Design and Implementation of a CORBA Commodity Grid Kit

Snigdha Verma[1], Manish Parashar[1], Jarek Gawor[2] and Gregor von Laszewski[2]

[1] The Applied Software Systems Laboratory,
Department of Electrical and Computer Engineering,
Rutgers University, 94 Brett Road, Piscataway, NJ 08854-8058, U.S.A
{snigdha,parashar}@caip.rutgers.edu
http://www.caip.rutgers.edu/TASSL/CorbaCoG/
[2]Mathematics and Computer Science Division
Argonne National Laboratory, 9700 S. Cass Ave, Argonne, Il, 60440, U.S.A.
{gawor,gregor}@mcs.anl.gov
http://www.globus.org/cog/

Abstract. This paper reports on an ongoing research project aimed at designing and deploying a CORBA Commodity Grid (CoG) Kit. The overall goal of this project is to explore how commodity distributed computing technologies and state-of-the-art software engineering practices can be used for the development of advanced Grid applications. As part of this activity, we are investigating how CORBA can be integrated with the existing Grid infrastructure. In this paper, we present the design of a CORBA Commodity Grid Kit that provides a software development framework for building a CORBA "Grid domain." We then present our experiences in developing a prototype CORBA CoG Kit that supports the development and deployment of CORBA applications on the Grid by providing them access to the Grid services provided by the Globus toolkit.

1. Introduction

The past decade has seen the emergence of computational Grids aimed at enabling programmers and application developers to aggregate resources[1] scattered around the globe. However, developing applications that can effectively utilize the Grid still remains a difficult task. Although, there exist Grid services that enable application developers to authenticate, access, discover, manage, and schedule remote Grid resources, these services are often incompatible with commodity technologies. As a result, it is difficult to integrate these services into the software engineering processes and technologies that are currently used by application developers. Recently, a number of research groups have started to investigate Commodity Grid Kits (CoG Kits) to address this problem. Developers of CoG Kits have the common goal of developing mappings and interfaces between Grid services and a particular

[1] In this paper we use resources to collectively refer to computers, data stores, services and applications.

commodity technology (such as Java platform [1] [2], Java Server Pages [3], Python [4], and Perl [5]). We believe that CoG Kits will encourage and facilitate the use of the Grid, while at the same time leveraging the benefits of the commodity technology.
Recent years have also seen significant advances in commodity distributed technologies aimed at easing application development in distributed environments. One such technology is the **C**ommon **O**bject **R**equest **B**roker **A**rchitecture (CORBA) [6] defined by the Object Management Group (OMG). CORBA specifies an open, vendor independent and language independent architecture for distributed application development and integration. Furthermore, CORBA defines a standard interoperability protocol (i.e. GIOP and IIOP) that enables different CORBA implementations and applications to interoperate and be portable across vendors. CORBA has emerged as a popular distributed computing standard and meets the necessary requirements to be considered by application developers as part of the Grid infrastructure. It is therefore natural to investigate the development of a CoG Kit that integrates CORBA with the Grid such that CORBA applications can access (and provide) services on the Grid. Such an integration would provide a powerful application development environment for high-end users and create a CORBA "Grid domain".
This paper presents the design and implementation of a CORBA CoG Kit that provides CORBA application with access to Grid services provided by the Globus toolkit [7]. In this paper we first give a brief overview of the Grid and its architecture and introduce the services and protocols that we intend to integrate within the CORBA CoG Kit. We then briefly outline requirements, advantages and disadvantages of CORBA technologies from the point of view of Grid application developers. Next, we present the architecture of the CORBA CoG Kit, and describe the design, implementation, and application of a prototype. Finally, we conclude our paper and identify the directions of ongoing and future activities.

2. The Grid

The term "Grid" has emerged in the last decade to denote an integrated distributed computing infrastructure for advanced science and engineering applications. The Grid concept is based on coordinated resource sharing and problem solving in dynamic multi-institutional virtual organizations [8]. Grid computing not only provides access to a diverse set of remote resources distributed across different organizations, but also facilitates highly flexible sharing relationships among these resources, ranging from client-server to peer-to-peer. An example of a typical client-server relationship is the classical model where a remote client submits jobs to batch queues for resources at a supercomputer center. An example of peer-to-peer relationship is the collaborative online interaction and steering of remote (distributed) high-end applications and advanced instruments [9].

