp 207-215, IEEE Computer Society Press, August 2001.

[12]. G. Allen, W. Benger, T. Goodale, H.-C. Hege, G. Lanfermann, A. Merzky, T. Radke, E. Seidel, J. Shalf, "The Cactus Code: A Problem Solving Environment for the Grid", http://www.cactuscode.org.

[13]. DISCOVER (Distributed Interactive Steering and Collaborative Visualization EnviRonment), http://www.discoverportal.org.

[14]. S. Kaur, V. Mann, V. Matossian, R. Muralidhar, M. Parashar, "Engineering a Distributed Computational Collaboratory", 34th Hawaii Conference on System Sciences, January 2001

[15]. I. Foster, C. Kesselman, S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", Intl. J. Supercomputing Applications, 2001.

[16]. 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, Spinger LNCS 2242, Denver, (2001), 2-12, http://www.caip.rutgers.edu/TASSL/Papers/corbacog-gcw01.pdf.

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

[18]. HyperText Transfer Protocol (HTTP), http://www.w3.org/Protocols/

[19]. I. Foster, "Internet Computing and the Emerging Grid", Nature, Web Matters, (http://www.nature.com /nature/webmatters/grid/grid.html) Dec., 2000.

[20]. N. H. Kapadia and J. A. B. Fortes," PUNCH: An Architecture for Web-Enabled Wide-Area Network-Computing", Cluster Computing: The Journal of Networks, Software Tools and Applications; special issue on High Performance Distributed Computing. September 1999.

[21]. D. Bhatia, V. Burzevski, M. Camuseva, G. Fox, W. Furmanski, and G. Premchandran, "WebFlow - A Visual Programming Paradigm for Web/Java Based Coarse Grain Distributed Computing", Presented at Workshop on Java for Computational Science and Engineering Workshop, Syracuse University, December 1996.

[22]. E. Akarsu, G. Fox, T. Haupt, A. Kalinichenko, K. Kim, P. Sheethaalnath, and C. H. Youn, "Using Gateway System to Provide a Desktop Access to High Performance Computational Resources", 8th IEEE International Symposium on High Performance Distributed Computing (HPDC-8), Redondo Beach, California, August, 1999.

[23]. M. Thomas, S. Mock, and J. Boisseau, "Development of Web Toolkits for Computational Science Portals: The NPACI HotPage", The 9th IEEE International Symposium on High Performance Distributed Computing (HPDC 2000), Pittsburgh, Aug. 1-4, 2000, https://hotpage.npaci.edu/

[24]. SDSC GridPort Toolkit - http://gridport.npaci.edu/

[25]. Grid Portal Development Kit (GPDK), http://www-itg.lbl.gov/grid/projects/GPDK/

[26]. G. von Laszewski, I. Foster, J. Gawor, P. Lane, N. Rehn, and M. Russell, "Designing Grid-based Problem Solving Environments and Portals", Proceedings of the 34th Hawaii International Conference on System Sciences, January 2001, http://www.globus.org/cog (Commodity Grid Toolkits (CoG))

[27]. R. Buyya, D. Abramson, J. Giddy, "Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid", The 4th International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2000), Beijing, China. IEEE Computer Society Press, USA, 2000, http://www.csse.monash.edu.au/~rajkumar/ecogrid/

[28]. JiPANG - A Jini based Computing Portal System, http://ninf.is.titech.ac.jp/jipang/

[29]. S. Matsuoka and H. Casanova, "Network-Enabled Server Systems and the Computational Grid", White Paper, http://www.eece.unm.edu/~apm/WhitePapers/GF4-WG3-NES-whitepaper-draft-000705.pdf

[30]. G. R. Malan, F. Jahanian, and S. Subramanian, "Salamander: A Push-based Distribution Substrate for Internet Applications", Proceedings of the USENIX Symposium on Internet Technologies and Systems, December 1997, Monterey, CA.

[31]. Internet Performance Measurement and Analysis (IPMA) project homepage, http://nic.merit.edu/ipma/

[32]. G. C. Fox, "Portals for Web Based Education and Computational Science", http://new-npac.csit.fsu.edu/users/fox/documents/generalportalmay00/erdcportal.html

[33]. W. Allcock, I. Foster, S, Tuecke, A. Chervenak and C. Kesselman, "Protocols and Services for Distributed Data-Intensive Science", to be published in ACAT2000 proceedings.

[34]. The Collaboratory Interoperability Framework Project (CIF), http://www-itg.lbl.gov/CIF/

[35]. S. Matsuoka, H. Nakada, M. Sato and S. Sekiguchi, "Design issues of Network Enabled Server Systems for the Grid", White Paper, http://www.eece.unm.edu/~apm/WhitePapers/satoshi.pdf

[36]. Java Servlet API Specification, http://java.sun.com/products/servlet/2.2/.

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

[38]. CORBA Trader Service Specification, ftp://ftp.omg.org/pub/docs/formal/97-07-26.pdf.

[39]. Apache Web Server, http://httpd.apache.org

[40]. Apache Jserv Servlet Engine, http://java.apache.org

[41]. Visibroker for Java (CORBA ORB), http://www.borland.com/visibroker/

[42]. JacORB, http://www.jacorb.org

[43]. MICO, http://www.mico.org

[44]. Project JXTA, http://www.jxta.org

[45]. M. Baker, R. Buyya and D. Laforenza, "The Grid: A Survey on Global Efforts in Grid Computing", ACM Journal of Computing Surveys, 2000 (submitted).

[46]. I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit", Intl J. Supercomputer Applications, 11(2): 115-128, 1997.

[47]. A. Grimshaw, A. Ferrari, F. Knabe and M. Humphrey, "Legion: An Operating System for Wide-Area Computing", IEEE Computer, 32:5, May 1999: 29-37.