MIDDLEWARE ARCHITECTURE FOR INTEGRATED COMPUTATIONAL COLLABORATORIES

by

VIJAY MANN

A thesis submitted to the

Graduate School-New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Master of Science

Graduate Program in Electrical and Computer Engineering

written under the direction of

Prof. Manish Parashar

and approved by

_____

_____

_____

New Brunswick, New Jersey

October, 2001

ABSTRACT OF THE THESIS

Middleware Architecture for Integrated Computational Collaboratories

by Vijay Mann

Thesis Director: Professor Manish Parashar

A Collaboratory is defined as a place where scientists and researchers work together to solve complex interdisciplinary problems, despite geographic and organizational boundaries. Computational collaboratories provide uniform (collaborative) access to computational resources, services and/or applications. 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 expand the resources available to researchers, enable multidisciplinary collaborations and problem solving, increase the efficiency of research, and accelerate the dissemination of knowledge.

While combining these systems can lead to truly collaborative, multi-disciplinary and multi-institutional problem solving, integrating these "focused" collaboratories presents significant challenges. This is because each of these collaboratories has a unique architecture and implementation, and builds on different enabling technologies. Key among these challenges is the design and development of robust middleware support that addresses scalability, service discovery, security and access control, and interaction and collaboration management for consistent access. Such a middleware 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 thesis investigates the requirements for achieving interoperability among collaboratories operating on the Grid. It then presents the design of a middleware substrate that addresses interoperability, and a prototype implementation of this middleware substrate, to enable a peer-to-peer integration of and global collaborative access to multiple, geographically distributed instances of the DISCOVER computational collaboratory. DISCOVER provides collaborative access to high-performance parallel and distributed applications for interaction and steering using a web-based portal. 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. A retrospective evaluation of the design and an experimental evaluation of the prototype middleware substrate are also presented.

Acknowledgements

I am grateful to my advisor Prof. Manish Parashar for his invaluable guidance, immense patience, encouragement and support throughout my stay at Rutgers. I am thankful to Prof. Michael Hsiao and Prof. Ivan Marsic for their valuable advice and pertinent suggestions regarding my thesis. Thanks are due to all the current and former team members of the DISCOVER project for their invaluable suggestions. I would also like to thank the CAIP support staff for their prompt and detailed responses to my queries and for the excellent facilities that they provide in the various laboratories at CAIP and in particular at The Applied Software Systems Laboratory (TASSL). I acknowledge the support and love of all my friends at TASSL and at Rutgers University for making my studies at Rutgers an enjoyable phase of my life. Finally, this work wouldn't have been possible without the love, support and encouragement from my family members here in the United States and back home in India.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Objective

The objectives of this thesis are to:

- Investigate the requirements for achieving interoperability among collaboratories operating on the computational Grid.
- Develop a middleware design for collaboratories operating on the Grid, which meets the above requirements.
- Design, implement and evaluate a prototype middleware architecture, which integrates multiple, geographically distributed instances of the DISCOVER computational collaboratory to enable collaborative sharing and steering of applications across such instances.

1.2 Background

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 and/or applications such as analytical tools, instruments and raw data, summaries and analyses for multidisciplinary research, archival information and tools for synchronous and asynchronous collaboration. These systems expand the resources available to researchers, enable multidisciplinary collaborations and problem solving, increase the efficiency of research, and accelerate the dissemination of knowledge.

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 following:

a) The Upper Atmospheric Research Collaboratory (UARC)[6][7] provides a virtual shared workspace in which a geographically dispersed community of space scientists perform real time experiments at remote facilities, in locations such as Greenland, Puerto Rico, and Alaska.

b) The Diesel Combustion Collaboratory (DCC) [8][9] is a problem solving environment (PSE) for combustion research providing tools such as a distributed execution management system for running combustion models on widely distributed computers (including supercomputers), web accessible data archiving capabilities; electronic notebooks and shared workspaces; visualization of combustion data; and video conferencing and data-conferencing tools.

c) Access Grid[10] is an ensemble of resources that can be used to support human interaction across the grid, and consists of multimedia displays, presentation and interactions environments, interfaces to grid middleware, and interfaces to visualization environments.

d) Netsolve [11] is a client-server system that enables users to solve complex scientific problems remotely and allows users to access both hardware and software computational resources distributed across a network.

e) EMSL [12] is a symmetric collaboration between computer scientists, domain scientists (physical and biological sciences), and sociologists and relies on the development of new communications technologies - shared computer displays, electronic notebooks, virtual reality collaboration spaces - and an integration of these technologies with current videoconferencing and email capabilities.

f) The Astrophysics Simulation Collaboratory [13] involves a community of scientists, researchers, and developers who wish to collaborate on the development of scientific codes for the astrophysics community at large. It builds on Cactus [14], which is an open source problem-solving environment designed for scientists and engineers in the field of numerical relativity and astrophysics.

g) DISCOVER [1][15][16] provides a virtual shared workspace for scientists and researchers to steer and collaboratively interact with large parallel and distributed applications in diverse fields such as oil reservoir simulations, seismic whole-earth simulations, computational fluid dynamics, and numerical relativity.

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 simulations into global modalities for research and instruction. However, seamlessly integrating these systems presents many challenges.

1.3 Problem Statement

The collaboratories listed in section 1.2 provide specialized services to their user community for their specific application domain. There is hardly any interaction across such collaboratories and as a result, these services are restricted to their specific user domains. Combining these "focused" collaboratories and allowing them to interoperate presents many advantages. The services provided by the different collaboratories can be reused reducing duplication of effort. At a higher level, the domain specific services provided by the collaboratories can be composed and combined 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. The design of such systems has rarely focused on the issues of interoperability and extensibility. Although, some systems like Ninf [34][40][41] and NetSolve [11] have been able to interoperate with each other through a joint collaborative effort by their respective development teams, it is still limited to bilateral sharing, as opposed to global sharing of resources recommended by the Grid architecture. Such efforts have further identified the significance of interoperability and the need for a general solution for interoperability among collaboratories. Integration of such systems by making them interoperate with each other will definitely provide scientists and researchers with an interesting opportunity for multi-disciplinary research.

While combining these systems can lead to truly collaborative, multi-disciplinary and multi-institutional problem solving, integrating these "focused" collaboratories presents significant challenges. Key among these challenges is the design and development of high-level services as part of a robust middleware support that addresses interoperability, scalability, service discovery, security and access control, and interaction and collaboration management for consistent access. Such a middleware 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.

1.4 Overview

This thesis first investigates the requirements for achieving integration and interoperability among collaboratories on the Grid and discusses how the Grid architecture proposed by Foster et al. in [17] can be naturally extended to define an architecture for collaboratories on the Grid. It

then presents the design of a middleware substrate within this architecture that addresses the interoperability issues discussed above.

A prototype implementation of a middleware substrate, based on the proposed design, is then presented. This prototype enables a peer-to-peer integration of and global collaborative web-based access to multiple, distributed instances of the DISCOVER computational collaboratory. DISCOVER provides collaborative access to high-performance parallel and distributed applications for interaction and steering using web-based portals [1][15][16]. The key 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 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 design and implementation of the DISCOVER middleware substrate builds on existing web servers and leverages commodity technologies and protocols such as CORBA [18] and HTTP [19]. Its goal is to enable rapid deployment, ubiquitous and pervasive access, and easy integration with 3rd party services, while evaluating the viability of these technologies for advanced Grid applications.

The overall aim of Grid computing is to enable collaborative and coordinated problem solving in dynamic, multi-institutional virtual organizations and it focuses on large-scale resource sharing, innovative applications, and high performance computing [17]. The middleware substrate presented in this thesis addresses one aspect of this general problem by providing global collaborative access to grid applications and services. The middleware substrate presents a simple implementation of a collaboration service, which handles collaboration among groups spanning multiple domains or servers. The fact that the clients on all servers communicate over HTTP, which is a stateless request-response protocol and thus not the ideal protocol for collaboration,

further illustrates the significance of this service. It also reflects on the various design issues that come up while designing such a service for web clients (communicating over HTTP).

An experimental evaluation of the middleware substrate is also presented. It measures latency overheads for indirect access over CORBA/IIOP as compared to direct access over HTTP (both on a local area network as well as on a wide area network), latency overheads for multiple simultaneous clients and memory overheads for different configurations of the middleware substrate. The latency measurement results over a wide area network validate our design and justify the use of IIOP as the inter-server communication protocol.

1.5 Contributions

This thesis makes the following contributions:

a) Formulation of the requirements for achieving interoperability among collaboratories operating on the Grid.

b) Design and implementation of a middleware substrate that enables a peer-to-peer integration of and global collaborative web-based access to multiple, distributed instances of the DISCOVER computational collaboratory. It uses commodity technologies and protocols such as CORBA [18] and HTTP [19] for middleware development to enable rapid deployment, ubiquitous and pervasive access, and easy integration with 3rd party services, while evaluating the viability of these technologies for advanced Grid applications.

c) Implementation of a collaboration service, which handles collaboration among groups spanning multiple domains or servers and reflects on the various design issues that come up while designing such a service for web clients (communicating over HTTP).

d) An experimental evaluation of the middleware substrate comparing its performance on a wide area network to that on a local area network.

1.6 Organization

This thesis is organized in 7 chapters. Chapter 2 provides some background and discusses related work on interoperability among collaboratories. It also discusses recent efforts in developing technologies that enable interoperability.

Chapter 3 outlines the requirements for achieving interoperability among collaboratories on the Grid. It also presents a hierarchical architecture for collaboratories on the Grid and a middleware design for such collaboratories.

Chapter 4 introduces the DISCOVER web based computational collaboratory for interaction and steering, and describes the design, implementation, and operation of its interaction and collaboration server. This chapter also introduces the middleware substrate for peer-to-peer integration of a network of DISCOVER servers to provide global collaborative access to remote applications.

Chapter 5 describes the implementation and operation of the DISCOVER middleware substrate.

Chapter 6 presents an experimental evaluation of the middleware substrate and a retrospective evaluation of the design and discusses its advantages and disadvantages. This chapter also presents an evaluation of commodity distributed technologies and protocols and their ability to support Grid applications, and briefly discusses open issues and challenges.

Chapter 7 presents some conclusions and outlines current and future work.

Chapter 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] such as the Upper Atmospheric Research Collaboratory (UARC)[6][7], the Diesel Combustion Collaboratory (DCC) [8][9], Access Grid [10], Netsolve [11], EMSL [12], the Astrophysics Simulation Collaboratory (ASC) [13][14], Punch [21][22], WebFlow [23], Gateway [24], HotPage/GridPort [25][26][27], GPDK [28], Commodity CoG Kits [29][30][31], Nimrod-G [32], JiPang [33], and DISCOVER [1][15][16]. These systems provide specialized services to their user communities and address different issues in wide area resource sharing and the overall Grid computing problem [17][20]. 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. Some of these systems are briefly discussed below.

Punch (Purdue University Network Computing Hubs) provides the user with an illusion of a wide area computer that appears as a computing portal. PUNCH provides a computing portal for allocating resources, deploying and running applications, and provides a desktop view of that application. Using Punch, users can select an application; specify the location where the data is stored, and then run the application using that data. Punch first verifies if the user is authorized to run the selected application, then uses its active yellow pages service to locate an appropriate machine for the run and its virtual file system to mount the application and data disks on the

selected machine. Finally it invokes the application on the selected machine and routes the display to the user's browser via a remote display management technology such as VNC.