Grids must support different levels of control ranging from fine-grained access control to delegation and from single user to multi user, and different services such as scheduling, co-allocation and accounting. These requirements are not sufficiently

addressed by the current commodity technologies, including CORBA. Although sharing of information and communication between resources is allowed, it is not easy to coordinate the use of distributed resources spanning multiple institutions and organizations. The Grid community has developed protocols, services and tools, which address the issues arising from sharing resources in peer communities. This community is also addressing security solutions that support management of credentials and policies when computations span multiple institutions, secure remote access to resources, information query protocols that provide services for obtaining the configuration and status information about the resources. Because of the diversity of the Grid it is difficult to develop an all-encompassing Grid architecture. Recently, a layered Grid architecture representation has been proposed [8] that distinguishes a

- ***fabric layer***, that interfaces to local control including physical and logical resources such as files, or even a distributed file system,
- ***connectivity layer***, that defines core communication and authentication protocols supporting Grid-specific network transactions,
- ***resource layer***, that allows the sharing of a single resource while using a
- ***collective layer*** that allows to view resources as collection,
- and an ***application layer*** that uses the appropriate components of each layer to support applications.

Each of these layers may contain protocols, APIs, and SDKs to support the development of Grid applications. This general layered architecture of the Grid is shown in the left part of Fig. 1.

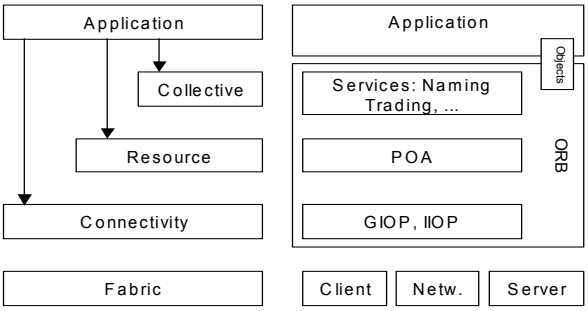

Fig. 1. The Grid Architecture and CORBA (The figure on the left shows the Grid architecture. The figure on the right shows how CORBA fits into the Grid Architecture).

3. CORBA and Grid Computing

CORBA provides advanced capabilities and services for distributed computing and can support the Grid architecture as shown in Fig. 1. Features of CORBA that makes it a suitable candidate for a CoG Kit include its high-level modular programming model, availability of advanced services (e.g. security, naming, trading, event, transaction, etc.) and readymade solutions, interoperability, language independence, location

transparency and an open standard supported by industry and academia. The interest in CORBA within the Grid Community has led to a number of efforts aimed at combining functionalities of CORBA and Globus [10]. While most of these efforts address specific problems encountered in individual applications, the goal of this work is to examine the affinities of these two models as well as the breadth of functionality they cover, and to define a consistent set of functionality that would fulfil the needs of CORBA Grid applications.

Integration and interoperability between CORBA and Grid applications/services can be achieved at atleast two levels – a high-level integration where CORBA interfaces are wrapped around Grid services, and a low level integration wherein CORBA services are extended (and new services added) to support Grid applications. While our final solution will combine these approaches, the design and implementation presented in this paper focuses on a high-level integration. Furthermore, the discussion in this paper concentrates on providing CORBA applications access to Grid services. Our approach however, enables true interoperability between CORBA and Grid services.

4. CORBA Interfaces to Globus Grid Services

This section describes the development of CORBA interfaces to Grid services provided by the Globus toolkit [7]. The section focuses on information management, security, remote job submission, and data access services, as these services are elementary and essential to enabling computational Grids and provide the foundation for building more advanced Grid services. The corresponding Globus services are Meta-computing Directory Service (MDS) [11], Grid Security Infrastructure (GSI) [12], Grid Resource Allocation Manager (GRAM) [13] and Globus Access to Secondary Storage (GASS) [14].

4.1 CORBA CoG Kit Architecture

In the overall CORBA CoG Kit architecture, the CORBA ORB (object resource broker) forms the middle-tier providing clients access to CORBA server objects that interface to services on the Grid. Our current implementation provides server objects for the Globus MDS, GSI, GRAM and GASS services. Each of these server objects is a wrapper around the corresponding Globus service. Clients access these server objects using the CORBA naming service, which maps names to object references. The CORBA security service is used for authenticating clients and enabling them to interact securely with server objects. The server objects notify clients of any status changes using the CORBA event service.

4.2 Grid Information Service

The Globus Meta-computing Directory Service (MDS) provides the ability to access and manage information about the state of Grid resources. The current implementation of MDS consists of a distributed directory based on LDAP [15]. A Grid application can access information about the structure and state of the resource through the uniform LDAP API. Information in the MDS is structured using a standard data model consisting of a hierarchy of entities. Each entity is described by a set of "objects" containing typed attribute-value pairs.