The NPACI HotPage and the Grid Port toolkit provide secure and customized access to grid services through web portals. HotPage is a user portal that attempts to simplify access to HPC resources distributed across member organizations, and allows them to be viewed either as an integrated Grid system or as individual machines. GridPort generalizes the HotPage infrastructure and develops a reusable portal toolkit. The two key components of GridPort are the web portal services and the application APIs. The Web portal module runs on a Web server and provides secure connectivity to the grid. The APIs provide a Web interface that enables the development of customized science portals by end users. The GridPort modules are based on commodity Internet and Web technologies as well as existing grid services and applications.

The Astrophysics Simulation Collaboratory (ASC Portal) provides an application specific PSE for composing, configuring and deploying astrophysical simulations on the Grid. ASC uses a N-tier application model and builds on commodity technologies such as HTTPS, servlets, and RDBMS. The ASC server architecture leverages ongoing efforts aimed at providing high-level access to grid services such as the Java CoG [29], and the Grid Portal Development Toolkit (GPDK). The Java CoG kit is part of the Commodity Grid (CoG) project, which is working to overcome the difficulties of accessing advanced grid services, such as authentication, remote access to computers, resource management, and directory services by defining mappings and interfaces between the Grid and commodity frameworks (e.g. Java, CORBA, Python) that are familiar to PSE developers.

WebFlow provides a framework for publishing and reusing computational modules on the web, so that end users, using a web browser, can visually compose distributed applications using these modules. The overall WebFlow architecture is very similar to DISCOVER – however, it addresses issues in composing, configuring and deploying scientific applications on the Grid, whereas DISCOVER is primarily aimed at computational steering and collaborative interactive

visualization of large parallel and distributed scientific simulations. The WebFlow middle tier also uses a network of Java enhanced web servers (although it does not exploit the peer-to-peer nature of the servers). Furthermore, WebFlow, like DISCOVER, uses high level distributed technologies like servlets and CORBA, and provides similar advantages such as portability and extensibility.

The Salamander middleware substrate [35][36][37], which is used in the UARC and the IPMA (Internet Performance Measurement and Analysis) project [38], is a wide area network data dissemination substrate. The Salamander substrate provides support for both web casting and groupware applications by providing virtual distribution channels through its channel subscription service. The channel subscription service provides an abstraction for the distribution of data from publishers to subscribers with both anonymous and negotiated push techniques. In contrast, DISCOVER uses a poll and pull technique. Clients in DISCOVER poll their nearest server for new data, and fetch it if it is available. Poll and pull is a natural choice for DISCOVER since the clients use HTTP which is a request response protocol. Salamander supports a limited interaction and steering capability through its negotiated push technology. The DISCOVER computational collaboratory provides a richer control interface allowing users to collaboratively monitor and control overall application execution, to access, interact with and steer individual computation objects, to manage object dynamics and distribution, and to schedule automated periodic interactions.

2.2 Motivation for Interoperable Collaboratories

There are several compelling reasons for allowing multiple types of collaboratories to co-exist and interoperate on the Grid [34]. 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, each system may have a different requirement on how its clients should access its services. 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, the services provided by the different systems can be reused reducing duplication of effort. For example the ASC [13] reuses the authentication services provided by the Java CoG [29] rather than re-implement them. At a higher level, the domain specific services provided by the collaboratories can be composed and combined 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 [35][36][37] (used in the UARC and the IPMA project [38]). Building each system to provide all required capabilities will not only lead to duplication but is rapidly ceasing to be a viable option. However, most of these are standalone systems with customized architectures, and 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 [34]. 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 Related Work on Interoperable Collaboratories

Although interoperability has been identified as a central issue for Grid based systems in previous work [3][17][39][40], there has been limited progress towards achieving this goal. This is particularly true in the case of computational collaboratories, which typically have a monolithic design aimed at serving the domain specific needs of their user community. Although, there have been efforts aimed at bilateral sharing and interoperability such as those between Ninf [34][40][41] and NetSolve [11], these have been made possible because of joint development efforts by their respective development teams. These efforts have further identified the significance of interoperability and the need for having a general solution for interoperability.

The Collaboratory Interoperability Framework Project [39] proposes a common communication API that can be used by collaboratory developers to build tools for collaboration like videoconferencing, text based collaboration through chat, whiteboard and electronic notebooks. 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. The API supports a wide range of protocols like Unreliable Unordered Multicast (e.g IP Multicast), Unreliable Ordered Multicast (IP Multicast with out of order packets filtered), Unreliable Datagram Protocol (UDP), Transmission Control Protocol (TCP), Reliable Ordered Multicast (e.g the Totem protocol) and Reliable Source Ordered Multicast (e.g the XTP protocol). However, this approach is too low-level, which means that collaboratories will continue building customized services on top of the common communication API.

Another approach for enabling interoperability has been presented in [42]. This apporach characterizes portals as web based interfaces to applications. In particular it focuses on portals for computational science and web based education. These portals will be based on technologies

developed for areas such as e-commerce and the large Enterprise Information Portal market. 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. The use of two interfaces separates the user and system view and insulates both the user interface and repository resources from the changing server infrastructure.

2.4 Enabling Technologies for Interoperability

Recent efforts in peer-to-peer (P2P) computing and enterprise computing address problems and issues, which are equally valid in a Grid environment.

2.4.1 Peer-to-Peer Computing

Collaboratories interoperating with each other on the Grid are essentially peers interacting in a P2P network. Peer-to-peer computing, as implemented in Internet communication and file sharing tools like Napster [43], Gnutella [44], and Freenet [45], and Internet computing as implemented by systems such as SETI@home [46], Parabon [47], and Entropia [48], are examples of more general sharing modalities and computational structures beyond traditional client server systems. These characterize virtual organizations where information and resource sharing can take place among any subset of participants. These technologies and systems present a radical paradigm shift from client-server systems. These systems have so far focused entirely on vertically integrated solutions, rather than seeking to define common protocols that would allow for a shared infrastructure and interoperability. Also, the forms of sharing targeted by various applications are quite limited; for example, file sharing with no access control, computational sharing with a centralized server or multi-user text based collaboration.

Project JXTA [49] from Sun Microsystems, is a network programming and computing platform designed to solve a number of problems in modern distributed computing, especially in the area of peer-to-peer computing. The objective of the project is to build interoperable, platform independent, and ubiquitous peer-to-peer systems. JXTA technology is still evolving and will soon be open-sourced and co-developed by many contributors. At the highest abstraction level JXTA technology is a set of protocols and currently defines protocols such as Peer Discovery Protocol, Peer Resolver Protocol, Peer Information Protocol, Peer Membership Protocol, Pipe Binding Protocol and Endpoint Routing Protocol. These protocols use XML-encoded messages. JXTA stays away from APIs and remains independent of programming languages. More importantly, JXTA is designed to be also independent of transport protocols. It can be implemented on top of TCP/IP, HTTP, Bluetooth, HomePNA and many other protocols. Project JXTA holds a lot of promise and it will be interesting to see how it evolves. Many of the protocols and the objectives defined by JXTA, are equally valid and significant in Grid environments.

A related proposal from Intel [50] presents an approach for peer-to-peer computing for enterprise systems, where jobs are "split" into byte-sized tasks for individual PCs. Intel's peer to peer computing proposals categorize its use in four main categories:

Collaboration – empowering individuals and teams to create and administer real-time and off-line collaboration areas in a variety of ways, whether administered, unadministered, across the Internet, or behind the firewall.

Edge services - In essence, edge services move data closer to the point at which it is actually consumed acting as a network caching mechanism.

Distributed computing and resources - Using a network of computers, peer-to-peer technology can use idle CPU MIPS and disk space, allowing businesses to distribute large computational jobs across multiple computers.

Intelligent agents - Agents reside on peer computers and communicate various kinds of information back and forth. Agents may also initiate tasks on behalf of other peer systems.

These categories pretty much summarize the use of peer-to-peer technology. The DISCOVER peer to peer server architecture has features representative of most of these categories: it allows collaboration among users and servers, it allows clients to connect to their nearest server and access remote resources through it, the servers maintain references to remote resources which act as intelligent agents at the remote server, etc. Our current implementation however, does not try to use idle cycles on peer servers.

2.4.2 Enterprise Computing Technologies

Enterprise computing technologies such as Universal Description Discovery and Integration (UDDI) [51] and Microsoft's .NET [52] are related efforts aimed at supporting discovery of web services and interactions between them. UDDI specifications define a way to publish and discover information about Web Services. The term "web service" describes specific business functionality exposed by a company, usually through an Internet connection, to enable another company or software program to use the service. UDDI registries are used to promote and discover these distributed Web Services. The UDDI approach relies upon a distributed registry of businesses with their service descriptions implemented in a common XML format. Another related effort is WSDL (Web services Description Language) [53], which is a general purpose XML language for describing the interface, protocol bindings and the deployment details of network services. WSDL complements the UDDI standard by providing a uniform way of describing the abstract interface and protocol bindings of arbitrary network services.

Chapter 3

Building Interoperable Collaboratories on the Grid

3.1 Approaches to Interoperability

As motivated earlier, interoperability among computational collaboratories can be achieved through a middleware layer on top of the Grid. Such a middleware can be 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 the same functionality in customized ways, resulting in non-reusability and no interoperability. Following this approach, there are three distinct ways to implement the higher-level services so as to ensure some level of interoperability – same implementation everywhere, same interface and/or API everywhere, and same protocol everywhere.

Shared Implementation: The first approach is to have these higher level services built into one metacomputing toolkit and have each system use the same metacomputing toolkit. A real world example of this approach is the use of different commercial instant messaging software available from Yahoo, Microsoft, etc. Users using these systems can only share messages with other users who have the same software. The scalability, extensibility and the level of interoperability of such an approach for wide area resource sharing leaves much to be desired. It is not reasonable to assume that everyone will use the same metacomputing toolkit to implement these services.

Shared Interfaces and APIs: The second approach is to have each system publish a set of APIs and interfaces for others to use. This is the approach taken by CORBA applications to achieve interoperability. The API is specified through the IDL (Interface Definition Language)

and the IDLs are shared by all systems. This is a feasible solution and it will allow interoperability between limited numbers of systems, but it requires all systems to have the same IDL or at least know about its contents. Each system should also have an implementation that conforms to the API specified in the IDL. Hence, it will not provide truly global and seamless sharing of resources across systems 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 [17], 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.

The importance of common protocols in Grid environments has been emphasized for a long time [17] and APIs and SDKs have been identified as auxiliary to protocols because without standard protocols, interoperability can only be achieved at the API level either by using a single implementation everywhere or by using IDL based approaches where every implementation knows the interface supported by every other implementation. However, the process of standardization is a time consuming one and many systems might prefer a quick solution. Furthermore, any change in a standard protocol to make it adaptable in a dynamic environment has to go through the same lengthy process.

True (protocol-based) interoperability can be achieved using CORBA by leveraging the fact that CORBA uses the IIOP protocol for all communication. The service to be shared can be built into the CORBA ORB as a standard CORBA service. As the service is now a standard CORBA service, it is easier for all other systems to know its API. This is analogous to the sockets API supported by various implementations of TCP/IP.

3.2 Architecture of a Grid-based Collaboratory

An overall architecture for the Grid has been defined using an hourglass model [17]. The services defined at the *Application layer* and the *Collective layer* at the top of the hourglass are used to construct a wide range of global services and application-specific behaviors. The neck of the hourglass consists of the *Resource* and the *Connectivity* layer, which define protocols for sharing of individual resources. Protocols at these layers should be designed so that they can be implemented on top of a diverse range of resources types, defined at the *Fabric* layer at the base of the hourglass. The hourglass model emphasizes that the number of protocols at the neck of the hourglass (the *Resource* and the *Connectivity* layers) should be small, so as to encourage widespread and easy deployment.

The hourglass Grid architecture model can be naturally extended to build interoperable collaboratories on the Grid by formulating their requirements in terms of high-level *infrastructure services* and building these services using the standard Grid protocols and services identified in [17]. Figure 1 shows the hierarchical architecture for a collaboratory on the Grid. It consists of user level services, infrastructure services and infrastructure protocols. The user level services are the customized set of services that each system provides to its users, and are specific to each collaboratory. However, each of these user services can be formulated as a combination of one or more underlying *infrastructure services*. These services build on underlying *infrastructure protocols* and are typically implemented as daemon processes running at the middle tier. It is these infrastructure services that interoperate with each other on the Grid. Each service should build on one of the infrastructure protocols and more than one service can use the same protocol. The user services and the infrastructure services in this architecture can be thought of as specific instances of the *Application* and the *Collective layer* at the top of the hourglass. The Infrastructure protocol layers form the neck of the hourglass.

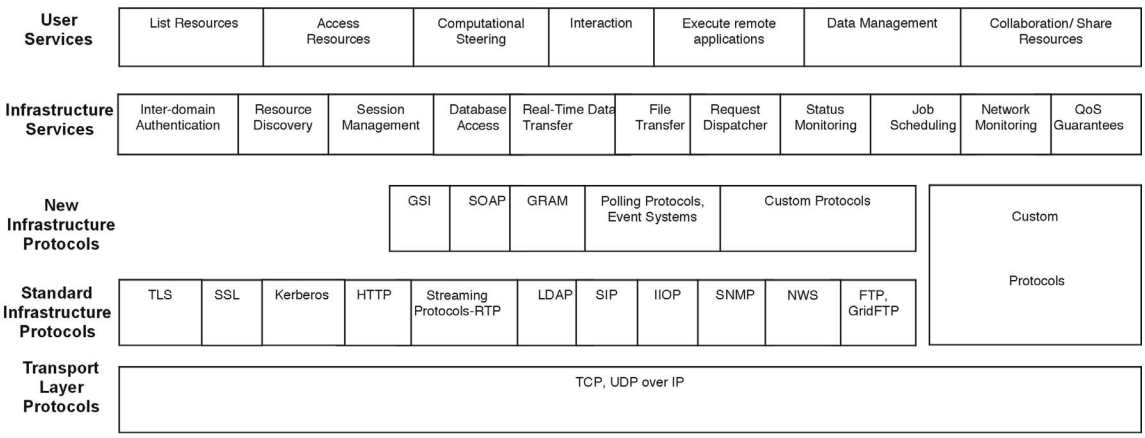

Figure 1. Hierarchical architecture of a collaboratory on the Grid

Table 1 and Table 2 illustrate the application of this architecture to two specific collaboratories – a collaboratory for interaction and computational steering (e.g. DISCOVER) and a collaboratory for executing applications remotely (e.g. PUNCH).

User Services *(Application Layer)*	List Available Applications	Access an Active application	Computational Steering	Data Management	Collaboration on output data
Infrastructure Services *(Collective Layer)*	Inter-domain Auth. + Resource Discovery	Inter-domain Auth. + Resource Discovery + Request Dispatcher + Status Monitoring	Request Dispatcher + Real-Time Data Transfer + Status Monitoring	Request Dispatcher + Database Access + File Transfer	Request Dispatcher + Real time Data Transfer + Status Monitoring
Infrastructure Protocols *(Resource & Connectivity Layers)*	Layer 1 GSI, GIS, MDS, GRAM, SOAP, GridFTP, NWS, …				
	Layer 2 SSL, LDAP, HTTP, FTP,RTP, SNMP, …				

Table 1. A Collaboratory for interaction and computational steering (e.g. DISCOVER)

User Services *(Application Layer)*	List available computing resources	Access to a computing resource for job submission	Deploying and executing an application	Data Management (Optiponal)	Collaboration on output data (Optional)
Infrastructure Services *(Collective Layer)*	Inter-domain Auth. + Resource Discovery + Network Monitoring	Inter-domain Auth. + Resource Discovery + Request Dispatcher + Status Monitoring	Request Dispatcher + File Transfer + Status Monitoring	Request Dispatcher + Database Access + File Transfer	Request Dispatcher + Real time Data Transfer + Status Monitoring
Infrastructure Protocols *(Resource & Connectivity Layers)*	Layer 1 GSI, GIS, MDS, GRAM, SOAP, GridFTP, NWS, …				
	Layer 2 SSL, LDAP, HTTP, FTP,RTP, SNMP, …				

Table 2. A Collaboratory for launching applications remotely (e.g. PUNCH)

3.2.1 Interoperability through available Protocols

Since the protocols defined for interoperability should be small in number, we should not strive for new protocols for each of the above services. Generic request response protocols like SOAP which build on HTTP hold a lot of promise and should be experimented with. However, these HTTP based protocols largely use XML for message encoding and are therefore much slower because of the associated parsing time. For efficient and fast transfer of real time large data sets and of files, protocols like RTP, multicast, and GridFTP should be considered. Peer-to-peer (P2P) initiatives address similar issues and problems as collaboratories interoperating with each other on the Grid are similar to peers interacting with each other in a P2P network. However, the state of the art in P2P technologies is still in its infancy and most of the solutions are vertical solutions aimed at sharing of specific file formats rather than generic wide area resource sharing. The peer-to-peer initiative JXTA is a promising technology as many of the protocols and objectives defined by JXTA, are equally valid and significant in Grid environments.

IIOP (Internet Inter-ORB Protocol), on the other hand, is perhaps the only protocol that defines interoperability between peers. CORBA IDL based approaches (which use IIOP for all

communication) will achieve interoperability but each implementation should possess this IDL or at least know about its contents (this is true in case of DII - Dynamic Invocation Interface, where one has to query the interface for supported methods and make a call to the appropriate method, which requires a previous notion of the functionality, that each method is supposed to provide). A more generic approach will be to integrate these IDLs into the ORB itself as CORBA services similar to the existing CORBA services like the Naming service, the Trading service, the Event service, etc to form a Grid-base ORB. Open source ORBs can be used for this purpose.

The protocol description above is not exhaustive. This is just an effort to extract the requirements for obtaining interoperability among the various systems in a Grid environment. We have tried to list some of the protocols that are available today which can be used in the Grid environment (some of them like the GSI protocols, the GRAM protocol and the GridFTP protocol have been designed specifically for systems on the Grid) and the custom protocols that should be designed on top of existing application level protocols like HTTP, SIP, IIOP, RTP, etc and on top of TCP/IP directly for building the infrastructure services.

3.3 Middleware Design for Grid-based Collaboratories

3.3.1 DISCOVER Middleware Approach

The overall goal of the middleware substrate presented here (and the related CORBA CoG effort [31]) is to define interfaces and mechanisms for integrating the services provided by domain specific collaboratories. DISCOVER middleware uses the peer-to-peer computing concepts to define the architecture for integrating computational collaboratories and thereby achieving interoperability.

The DISCOVER design philosophy and architecture presents a hybrid approach as compared to true peer-to-peer and client-server systems. While the middleware design has a client server architecture from the users' point of view, the middle tier of servers has a peer-to-peer architecture with each server functioning partially as a client and partially as a server for all the

other servers. This approach reduces the performance requirements of the server, and allows for more secure and better-managed peers as compared to a system comprising only of peers. This is because security and manageability are still open issues in true peer-to-peer systems, whereas one of the reasons for the success of client server systems is the security and manageability associated with having centralized servers. The DISCOVER hybrid approach of having peer-to-peer servers drastically reduces the number of peers in the system and restricts the security and manageability concerns to the middle tier of servers. Furthermore, this approach allows thin clients, as no assumptions are made about the computing capabilities of the clients or the bandwidth available to them.

DISCOVER middleware design builds on existing web servers and leverages commodity technologies and protocols such as CORBA [18] and HTTP [19]. Its goal is to enable rapid deployment, ubiquitous and pervasive access, and easy integration with 3rd party services, while evaluating the viability of these technologies for advanced Grid applications. Interoperability is achieved through a set of remote method invocations defined in IDLs, which are shared across systems. True interoperability can be obtained by integrating these IDLs into the ORB as standard CORBA services. The portalML and resourceML interfaces, mentioned in section 2.3 for interoperable web portals, are similar in nature to the interfaces specified by the DISCOVER middleware substrate. However, the DISCOVER interfaces, instead of defining a user view of a portal like portalML, define a system view of a server which can be used by other servers in the system to access its services. DISCOVER middleware substrate uses an interface to describe a particular application or service at a server, which is similar to the resourceML. DISCOVER is based on a pool of services model that makes use of this interface to provide services and applications as a pool of distributed commodities. This service can be accessed by any other server in the system. DISCOVER interfaces are defined in the CORBA IDL (Interface Definition Language) instead of XML.

The servers in this model are lightweight, portable and easily deployable and manageable instead of being heavy weight and difficult to manage. A server should be deployed anywhere where there is a growing community of users, much like a HTTP Proxy server. This allows each collaboratory to support a large user community as well as a large number of applications without being overloaded and suffering degradation in performance.

3.3.2 DISCOVER Middleware Design

A schematic overview of the design is presented in Figure 2. It has a 3-tier architecture consisting of collaborative client portals at the front end, the computational resource, services or applications at the backend, and the server(s) in the middle. In order to enable ubiquitous web-based access, clients are kept as simple as possible. The middle-tier has the responsibility for providing controlled access to the backend, interacting with peer servers, providing a "repository of services" view to the client, 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, and network-monitoring tools. The backend services (resource, service or application) may be specific to a server or may form a pool of services. In the former case, direct access to the service is restricted to the local server, typically due to security, scalability or compatibility constraints. This is typically true of most scientific resources, services and applications. In this case, the local server advertises the service and its interface, through which clients and peer servers can discover and access the service. All communication is done through the local server. The server can 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 as well as the backend services are accessed using standard distributed object technologies such as CORBA, RMI/RMI-IIOP, DCOM, etc.

The various infrastructure services outlined in the previous section are implemented as daemon processes or threads either as a part of the web server (as server side extensions, CGI scripts or Java servlets) or as a part of the pool of services. Their functionality is invoked through remote method calls defined in the IDL, which is shared across all systems or can be downloaded at runtime (in case of Dynamic Invocation Interface or DII). The client-server interaction is achieved through a custom protocol built on top of HTTP.

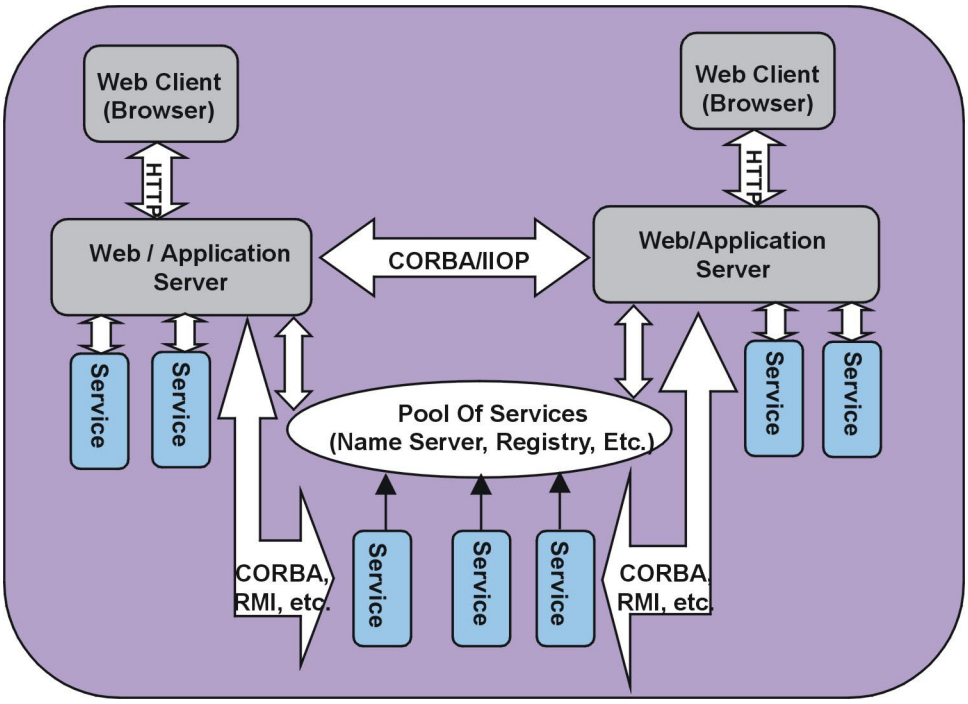

Figure 2. Middleware architecture for integrating web-based computational collaboratories

The middleware architecture defines two levels of interfaces and interactions for each server. The first level interfaces provide a means for peer servers to authenticate with the server and query it for active services, applications and users. The second level interfaces define interactions with the active services and/or applications at the server, and enable a peer server or client to authenticate, interact with and invoke the service. For example, in the case of an interactive application it would provide methods for monitoring application state, requesting/releasing locks for steering accesses, querying and/or changing its parameters, etc. If a

server provides only a single instance of an application or a service, e.g. a server providing access to grid services using Java/CORBA CoG and GPDSK, only the second level interfaces would be required.