4.2.1 The MDSServer Object

The CORBA MDSServer object implements a CORBA object wrapper to MDS providing a simple interface with the following functionality:

1. Establishing connection to the MDS server.
2. Querying the MDS server.
3. Retrieving results from an MDS query.
4. Disconnecting from the MDS server.

The CORBA MDSServer object accesses Globus MDS using JNDI (Java Naming and Directory Interface) [16] libraries, i.e. it essentially replicates the approach used by the Java CoG Kit [1]. Fig. 2 presents the IDL used for this purpose. The data types returned by the calls to the MDSServer are very specific to the JNDI libraries. As CORBA is a language independent middleware, it is necessary to map these specific data types into a generic data type. This is achieved by the structures (i.e. *Result, ListResult, MDSList, MDSResult*) defined within the MDSServer object IDL in Fig. 2. For example, when the *getAttributes()* method is invoked on the CORBA MDSServer object, the JNDI libraries return an array of *NamingEnumeration* objects which have to be mapped into a *Result* data variable. This is done by retrieving the *id* and *attribute* for each *NamingEnumeration* object in this array as string types, and storing the string array as the value variable in the *Result* object. An array of this *Result* object forms the *MDSResult* data variable. Similarly *MDSList* data variable is created by mapping the values returned by the *search()* and *getList()* methods.

```
module MDSService {
  struct Result {string id; sequence<string> value; };
  typedef sequence<Result> MDSResult;
  struct ListResult {string id; MDSResult value; };
  typedef sequence<ListResult> MDSList;

  interface MDSServer {
    exception MDSException{string mdsMessage;
                           string ldapMessage;}};
    void connect(in string name, in long portno,
                 in string username,
                 in string password)
         raises MDSException);
    void disconnect()
         raises (MDSException);
    MDSResult getAttributes(in string dn)
         raises (MDSException);
```

```
    MDSResult getSelectedAttributes (in string dn,
                                     in Attributes attrs)
          raises (MDSException);
    MDSList getList(in string basedn)
          raises (MDSException);
    MDSList search (in string baseDN, in string filter,
                                     in long searchScope)
          raises (MDSException);
    MDSList selectedSearch (in string baseDN, in string
             filter, in Attributes attrToReturn,
             in long searchScope)
          raises (MDSException);
};};
```

Fig. 2. The IDL for accessing CORBA MDS Service.

4.2.2 Grid Domain Trading Service

The CORBA Trader Service is used to store service advertisements from remote objects. In the CORBA CoG Kit, we use the CORBA trader to provide service offers from Grid resources. For example, Fig. 3 presents the interface for a trader that returns information about the number of free nodes at a compute resource. The trader obtains this information from the MDS either using direct access or using the CORBA MDSService. More sophisticated and customized trader services can be similarly defined. These traders provide bridges between different information sources and form the basis for a more sophisticated information service within the Grid. Such a trading server has been successfully prototyped and implemented as part of [10].

```
module GlobusMachineTrader {
  struct MachineType {string dn;  string hn; string
    GlobusContact; long freenodes; long totalnodes;};
  typedef sequence<MachineType>  MachineTypeSeq;
  …
  interface GetMachineInfofromMDS {
    void update_seq();
    void initialize_trader();
    void update_trader();
    void refresh_trader();
};};
```

Fig. 3. A simple example for a CORBA trader accessing selected MDS information

4.3 Accessing Grid Security

Providing access to Grid security is an essential part of the CORBA CoG Kit. We base our current implementation on the Globus Grid Security Infrastructure (GSI) [12]. GSI provides protocols for authentication and communication and builds on the Transport Layer Security (TLS) protocols. It addresses single sign-on in virtual

organizations, delegation, integration with local security solutions, and user-based trust relations, and is designed to overcome cross-organizational security issues.