The DISCOVER middleware substrate presented in the following sections is a prototype implementation of these interfaces to enable global discovery of, and provide access to, distributed applications for interaction and steering.

Chapter 4

DISCOVER: A Computational Collaboratory for Interaction and Steering

This chapter introduces the DISCOVER computational collaboratory and gives an overview of the design and operation of its Interaction and Collaboration server. It also introduces the DISCOVER middleware substrate that enables a peer-to-peer integration of multiple DISCOVER Interaction and Collaboration servers to provide global collaborative web-based access to multiple, distributed instances of the DISCOVER computational collaboratory.

DISCOVER is a virtual, interactive computational collaboratory that enables geographically distributed scientists and engineers to collaboratively monitor, and control (new and existing) high performance parallel/distributed applications. Its primary goal is to bring large (remote) distributed simulations to the scientists'/engineers' desktop by providing collaborative web-based portals for interrogation, interaction and steering. DISCOVER architecture (see Figure 3) 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 can connect to a server at any time using a web browser to receive information about active applications. Furthermore, they can form or join collaboration groups and can (collaboratively) interact with one or more applications based on their capabilities. The middle tier consists of the Interaction and Collaboration server, which extends a commodity web-server 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 mechanism 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[1] 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 DISCOVER can be found in [16].

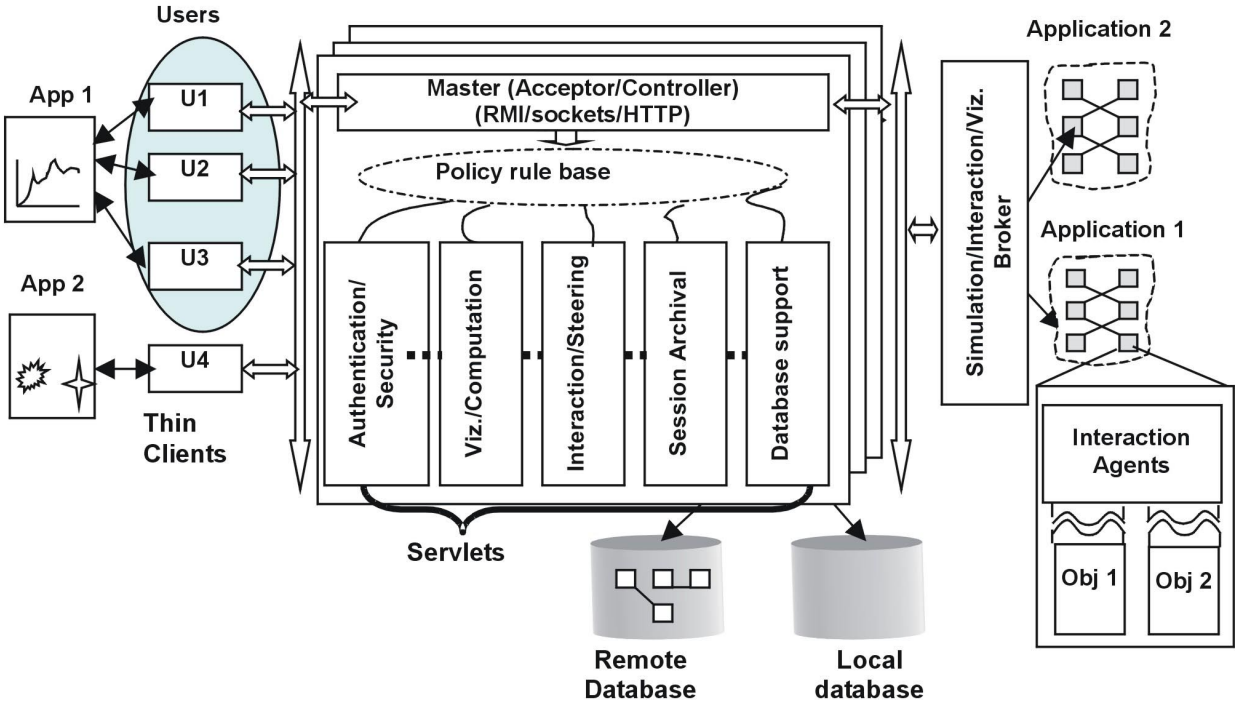

Figure 3. Architectural schematic of the DISCOVER computational collaboratory

4.1 DISCOVER Interaction and Collaboration Servers

The DISCOVER Interaction and Collaboration server builds on a commodity web server, and extends its functionality using Java servlets [55][56], to provide specialized services for real-time application interaction and steering and collaboration between client groups. Clients connect to the server using standard HTTP communication using a series of HTTP *GET* and *POST* requests. At the other end, application-to-server communication is achieved either using standard

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

distributed object protocols such as CORBA [18] and Java RMI [57], or a more optimized, custom protocol using TCP sockets.

The DISCOVER middleware creates 3 communication channels 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 local or remote application, and (3) a *ResponseChannel* for communicating application responses to the interaction requests. Clients differentiate between the different messages (i.e. Response, Error or Update) using Java's reflection mechanism, by querying the received object for its class name. Messages are processed differently at the client based on their type.

An *ApplicationProxy* object is created at the server for each active application, and is given a unique identifier. This object encapsulates the entire context for the application.

The 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 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. We briefly discuss some of the core handlers below.

The *master (accepter/controller) handler* servlet is the client's gateway to the server. The master servlet creates a session object for each connecting client and uses it to maintain information about client-server-application sessions. It provides each client with a unique client-id. The client-id along with an application-id (corresponding to the application to which the client is connected) is used to identify each session.

The *command handler* servlet manages all client view/command requests. On receiving these requests from the clients, this handler looks up the appropriate application proxy, and redirects them to this proxy. The collaboration handler described below handles the responses to these

requests. All requests and responses are Java objects and take advantage of Java's object serialization capability.

The *collaboration handler* enables multiple clients to collaboratively interact with and steer applications. All clients connected to a particular application form a collaboration group by default. Global updates (e.g. current application status) are automatically broadcast to this group. Clients can form or join (or leave) collaboration sub-groups within the application group. Clients can also disable all collaboration so that their requests/responses are not broadcast to the entire collaboration group. Individual views can still be explicitly shared in this mode. In addition to collaborative interaction/steering, the client portal is provided with chat and whiteboard tools to further assist collaboration.

The *Daemon servlet* forms the bridge between the server and the applications. Each application is authenticated at the server using a pre-assigned unique identifier. The daemon servlet creates an Application Proxy for each new application that connects to it, and maintains a handle to this proxy object. It also assigns the application a unique session identifier. It buffers all client requests and sends them to the application when the application is in the "interaction" phase. This ensures that requests are not lost while the application is busy computing.

4.1.1 DISCOVER Model for Requests - Responses

DISCOVER uses an asynchronous mode for sending client requests and getting the respective responses. This is primarily due to the iterative nature of the applications currently used as the applications alternate between cycles of computing and interaction. All client commands are buffered at the server when the application is in the computing cycle and this buffer is emptied sequentially during the interaction phase. The application responses are received asynchronously at a later point of time. Because of this asynchronous mode, the server has to keep track of all the clients who issued a particular request. The server maintains this in a multi level hash table indexed by the command key, which is unique for each request. When a

response is received at the server from the application, a command key is extracted out of it, and matched against the keys in the table. On a successful match, the corresponding list of clients who issued this command is retrieved, and all those clients are updated with the new response. The table also maintains the mode (Collaborative or non Collaborative) in which a particular request was sent, and the server uses this mode to decide whether all clients should be updated (if it was issued by a client in the collaborative mode) or only the client who issued it should be updated (if that client was in a non collaborative mode).

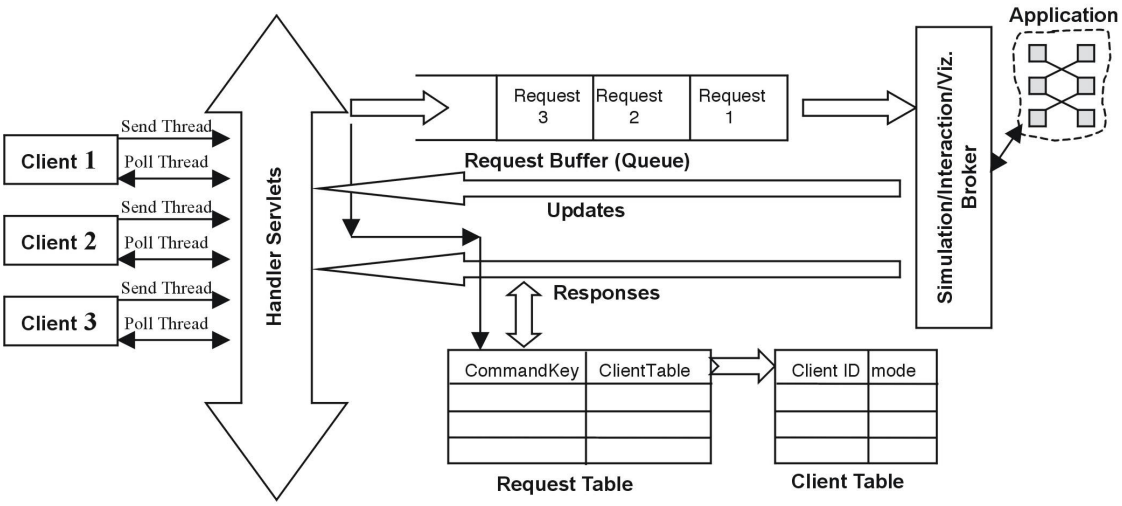

Figure 4. Asynchronous communication at the server for requests and responses

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

The primary objective of the DISCOVER middleware substrate is to enable integration of 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. Having all the 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 architecture with coupled server/application(s) sets is a more appropriate architecture for such integration.

An overview of the DISCOVER network of peer-to-peer servers is presented in Figure 5. The DISCOVER peer-to-peer architecture 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, an archival service or grid services using Java/CORBA CoG Kits. A domain will typically consist of different types of servers. Within a domain, all the servers share the same database of users and applications. They 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 applications in all domains based on its access privileges and credentials.

The middleware substrate builds on CORBA/IIOP, which provides peer-to-peer connectivity between DISCOVER servers within and across domains. Server/service discovery mechanisms are built using the CORBA *Trader Servic*e [58], 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 required to implement the middleware substrate. It allows interoperability between servers, while they can still maintain their different individual architectures and implementations. Moreover, we believe that the servers will be typically connected through reasonable bandwidth links (~1 Mbps). As no assumptions can be made about client-server connections, having the client connect to the "nearest server", and using CORBA/IIOP to connect that server and the desired application may actually reduce client latencies.

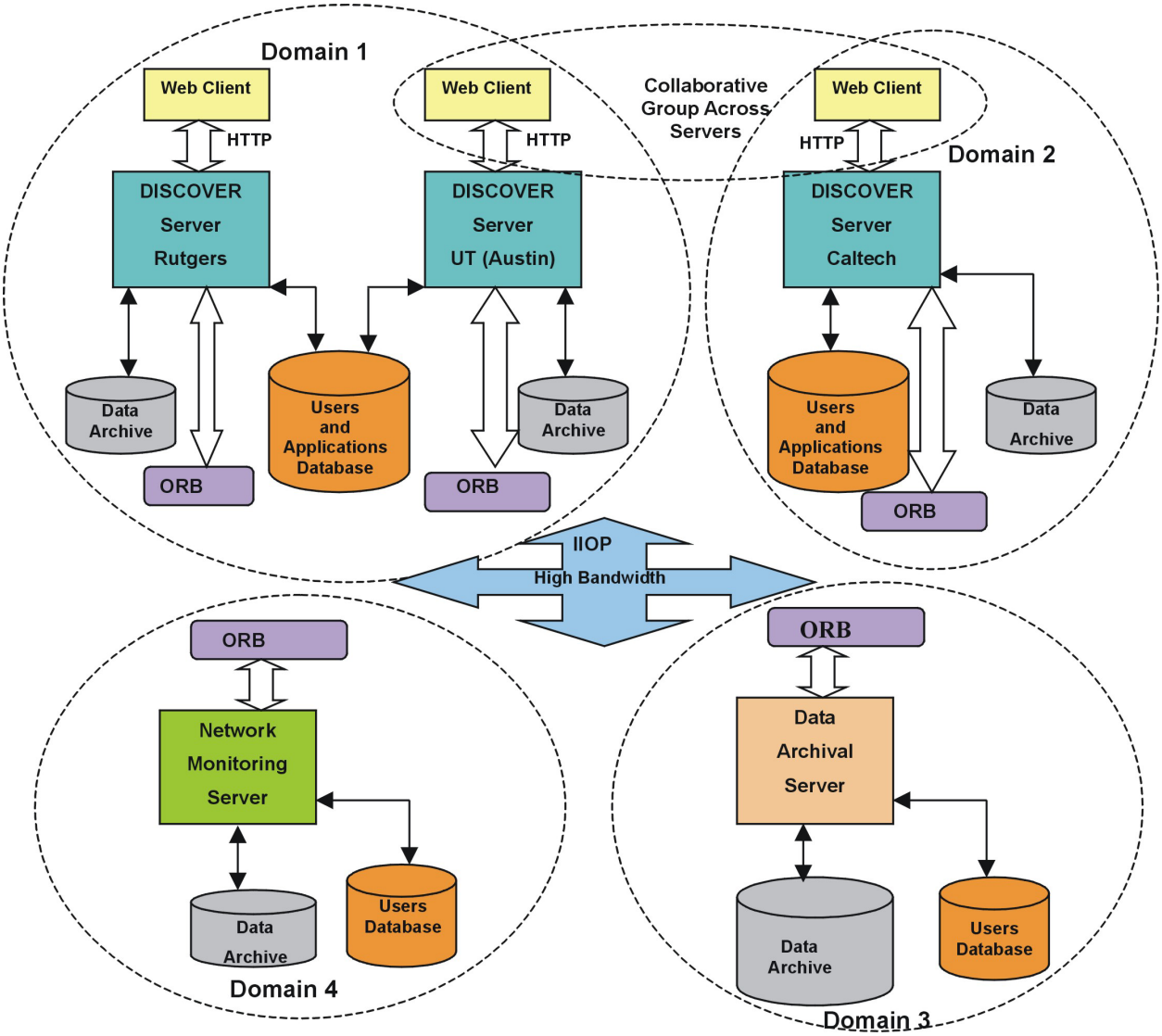

Figure 5. A deployment of DISCOVER servers providing access to a repository of services

In order to enable this integration of DISCOVER Interaction and Collaboration servers, the middleware substrate should support mechanisms such as:

- Authentication/Security across servers: Since clients will be accessing applications connected to remote servers, the middleware substrate should be able to authenticate a client with remote servers and/or remote applications.

- 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 should be able to form a collaborative group, even if they are interacting with it through different servers.
- Data/State Consistency across servers: DISCOVER servers use a simple locking mechanism to make sure that applications remain in a consistent state during collaborative sessions. The middleware substrate should be able to extend this locking mechanism, in order 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 must be maintained separately.

The implementation and operation of these mechanisms is described in the following chapter.

Chapter 5

Implementation and Operation of the DISCOVER Middleware Substrate

This chapter presents the implementation and operation of a CORBA-based prototype of the DISCOVER middleware substrate.

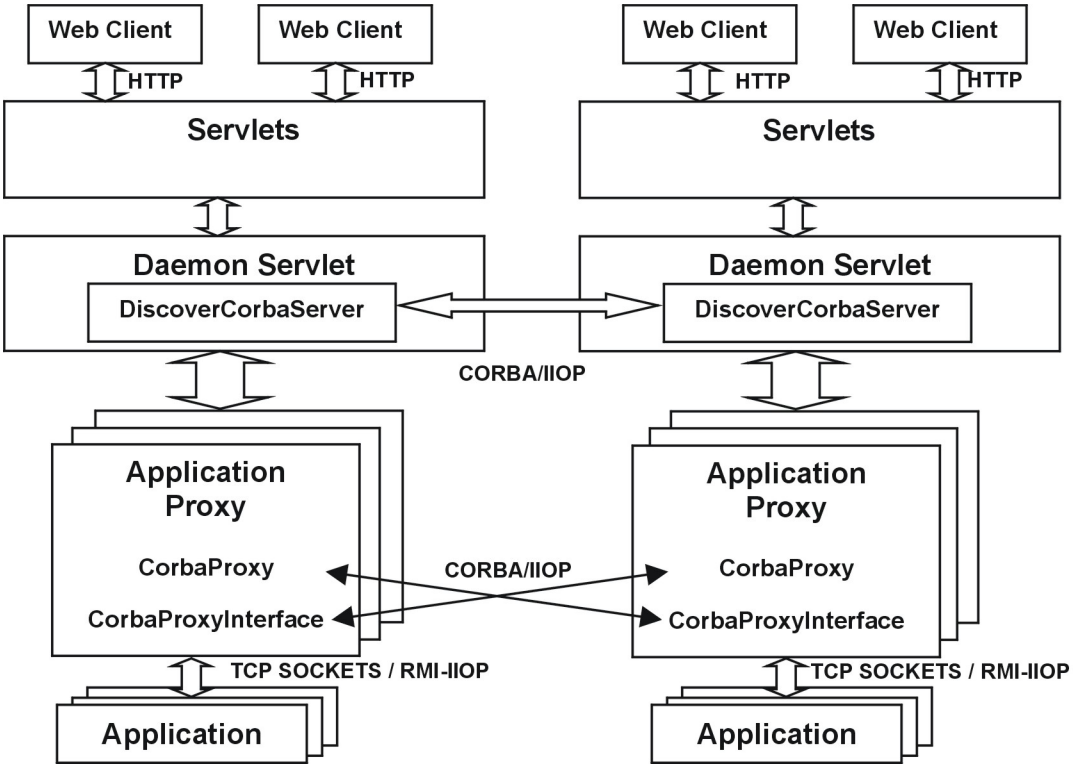

Figure 6. Interaction model between DISCOVER servers

5.1 Middleware Implementation

The middleware substrate builds on the DISCOVER interaction/collaboration server architecture described in Chapter 4. It implements the 2 interface levels described in Chapter 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 at a server. This

interface defines the functionality provided by the application and allows remote servers to access this functionality.

Sever to server communication in the DISCOVER middleware uses the same 3 communication channels set up for the application-server communication, i.e. *Main Channel*, *Command Channel* and *Response Channel* (see Section 4.1). In addition, server-server communication also uses a *Control Channel* for error messages and system events. The *Control Channel* serves as a notification service similar to the one used in Salamander substrate [35][36][37].

The *ApplicationProxy* object created at the server for each active application encapsulates the entire context for the application including its *CorbaProxy* interfaces. The schematic of the middleware implementation is presented in Figure 6. The two interfaces are described below.

5.1.1 The DiscoverCorbaServer Interface

The *DiscoverCorbaServer* interface is implemented by each server, and specifies methods for interacting with the server. This includes methods for authenticating with the server, getting a list of all active services on the server, and getting a 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. *DiscoverCorbaServer* publishes its availability using the CORBA trader service. It also maintains a table of references to the *CorbaProxy* objects (i.e. *CorbaProxyInterface*) for remote applications. Using this reference, the DISCOVER Daemon Servlet can provide transparent access to remote applications and support local clients' interactions with those applications. Thus all requests for remote applications from locally connected clients go through the *DiscoverCorbaServer*, which then forwards them to the appropriate *CorbaProxyInterface* reference.

5.1.2 The CorbaProxy Interface

The *CorbaProxy* interface represents a service or an application that is active at a particular server. This interface specifies all the methods that are 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* interface is therefore an application's gateway for all other servers. All servers that have clients interacting with remote applications maintain a reference to the *CorbaProxy* objects for those applications (as mentioned above, the DiscoverCorbaServer object maintains this table of references). *CorbaProxy* also binds itself to the CORBA naming service using the application's unique identifier as the name. This allows the application to be remotely accessed from any server.

A *CorbaProxy* object is contained within each DISCOVER *ApplicationProxy* object. The *ApplicationProxy* object manages all communication with the application required during application registration, or during interaction and steering with the application. In the case of local applications, *ApplicationProxy* directly communicates with the applications using the appropriate protocol. In the case of remote applications however, this communication is done with the remote *CorbaProxy* object using its local reference (i.e. *CorbaProxyInterface*).

5.2 Middleware Operation

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

5.2.1 Servers and Applications Discovery

DISCOVER servers locate each other using the CORBA trader services. The CORBA trader service maintains all the server (*DiscoverCorbaServer*) references as *service-offer* pairs. In our prototype we have implemented a minimalist 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 an object can be identified based on the service it provides or its properties list.

Applications are located using their globally unique identifiers, which are dynamically assigned by the *DaemonServlet*. The application identifier is chosen to be a combination of the server's IP address and a local count of applications on each server. This ensures that even if the same application is connected to multiple servers or multiple instances of an application are connected to the same server, each instance will have a unique identifier. Moreover, 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.

5.2.2 Security/Authentication across Servers

As described in Chapter 4 (Section 4.1), 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 the 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* to authenticate the client with each server in the network, and in return gets the list of active applications connected to all the servers to which the user has some 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 only it can interact in ways defined by its privileges and capabilities. Note that a client has access only to those servers where he is a registered user – i.e. he is on the authorized user list for at least one of the applications registered with the server. Thus, in the current system, a client's user-Id for a particular application is assumed to be consistent across all servers.

5.2.3 Collaboration across Servers

DISCOVER enables multiple clients to collaboratively interact with and steer applications. As described in Chapter 4 (section 4.1), the dedicated *collaboration handler* servlet handles all 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. These collaboration groups can span multiple servers. In this case, the *CorbaProxy* objects 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 any of the clients' requests, and the *control channel* for retrieving any error messages or steering control (locks) 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, now, 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 as shown in Figure 7. Since clients always interact through the server closest to them and the broadcast messages for collaborative updates 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 when the servers are geographically far away. It also leads to better scalability in terms of the number of clients that can be supported within a

collaboration session without overloading a server as the collaboration load now spans across multiple servers.