One can integrate Grid security at various levels of the CORBA architecture. In order to maintain portability across ORBs, we have not considered the modification of the protocol stack in this work, but have placed an intermediary object between the CORBA client and Grid services called the CORBA GSIServer object. This GSIServer object creates a secure proxy object, which allows other server objects, i.e. MDSServer, GRAMServer and GASSServer objects, to securely access corresponding Globus services. The creation of the secure proxy object consists of the following steps:

1. The client and the CORBA server mutually authenticate each other using the CORBA security service (CORBASec) [17][18]. One of the basic requirements for mutual authentication in CORBASec is to have private credentials i.e. a public certificate signed by a trusted certificate authority (CA), at both the client and server side. In our architecture both the CORBA client and server use Globus credentials where the trusted certificate authority is Globus CA.
2. As Globus services, such as gatekeeper [13] and gasserver [14], only accept connections from clients with secure Globus credentials, the CORBA client delegates the GSIServer object to create a secure proxy object that has the authority to communicate with the gatekeeper/gasserver on the clients' behalf.
3. After successful delegation, the GRAMServer and GASSServer objects use the secure proxy object to set up secure connections to the corresponding Globus servers (gatekeeper/gasserver) and access required Globus services.

The process of delegation from the CORBA client to the CORBA GSIServer object involves the following steps. First, the client sends over its public certificate in an encoded form to the server object. Next, the server object generates a completely new pair of public and private keys and embeds the new public key and the subject name from the client certificate in a newly generated certificate request. The certificate request is signed by the new private key and sent across to the client. The client retrieves the public key from the certificate request and embeds it a newly generated certificate. This new certificate is called a proxy certificate. It is signed by the client's original private key (not the one from the newly generated pair), and is sent back to the server object in an encoded form. The server object thus creates a chain of certificates where the first certificate is the proxy certificate, followed by the client certificate and then the certificate of the CA. It can then send this certificate chain to the gatekeeper as proof that it has the right to act on behalf on the client. The gatekeeper verifies the chain by walking through it starting with the proxy certificate, searching for trusted certificates and verifying the certificate signatures along the way. If no trusted certificate is found at the base of the chain the gatekeeper throws a *CertificateException* error. The IDL interface for the GSISever object is shown in Fig 4. Its methods are described below:

- setClientCredentials(): This method is called by the client to send its public certificate to the server in an encoded form. The client can access this method only after mutual authentication has been successful.
- getCertificateRequest(): This method provides the client access to the certificate request generated at the server end.

- setDelegatedCertificate(): Using the certificate request obtained from the server, the client generates a new certificate called the proxy certificate for delegating to the server the right to access the Globus services on its behalf. By invoking this method the client can send this proxy certificate in an encoded form to the server.

```
module GSIService {
  interface GSIServer{
    typedef sequence<octet> ByteSeq;
    void setClientCredentials(in ByteSeq certificate);
    ByteSeq getCertificateRequest();
    void setDelegatedCertificate(in ByteSeq
                                 certificate);
};};
```

Fig. 4. The IDL for accessing CORBA GSI Service.

4.4 Job Submission in a Grid

Remote job submission capabilities are provided by the GRAMServer object using the Globus GRAM service as described below.

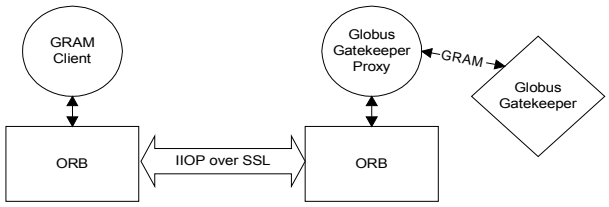

Fig. 5. Accessing a Globus Gatekeeper from a CORBA client.

Job submission using GRAM consists of the following steps: First, the client authenticates with the CORBA server object using CORBASec. After mutual authentication is successful, the client subscribes to the CORBA event channel on which the server is listening. Next the client gets a handle to the GSIServer object from the naming service and delegates the CORBA GSIServer object as described in the process above. Once delegation is successful, the client obtains a reference to GRAMServer object (using the CORBA naming service) and submits a job submission request specifying the name of the executable and the name of the resource on which the job is to be executed. On receiving the request, the GRAMServer uses the secure proxy object created by GSIServer during delegation to set up a secure connection with the GRAM gatekeeper. It then forwards the request to the gatekeeper and waits for status updates from the job via the CORBA event channel. The implementation of the GRAMServer object in the CORBA CoG Kit provides a simple interface with the following methods (see Fig. 6):

- setProxyCredentials() : This method sets the reference to the proxy secure object created by the GSIServer object.
- jobRequest(): This method is used by the client to request a job submission on a remote resource.