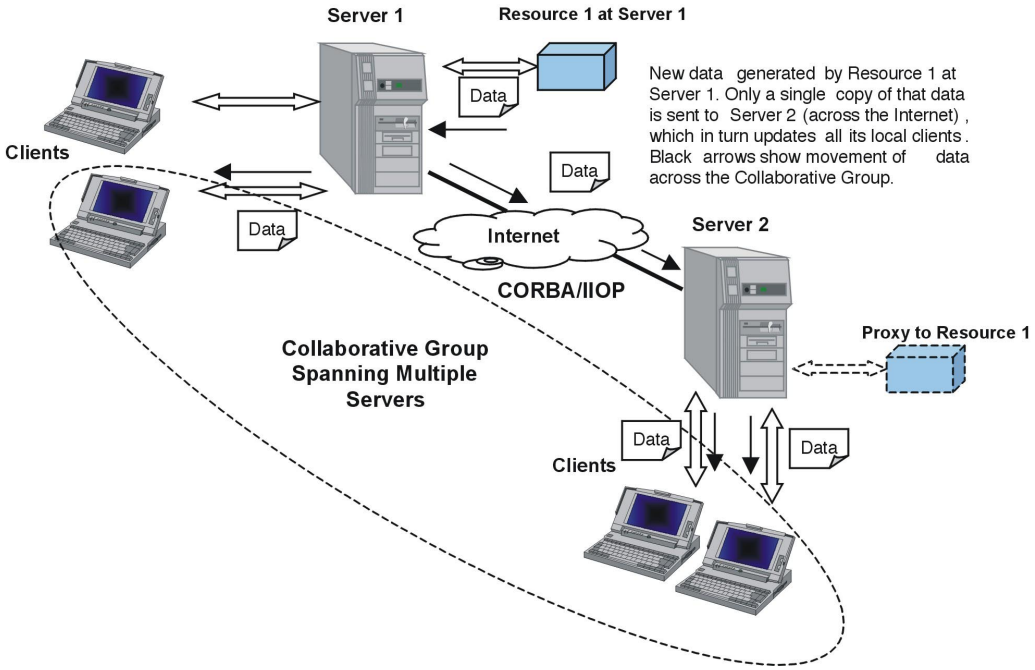

Figure 7. Collaborative group spanning multiple servers

5.2.4 Distributed Locking

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 framework, 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 this 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.

5.2.5 Distributed Logging

The session archival handler maintains two types of logs. The first one logs all interactions between a client(s) 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 data generated by the application is sent to all the servers who have clients interested in that data set. Thus handling of the output files, resulting data sets, etc., are all handled by the home server or the server to which the clients are directly connected and the home server creates the output files or the records under the ownership of the client who requested that data. The p2p architecture doesn't allow creation of files on a remote server by the clients.

The second log maintains all requests, responses, and status messages for each application. This log allows clients to have direct access to the entire history of the application. For remote applications, all the data generated by the application throughout its execution is logged at that application's host server (the server to which the application is directly connected).

Chapter 6

Performance and Design Evaluation

6.1 An Experimental Evalutation 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 (CACR), California Institute of Technology. Figure 8 shows the setup used for the experimental evaluation presented in this section. The middleware implementation used Apache Web Server 1.3 [59] with Apache Jserv 1.1.12 [60] as the servlet engine, and Visibroker for Java 4.5.1 [61] 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.

6.1.1 Experiment 1 – Access Latency over a Local Area Network (LAN) and a Wide Area Network (WAN)

This experiment consisted of 2 sets of measurements – the first set includes latency measurements taken on a LAN and the second set includes measurements taken on a WAN. For the local area network latency measurement, two DISCOVER servers running on a 10 Mbps local area network at Rutgers University were used. For the wide area network latency measurement, DISCOVER servers running at Rutgers University and at CSM, University of Texas at Austin were used. The clients were running on the local area network at Rutgers University for both sets of measurements.

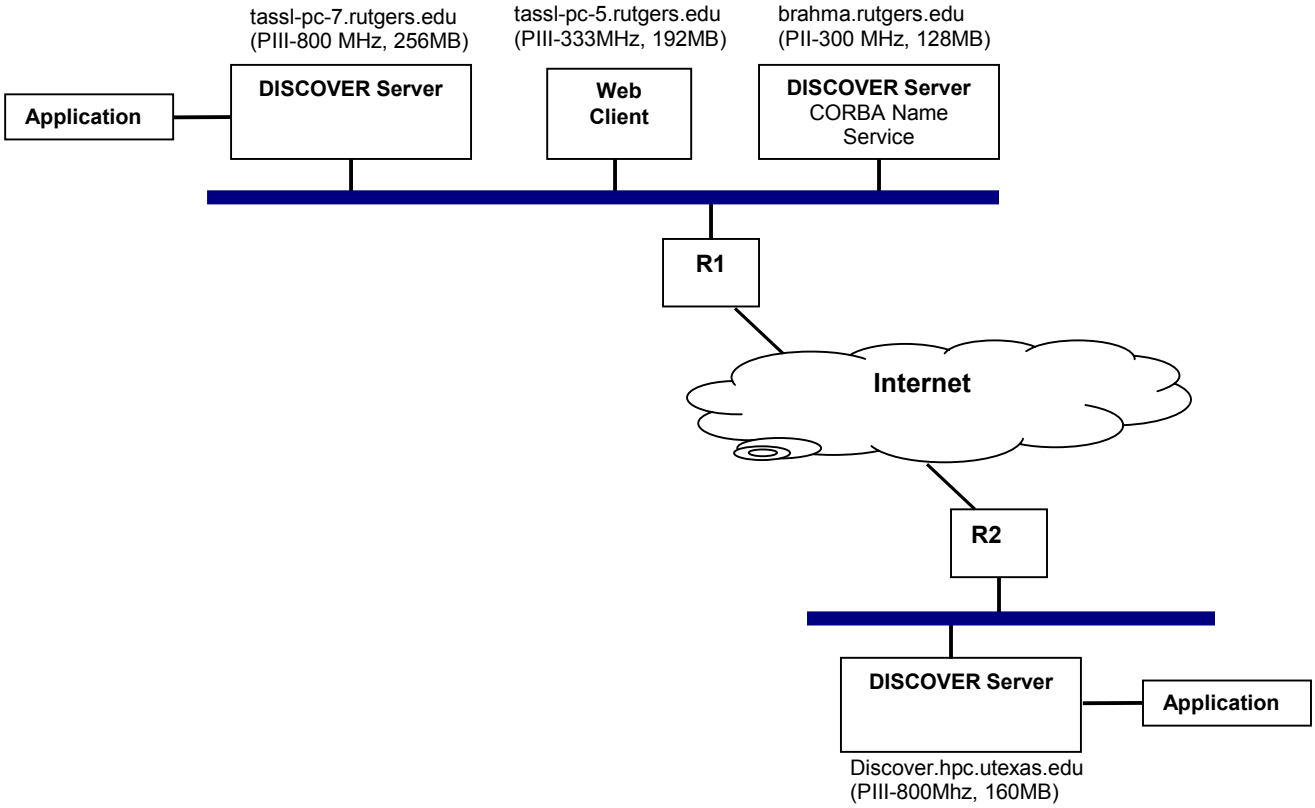

Figure 8. Setup for the experimental evaluation of the DISCOVER middleware

For all measurements 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 the case of the LAN measurements, the application was connected to one of the servers at Rutgers University, while in the case of the WAN measurements, the application was connected to the server at CSM, University of Texas at Austin. Requests for data of different sizes were issued. Response times were measured for direct access to the server where the application was connected and for indirect (remote) access through the middleware substrate. Direct access time includes the time taken for the client's request to be sent to the server over HTTP, the server handling the request and sending the request to the application, the server getting the response back and parsing it to form a Java Object and the response object being sent back to the client. The time taken by the application to compute the response is not included in this time. Indirect (remote) access time includes the direct access time and also the time taken by the server to send

the request to the remote server and then get the result back over IIOP. A mean response time was calculated over 10 measurements for each data size.

The resulting response latencies for direct and indirect accesses measured on the LAN are plotted in Figure 9. It can be seen that the response times for direct accesses to an application at a local server increase linearly with the increase in size of data. This indicates that the server overhead for request –response handling is almost constant and the change in latencies is largely due to the increase in communication times. The response times for indirect accesses to an application at a remote server also vary almost linearly. 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.

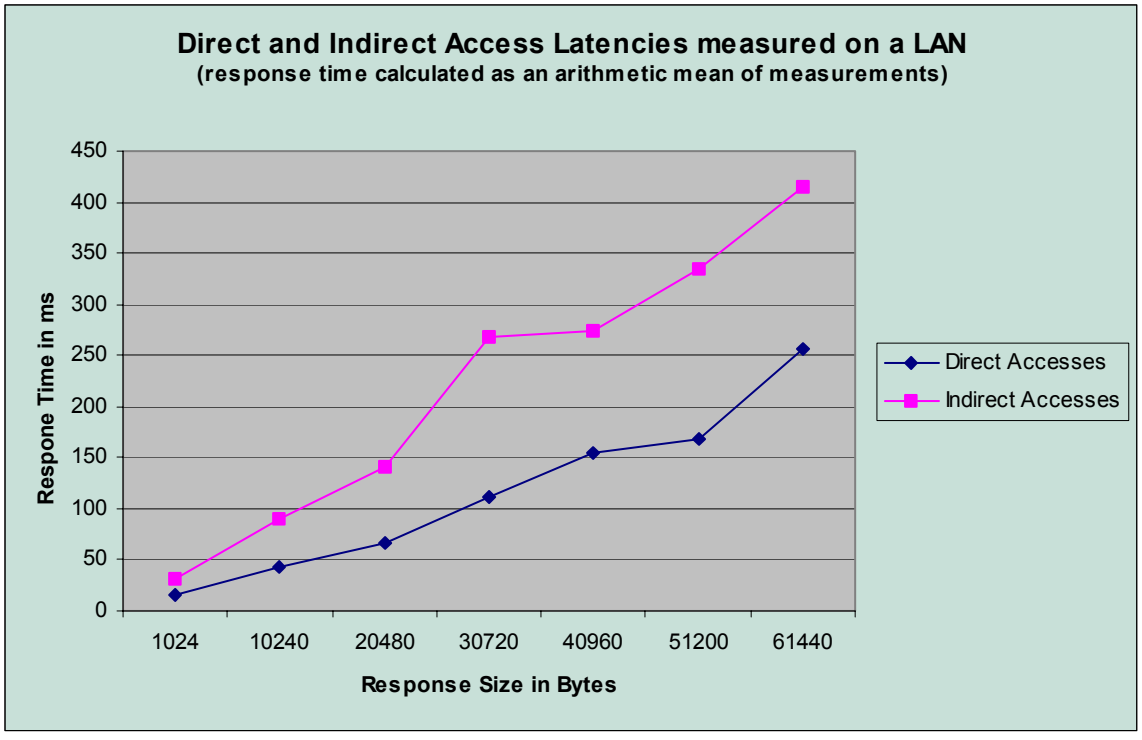

Figure 9. Comparison of latencies for direct and indirect application accesses on a Local Area Network (LAN)

The response latencies for direct and indirect responses measured on the WAN are plotted in Figure 10. Contrary to the results for the LAN, 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. At first look, these results might appear to be contradictory to expectations, but the experimental setup used for measurements on the WAN provides an explanation. The application was connected to the server at Austin, while both the second server and the client were running on machines at the local area network at Rutgers University. Direct access consisted of a client running at Rutgers accessing the server at Austin over HTTP. Indirect access consisted of a client running at Rutgers accessing the server within the same local area network at Rutgers over HTTP, which in turn accessed the server at Austin over IIOP. Thus for direct accesses, a large network path across the Internet was covered over HTTP which meant setting up a new TCP connection over a wide area network for every request. For indirect accesses, only a short network path was covered over HTTP (within the same LAN) and the larger network path (across the Internet) was covered over IIOP, which used 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.

For small data sizes HTTP performs worse than IIOP for a wide area network because the connection setup times constitute a large portion of the overall communication time. As data sizes increase, the overhead of connection set up time becomes a relatively smaller portion of the overall communication time involved and hence, the overall access latency is dominated by the communication time, which is larger for remote accesses as they involve accesses to two servers.

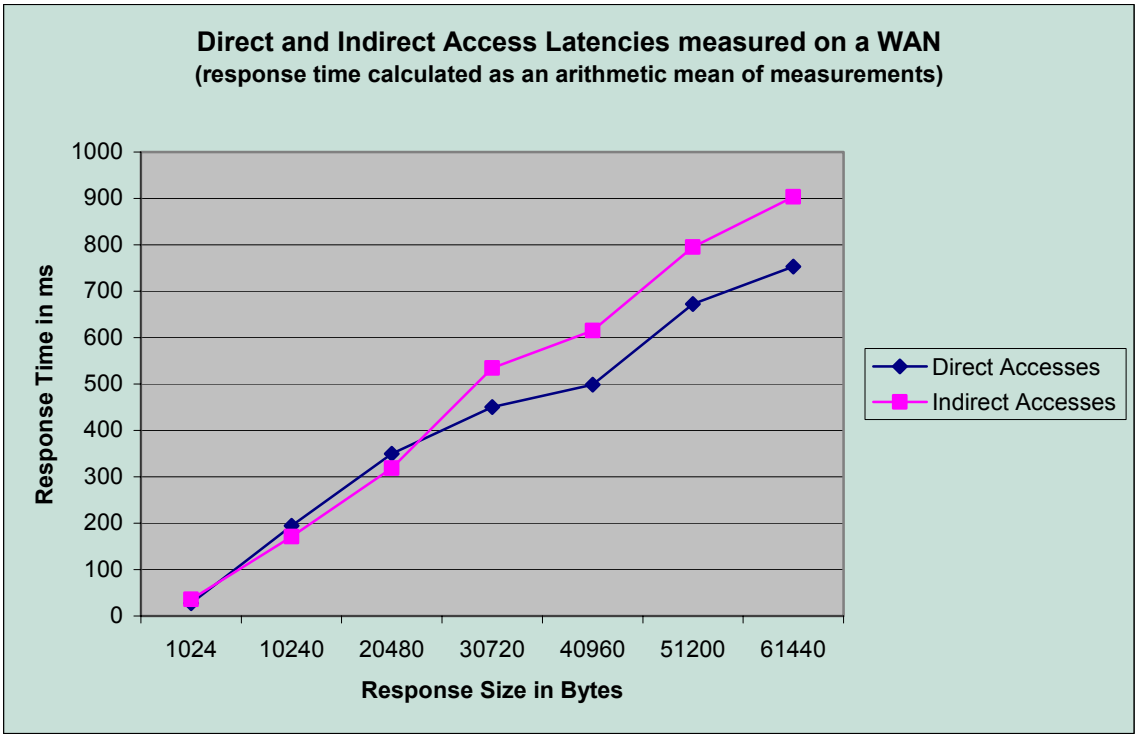

Figure 10. Comparison of latencies for direct and indirect application accesses on a Wide Area Network (WAN)

IIOP vs. HTTP for Wide Area Networks: One well-known problem with HTTP is that a new TCP connection is opened for each request, and closed after the response has been delivered. This is an inefficient use of system and network resources and a source of delay. Although persistent connections or keepalive connections are supported as a standard option in HTTP/1.1, both the client and the server should have the keepalive capability and this should be negotiated before the start of a new connection. By contrast, GIOP and its mapping for TCP/IP – IIOP is designed to allow a connection to be used for multiple requests, and also to allow overlapping requests; amortizing setup costs over many requests/replies. Several requests may be sent over the same connection without waiting for a reply, and the replies may be delivered in any order. The replies are matched to requests by the use of identifiers. In a CORBA system, the ORB provides connection and session management to make best use of the available resources. Thus, larger the network path, over which IIOP is used, higher will be the performance gain as compared to the

use of HTTP. This idea has been used in earlier work to improve performance. Gateways that convert HTTP interactions into the corresponding interactions described by the IDL are proposed in [64]. These two gateways are the I2H gateway, which converts IIOP requests to HTTP, and the H2I gateway, which converts HTTP requests to IIOP. By co-locating the I2H gateway near the web server, and the H2I gateway near the web client, the IIOP protocol can be used over most of the route. In cases where this part of the route accounts for a significant part of the round trip time, the ability of IIOP to re-use an existing connection shows an improvement.

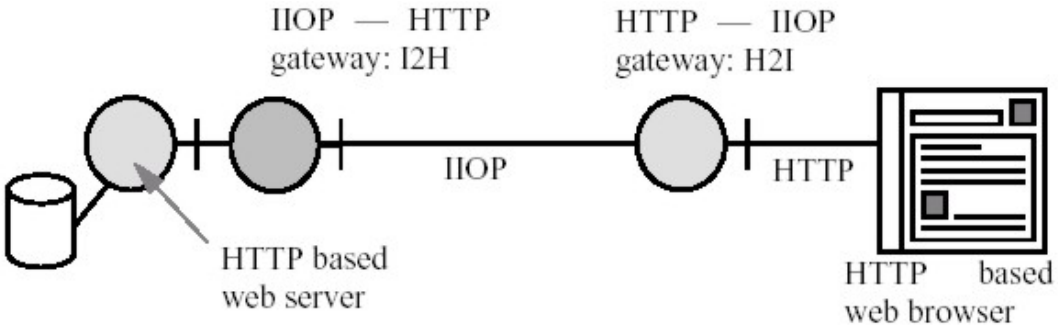

Figure 11. Use of IIOP as the protocol for the World Wide Web

6.1.2 Experiment 2 – Access Latency with Multiple Simultaneous Clients

This experiment measured the variation of direct and indirect access latencies in the presence of multiple simultaneous clients over a LAN. The setup used for this experiment was the same as that described above for experiment 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 12. 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 9), both for direct and indirect accesses. The 3 points towards the right end of the graph (for 16,

17 and 19 clients) are probably due to the communication and network irregularities – we are unable to explain these values and are in the process of repeating these experiments.

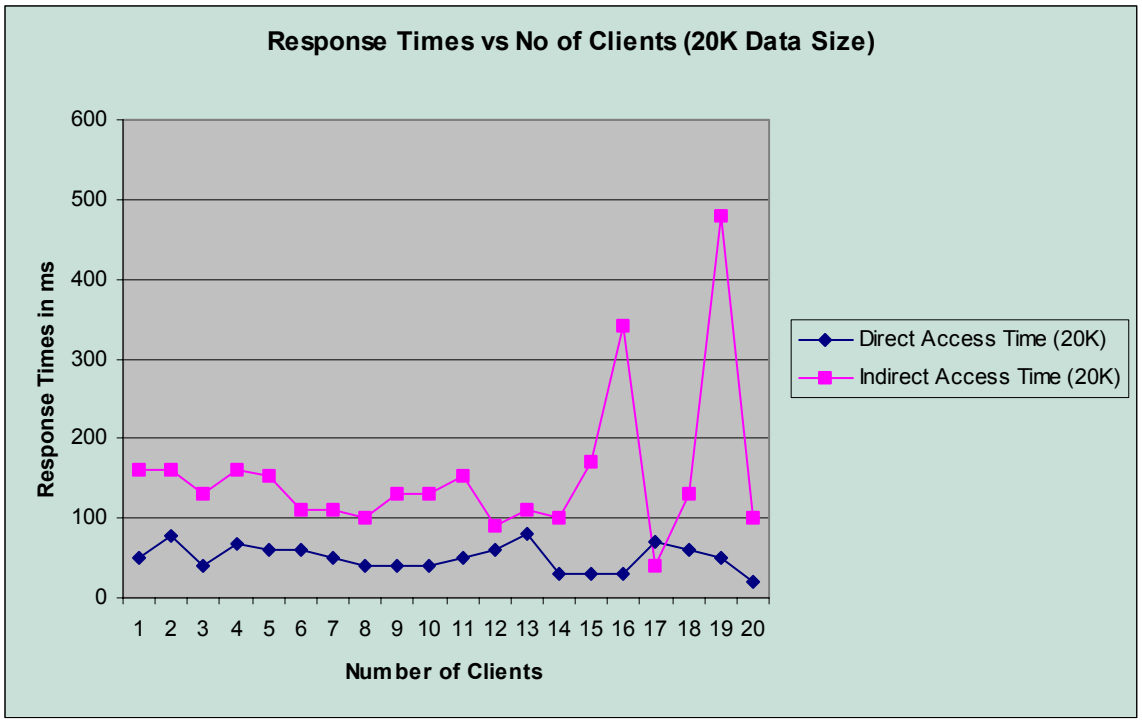

Figure 12. Variation in access latencies with multiple, simultaneous clients over a LAN

6.1.3 Experiment 3 – An Evaluation of Server Memory Requirements

This set of experiments was conducted to compare the memory used at the server for different configurations when multiple applications were connected to it. Memory usage was calculated 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 to the total amount of memory currently available for future allocated objects, measured in bytes and the total amount of memory currently available for current and future objects, measured in bytes, 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 approximate and actual values will be a little

lesser than those shown here. However, it was made sure that only the server process was in operation in the JVM during the experiment, so that the changes in memory usage reflect the memory allocated for new applications.

For the single standalone server configuration, applications were connected to a server with no CORBA calls and no ORB and Naming Service running. Thus, there were no updates generated for remote servers. A single client running on a different machine was used to issue requests for 1K of data and memory usage was measured as the number of applications connected to the server was increased. In the subsequent experiments, the two servers at the Rutgers University LAN, with configurations described earlier were used. For measuring the memory usage at a local server when it publishes all its local applications for remote accesses, applications were connected to a server and the server created the corresponding CORBA objects for the applications (*CorbaProxy* objects), which registered themselves with the CORBA Naming Service. CORBA updates for remote servers were generated. A single client running on a different machine accessed one of the applications remotely through CORBA. Memory usage was measured at the local server (the server where the application was connected). In the next measurement, multiple clients were used to access multiple applications remotely through CORBA. Memory usage was measured both at the local server (which is similar to the previous experiment, except that there are multiple clients accessing multiple applications instead of a single client accessing one of the many applications) and at the remote server (memory usage at the remote server represents the memory required for storing remote CORBA references of applications and invoking IDL methods on them).