Additionally the following data structure is used to monitor the status of the job:

- JobStatus: This data structure is used by the CORBA event service to notify the client of changes in the job status. The structure consists of two string data types – *jobid* and *jobstatus*. *jobid* identifies the id of the submitted job and *jobstatus* is one of the following values – PENDING, DONE, ACTIVE, FAILED, or SUSPENDED.

```
module GRAMService {
  exception GramException{short errorcode;};
  exception GlobusProxyException{short errorcode;};
  struct JobStatus{string jobid;string currstatus;};
    interface GRAMServer{
          void setProxyCredentials();
          void jobRequest(in string rsl, in string
          contact, in boolean batchjob);
};};
```

Fig. 6. The IDL for accessing CORBA GRAM Service

4.5 Data Transfer on the Grid

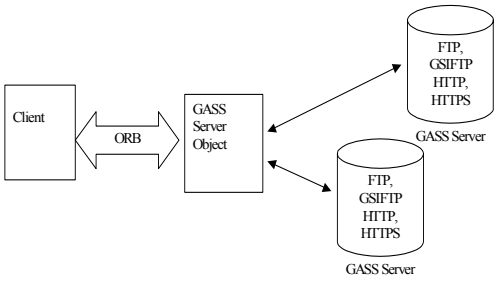

Fig. 7. The CORBA CoG Kit interface to GASS.

A frequent problem that needs to be addressed by Grid applications is access to remote data -- for example, when the application may want to pre-stage data on remote machines, cache data, log remote application output in real time or stage executables on a remote computer. In our current implmentation we use Globus GASS [14] for data transfer between resources on the Grid. The goal of GASS is not to build a general-purpose distributed file system but to support I/O operations commonly required by Grid applications. The strategy employed is to fetch the file and cache it on first read open, and write it to disk when it is closed.

The objective of the CORBA GASSServer object is to provide an interface to the Globus GASS service as shown Fig 7. The client gets a handle to the GASSServer object from the naming service, and then the server object forwards the request to the appropriate GASS servers using the protocol specified by the client. GASS supports FTP, HTTP, HTTPS, and GSIFTP. Both the FTP and GSIFTP protocol allows third-party file transfers; that is they allow file transfers from a sender machine to a receiver machine to be initiated by an third initiator machine. Both the sender and receiver machines have to provide a GASS server. Authentication is performed using GSI. The methods defined by the CORBA GASSServer object is defined in the IDL as shown in Fig. 8.

```
module GASSService {
   interface GASSServer {
        void setProxyCredentials();
        void setSourceURL(in string sourceurl);
        void setDestinationURL(in string destnurl);
        void allowThirdPartyTransfer(in boolean value);
        void URLcopy()
   };
};
```

Fig. 8: The IDL for accessing CORBA GASS Service.

5. Applications

We believe that many applications can benefit from the CORBA CoG Kit presented in this paper. One example is the Numerical Propulsion System Simulation (NPSS) [10], which is part of NASA IPG and provides an engine simulation using computational fluid dynamics (CFD). It consists of 0 to 3-Dimensional engine component models responsible for examining aerodynamics, structures and heat transfer. Previous studies have shown that NPSS's engine components can be encapsulated using CORBA to provide object access and communication from heterogeneous platforms, while at the same time enable coordination of multiple modeling runs across the Grid. In this application a large number of NPSS jobs (1000+) are submitted from a desktop interface using the CORBA CoG Kit. The longer-term goal of the application is to deploy computationally intense (3-Dimensional) NPSS jobs across the Globus-enabled NASA Information Power Grid (IPG). The primary benefit of the CORBA CoG Kit to this project is being able to access Globus functionality directly from the CORBA application.

6. Status

The current implementation of the CORBA CoG Kit provides server objects for MDS, GSI, and GRAM services. The performance of the CoG implementation with different

ORBs is currently being evaluated. We have also made significant progress in integrating the CORBA CoG Kit with DISCOVER[19], a collaboratory for interaction and steering. The current status of the CORBA CoG project and the software can be obtained from http://www.caip.rutgers.edu/TASSL/CorbaCoG/CORBA/.

7. Conclusion

This paper reports on an ongoing project aimed at designing, implementing and deploying a CORBA CoG Kit. The overall goal of this project is to provide a framework that will enable existing Grid Computing Environments and CORBA Service Providers to interoperate. CORBA is an accepted technology for building distributed applications and is widely used and supported by academia and industry. Its features include a high-level modular programming model, availability of advanced services (e.g. security, naming, trading, event, transaction, etc.) and readymade solutions, interoperability, language independence, location transparency and an open standard, making it a suitable candidate for developing Grid applications. Developing a CORBA CoG Kit facilitates this integration. The demand for such a CoG Kit has been expressed by various projects ranging from the creation of CORBA based control systems for advanced instruments to the collaborative interaction and computational steering of very large numerical relativity and fluid dynamics applications. Our current efforts are focused on enabling applications to combine and compose services on the Grid – e.g. combining services provided by Globus, with the collaborative monitoring, interaction, and steering capabilities provided by DISCOVER [19]. For example a scientific application can use CORBA CoG Kit to discover the available resources on the network, use the GRAM Service provided by CoG to run his simulation on the desired high end resource, and use DISCOVER web-portals to collaboratively monitor, interact with, and steering the application.