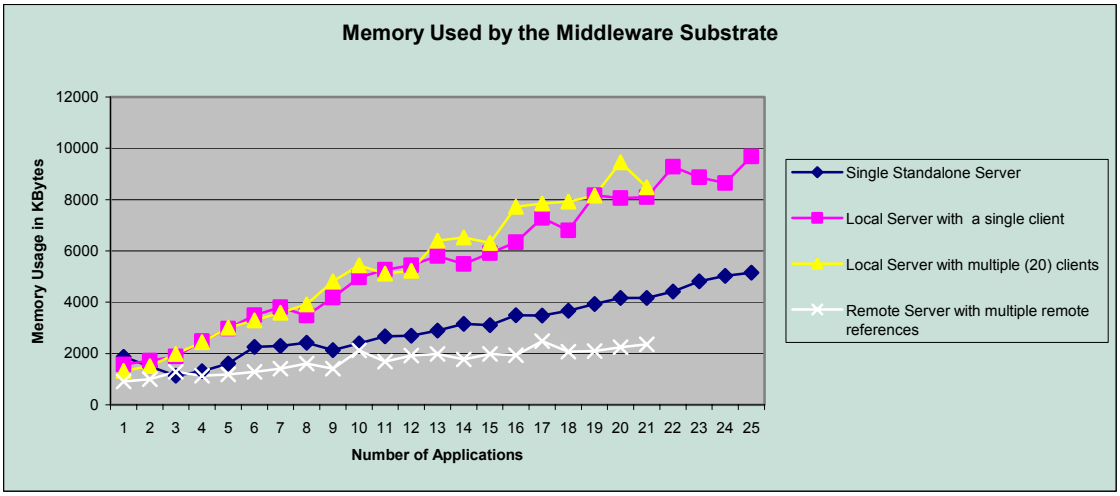

Figure 13. Sever memory utilization for different configurations

The results are plotted in Figure 13. As expected, the memory usage increased for all cases as the number of applications is increased. It is interesting to see that the memory usage for accessing remote applications is even less than that for accessing local applications on a single standalone server with no CORBA calls. However, in order to publish a local application for remote use by other servers, the memory usage at a local server with the middleware substrate is almost double as compared to the single standalone server case. Thus, by using the middleware substrate for integration, the local servers incur additional memory overheads, whereas the remote servers benefit by reducing their memory usage. It should be noted that the memory used by the server for maintaining client states is not significant. This can be inferred from the graphs of local server with a single remote client and local server with multiple remote clients cases.

6.2 Retrospective Evaluation of the Design and Technologies Used

The primary goal of the solutions presented in the thesis 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, rather than clients accessing remote servers through HTTP over a wide area network, provides comparable performance even for indirect accesses while enabling scalability, higher availability and enhanced set of services for the users.

But the use of these commodity technologies 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 FIFO buffers at the server for each client to support slow 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.

6.3 Challenges and Open Issues in a Peer-to-Peer Computational Environment

The peer-to-peer integration of DISCOVER servers in a Grid environment consisting of multiple institutions and different administrative domains has presented many challenges. We briefly discuss a few of them.

Authentication and Security across servers: Authentication of users and applications across servers presents a significant challenge. DISCOVER tries to minimize global knowledge

by having the services or applications identify the users (or user-IDs) that have access and their access privileges. Thus when an application or a service registers with a server, it supplies the server with this information in the form of a list of authorized users-IDs and their privileges. Once a user-ID is supplied, a server will automatically authenticate that user-ID. Thus, even though the system assumes consistent user-IDs across all servers, this is hardly a problem, as the user-IDs do not belong to a server but to an application/service. In the current implementation, the user is authenticated at his home server before he can proceed. One way to get around this problem is to have a centralized directory service like the GIS that maintains user-IDs and other global information. All the servers in the system can now use this directory service.

Resource utilization: It is important to account for the resources used by any remote server. Currently, the system does not track the use of resources. It is, however, possible to add control mechanisms by creating access policies for each server, and then restricting each server's use of resources according to that policy. The access policies can be added to the current ACLs and can be defined in terms of metrics like number of requests per second, or the data bytes being transferred to each server per second.

Data Management and ownership across servers: In a peer-to-peer environment enabling interaction and steering of remote applications, the management of the generated data becomes important. The current implementation of DISCOVER avoids these issues by using Relational Databases to store all the data generated in the form of records. Access to these databases is provided through customized interfaces and is protected through passwords. The data produced by the application/service in response to clients' requests is handled by the local server (i.e. the server to which the clients are directly connected) and the local server creates the output files or the records under the ownership of the user who requested that data. The periodic data generated by the applications/services are handled by the home server of the application (i.e. the server to which the application is directly connected) and records are created under the ownership of the user-id who owns the application. All the clients, who have access privileges to this application,

are also provided with read only access rights to these records. Thus, the peer-to-peer architecture doesn't allow creation of files/records on a remote server by the clients.

Some of these issues are still open and we are in the process of addressing them. Note that the current implementation of DISCOVER is a prototype aimed at evaluating the viability of the peer-to-peer architecture and current commodity technologies for addressing the requirements of current and emerging Grid applications.

Chapter 7

Conclusion and Future Work

This thesis presented the design, implementation, and operation of a middleware substrate that enables peer-to-peer integration of and global collaborative (web-based) access to multiple, geographically distributed instances of the DISCOVER computational collaboratory for interaction and steering. The substrate builds on the CORBA distributed object technology and enables dynamic application/service discovery on the Grid, remote authentication and access control, coordinated interactions for collaborative interaction and steering. A retrospective evaluation of the design and an experimental evaluation of the middleware substrate were also presented. The performance of the middleware substrate over a wide area network validated the middleware design and also justified the use of CORBA/IIOP for inter-server communication. Some open issues in peer-to-peer environments were also discussed.

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 the Center for Subsurface Modeling (CSM), University of Texas at Austin, and is being expanded to include a deployment at the Center for Advanced Computational Research (CACR), California Institute of Technology.

This thesis also investigated the requirements for achieving interoperability among collaboratories operating on the Grid. The impact of these requirements on the middleware design of a collaboratory was discussed and a hierarchical architecture that builds on common protocols for interoperability was presented. A key contribution of this thesis is the design of a middleware substrate that enables interoperability among multiple, independently administered and managed,

instances of an entire collaboratory deployed at geographically distributed locations, with the goal of enabling global access and sharing of services across these instances with acceptable performance. This requires the servers to be lightweight, portable and easily deployable so that a server or network of servers (constituting the collaboratory middleware) can be installed anywhere where there is a growing community of users, much like a HTTP proxy server.

DISCOVER middleware presents a prototype implementation of this design with different services like resource discovery, request dispatching, status monitoring, and remote authentication. Interoperability in the current implementation of the middleware is achieved by sharing interfaces defined in the CORBA IDL. As discussed in chapter 3, IDL based approaches require each implementation to know about the interface supported by every other implementation. This raises the issue whether such approaches can support the level of interoperability required for global sharing of resources. As mentioned earlier, one option is to build grid ORBs with these IDLs integrated into the ORB as standard CORBA services. Just as the prototype DISCOVER middleware substrate provided interesting information and comparison on the use of IIOP over a WAN, more prototype implementations like the DISCOVER middleware substrate, which target interoperability using different protocols and technologies should be experimented with. XML based protocols like SOAP and peer-to-peer initiatives like JXTA hold a lot of promise for the Grid computing community and deserve more attention.

The DISCOVER middleware substrate currently allows interoperability between multiple instances of the DISCOVER computational collaboratory which is the first step towards achieving overall interoperability among collaboratories on the Grid. We have started working on integrating the different services provided by the DISCOVER middleware substrate into an ORB as standard CORBA services and are currently evaluating open source ORBs like JacORB [62] and MICO[63]. We are also developing interfaces to further consolidate the pool of services model proposed in the design.

In a related research effort we are building a CORBA CoG kit to provide application developers with access to Grid services using CORBA [31][2]. The overall goal is to integrate Grid services provided by CORBA with the collaborative interaction and steering services provided by DISCOVER. For example a client can use Globus [66] services provide by the CORBA CoG Kit to discover, allocate and stage a scientific simulation, and then use the DISCOVER web-portal to collaboratively monitor, interact with, and steer the application.

[2] www.caip.rutgers.edu/TASSL/CorbaCoG/CORBACog.htm

References

[1]. V. Mann and M. Parashar, "Middleware Support for Global Access to Integrated Computational Collaboratories", Proc. of the 10th IEEE symposium on High Performance Distributed Computing (HPDC-10), San Francisco, CA, 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]. 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]. J. H. Lee, A. Prakash, T. Jaeger, and G. Wu, "Supporting multi-user, multi-applet workspaces in CBE", Proc. of the ACM 1996 Conf. on Computer Supported Cooperative Work (CSCW'96), Cambridge, MA, pp.344-353, November 1996.

[8]. 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.

[9]. R. A. Whiteside, E. J. Friedman-Hill, and R. J. Detry, "PRE: A framework for enterprise integration", Proc. of Design and Information Infrastructure Systems for Manufacturing (DIISM), Fort Worth, TX, May 1998.

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

[11]. NetSolve - http://www.cs.utk.edu/netsolve/

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

[13]. 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 Tenth IEEE International Symposium on High Performance Distributed Computing, August 2001 (submitted).

[14]. Cactus Computational Collaboratory. http://www.cactuscode.org.

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

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

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

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

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

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

[21]. 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.

[22]. N. H. Kapadia, R. J. Figueiredo, and J. A. B. Fortes, "PUNCH: Web Portal for Running Tools", IEEE Micro, May-June 2000.

[23]. 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.

[24]. 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.

[25]. HotPage User Portal- https://hotpage.npaci.edu/

[26]. 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.

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

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

[29]. Gregor von Laszewski, Ian Foster, Jarek Gawor, Peter Lane, Nell Rehn, and Mike Russell, "Designing Grid-based Problem Solving Environments and Portals", Proceedings of the 34th Hawaii International Conference on System Sciences, January 2001.

[30]. Commodity Grid Toolkits (CoG), http://www.globus.org/cog

[31]. S. Verma, J. Gawor, M. Parashar, and G. von Laszewski, "A CORBA Commodity Grid Kit", Submitted to the 2nd International Workshop on Grid Computing, November 2001.

[32]. Nimrod/G Problem Solving Environment and Computational Economies, http://www.csse.monash.edu.au/~rajkumar/ecogrid/

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

[34]. 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

[35]. 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.

[36]. G. R. Malan, F. Jahanian and P. Knoop, "Comparison of Two Middleware Data Dissemination Services in a Wide-Area Distributed System", Proceedings of the 17th IEEE International Conference on Distributed Computing Systems, May 1997, Baltimore, MD.

[37]. G. R. Malan, F. Jahanian, C. Rasmussen, and P. Knoop, "Performance of a Distributed Object-Based Internet Collaboratory", Technical Report CSE-TR-297-96, University of Michigan EECS Deptartment, July 1996.

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

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

[40]. 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

[41]. T. Suzumura, H. Nakada and S. Matsuoka, "Are Global Computing Systems Useful? – Comparison of Client-Server Global Computing Systems - Ninf, Netsolve versus CORBA", Proceedings of the 14th International Parallel and Distributed Processing Symposium (IPDPS'00).

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

[43]. Napster, http://www.napster.com/

[44]. Gnutella, http://gnutella.wego.com/

[45]. I.Clarke, O. Sandberg, B.Wiley, and T.W. Hong, "Freenet: A Distributed Anonymous Information Storage and Retrieval System", ICSI Workshop on Design Issues in Anonymity and Unobservability, 1999.

[46]. SETI@home, http://setiathome.ssl.Berkeley.edu

[47]. Parabon, www.parabon.com

[48]. Entropia, www.entropia.com

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

[50]. Intel Proposals on Peer-to-Peer Computing, http://www.intel.com/ebusiness/products/peertopeer/index.htm

[51]. Universal Description Discovery and Integration (UDDI), Technical White Paper, http://www.uddi.org, September 6,2000.

[52]. Microsoft .NET, http://www.microsoft.com/net/

[53]. Web Services Description Language (WSDL), http://www-106.ibm.com/developerworks/library/w-wsdl.html

[54]. 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.

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

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

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

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

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

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

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

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

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

[64]. O. Rees, N. Edwards, M. Madsen, M. Beasley, and A. McClenaghan, "A Web of Distributed Objects", Fourth International World Wide Web Conference, December 1995, Boston, Massachusetts (MA).

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

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

[67]. 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.