8. Acknowledgement

We would like to acknowledge Brian Ginsburg, Olle Larsson, Stuart Martin, Steven Tuecke, David Woodford, Isaac Lopez, Gregory J. Follen, Richard Gutierrez and Robert Griffin for their efforts towards providing a C++ based CORBA interface for the NPSS application performed at the NASA Glenn Research Center. We would also like to thank Kate Keahey and Nell Rehn for valuable discussions.
This work was supported in part by the National Science Foundation under Grant Number ACI 9984357 (CAREERS) awarded to Manish Parashar, by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38; by the Defense Advanced Research Projects Agency under contract N66001-96-C-8523, and by the NASA Information Power Grid program.

9. References

[1]G. v. Laszewski, I. Foster, J. Gawor, and P. Lane, "A Java Commodity Grid Kit," Concurrency and Computation: Practice and Experience, vol. 13, pp. 643-662, Issue 8-9, 2001. http://www.globus.org/cog/documentation/papers/

[2]V. Getov, G. v. Laszewski, M. Philippsen, and I. Foster," Multi-Paradigm Communications in Java for Grid Computing," ACM Communications, October 2001 (to appear). http://www.globus.org/cog/documentation/papers/

[3]"The Grid Portal Development Kit," 2001, http://dast.nlanr.net/Features/GridPortal/.

[4]"The Python CoG Kit," 2001, http://www.globus.org/cog.

[5]"The Perl CoG Kit," 2001, http://hotpage.npaci.edu.

[6] CORBA: Common Object Request Broker Architecture, http://www.omg.org.

[7] I. Foster, and C. Kesselman, "Globus: A Metacomputing Infrastructure," International Journal of Supercomputer Applications, 11(2): pp. 115-128, 1997

[8]I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," International Journal of Supercomputing Applications, 2001 (to appear). http://www.globus.org/research/papers/anatomy.pdf.

[9]Y. Wang, F. D. Carlo, D. Mancini, I. McNulty, B. Tieman, J. Bresnahan, I. Foster, J. Insley, P. Lane, G. v. Laszewski, C. Kesselman, M.-H. Su, and M. Thiebaux, "A high-throughput x-ray microtomography system at the Advanced Photon Source," Review of Scientific Instruments, vol. 72, pp. 2062-2068, 2001.

[10] Numerical Propulsion System Simulation NPSS),http://hpcc.lerc.nasa.gov/npssintro.shtml.

[11]K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman, "Grid Information Services for Distributed Resource Sharing," Proc. 10th IEEE International Symposium on High Performance Distributed Computing, August 2001.

[12]I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke, "A Security Architecture for Computational Grids," Proc. 5th ACM Conference on Computer and Communications Security Conference, pp. 83-92, 1998.

[13]K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke, "A Resource Management Architecture for Metacomputing Systems," Proc. IPPS/SPDP'98 Workshop on Job Scheduling Strategies for Parallel Processing, 1998.

[14]J. Bester, I. Foster, C. Kesselmaz, J. Tedesco, and S. Tuecke, "GASS: A Data Movement and Access Service for Wide Area Computing Systems," 6th Workshop on I/O in Parallel and Distributed Systems, May 1999.

[15]Netscape Directory and LDAP Developer Central, http://developer.netscape.com/tech/directory/index.html.

[16]JAVA Naming and Directory Interface (JNDI), http://java.sun.com/products/jndi. V1.2.

[17]U. Lang, D. Gollmann, and R. Schreiner, "Security Attributes in CORBA," Submitted to IEEE Symposium on Security and Privacy, 2001.

[18]B. Blakley, R. Blakley, and R. M. Soley, "CORBA Security: An Introduction to Safe Computing With Objects," The Addison –Wesley Object Technology Series

[19]S. Kaur, V. Mann, V. Matossian, R. Muralidhar, and M. Parashar, "Engineering a Distributed Computational Collaboratory," Accepted for publication at the 34th Hawaii Conference on System Sciences, January 2001.