DESIGN AND IMPLEMENTATION OF CORBA COMMODITY GRID KIT

BY SNIGDHA VERMA

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

Professor Manish Parashar

and approved by

New Brunswick, New Jersey

May, 2002

ABSTRACT OF THE THESIS

Design and Implementation of CORBA Commodity Grid Kit

by Snigdha Verma

Thesis Director: Professor Manish Parashar

This thesis presents the design, implementation, evaluation and deployment of a CORBA Commodity Grid (CoG) Kit. The overall goal of this thesis is to enable the development of advanced Grid applications while adhering to state-of-the-art software engineering practices and reusing the existing Grid infrastructure. As part of this activity, we are investigating how CORBA can be used to support this software engineering task. In this thesis, we outline the design of a CORBA Commodity Grid Kit that will provide a software development framework for building a CORBA "Grid domain." We also present our experiences in developing a prototype CORBA CoG Kit that support the development and deployment of CORBA applications on the Grid by providing them access to the Grid services provided by the Globus Toolkit.

Acknowledgements

I would like to thank my research advisor, Prof. Manish Parashar, for his guidance and support throughout my study as a graduate student. I would also like to thank Gregor von Lazweski and Jarek Grawor from Argonne National Lab, Illinois for patiently answering all my queries and helping me in the design of my work. I would especially like to thank Viraj Bhat for helping me in the experiments.

I would like to acknowledge my colleagues at TASSL lab and friends here in US and in India for their love and support during my studies.

I would also like to thank Kanishka Lahiri and Kundan Sen who helped me write my thesis in latex.

Finally, I thank my family for their love and constant encouragement throughout my life.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Problem Statement

The past decade has seen the emergence of computational Grids infrastructures aimed at allowing the programmer to aggregate powerful and sophisticated resources scattered around the globe. To enable this goal, the Grid computing community has concentrated on the creation of advanced services that allow access to high-end remote resources such as batch systems at supercomputing centers, large-scale storage systems, large-scale instruments, and remote applications. This effort has resulted in the development of Grid services that enable application developers to authenticate, access, discover, manage, and schedule remote Grid resources, but they are often incompatible with commodity technologies. As a result, it is difficult to integrate these services into the software engineering process adopted by most application developers. At the same time, considerable advances have been made in developing and refining commodity technologies for distributed computing. One such effort is the Common Object Request Broker Architecture (CORBA) [5] defined by the Object Management Group (OMG). OMG is an independent consortium of vendors; consequently, the standard it defined is open (vendor independent) and has resulted in many independent implementations for a variety of platforms. In order to provide interoperability between these diverse implementations, CORBA defines interoperability mechanisms. A high-level, distributed computing model, vendor independence, and a strong interoperability thrust all combined to make CORBA an attractive and popular distributed computing standard. As such CORBA meets the necessary requirements to be seriously considered by application developers as part of the Grid infrastructure.

1.2 Objective

Recently, a number of research groups have started to investigate Commodity Grid Kits (CoG Kits) in order to explore the affinities of the Grid and commodity technologies. Developers of CoG Kits have the common goal of developing mappings and interfaces between Grid services and a particular commodity technology. We believe that CoG Kits will encourage and facilitate the use of Grid technologies, while at the same time leveraging the benefits of the commodity technology. Currently, CoG Kits are being developed for the Java platform, Java Server Pages, Python, and Perl. This thesis will describe our experiments in defining yet another CoG Kit: one allowing CORBA applications to access (and provide) services on the Grid. Such integration would provide a powerful application development environment for high-end users and would create a CORBA "Grid domain".

1.3 Organization of the Thesis

Chapter 2 provides a brief overview of the Grid and its architecture followed by the introduction of the services and protocols that are going to be integrated in the CORBA CoG Kit. It also looks into other CoG Kits that have been developed by various other research groups. Then a brief introduction to the CORBA technology and its advantages and disadvantages from the viewpoint of Grid developers. Chapter 3 covers the design of the CORBA CoG Kit architecture and the implementation of the four basic services that provide access to Grid functionality. Chapter 4 describes the experiments and the results in details thus evaluating the software. The last chapter draws conclusion on the work and discusses the future work that can be done by integrating the software in other applications and other services which can be developed for enhancing the software.

Chapter 2

Background and Related Work

2.1 The Grid

The term "Grid" emerged in the past decade to denote an integrated distributed computing infrastructure for advanced science and engineering applications. The fundamental Grid concept is based on coordinated resource sharing and problem solving in dynamic multi-institutional virtual organizations [18]. Besides access to a diverse set of remote resources, services, and applications among different organizations, Grid computing is required to facilitate highly flexible sharing relationships among these organizations ranging from client-server to peer-to-peer. An example of a typical Grid client-server relationship is a supercomputer center in which a client submits jobs to the supercomputer batch queue. An example for peer-to-peer computing is the collaborative online steering of high-end applications, as demonstrated by the use of advanced instruments [28, 41]. Grids must support different levels of control, ranging from fine-grained access control to delegation, single user to multi-user and collaborations, 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 use of resources at multiple sites for computation. To date the Grid community has developed protocols, services, and tools that address the issues arising from sharing resources, service, and applications in peer communities. The community is also addressing security solutions that support management of credentials and policies when computations span multiple institutions, secure remote access to compute and data resources, and information query protocols that provide services for obtaining the configuration and status information of the resources. Because of the diversity of the Grid, however, it is difficult to develop an all-encompassing Grid architecture.

Recently, a Grid architecture has been proposed [18] that comprises five layers:

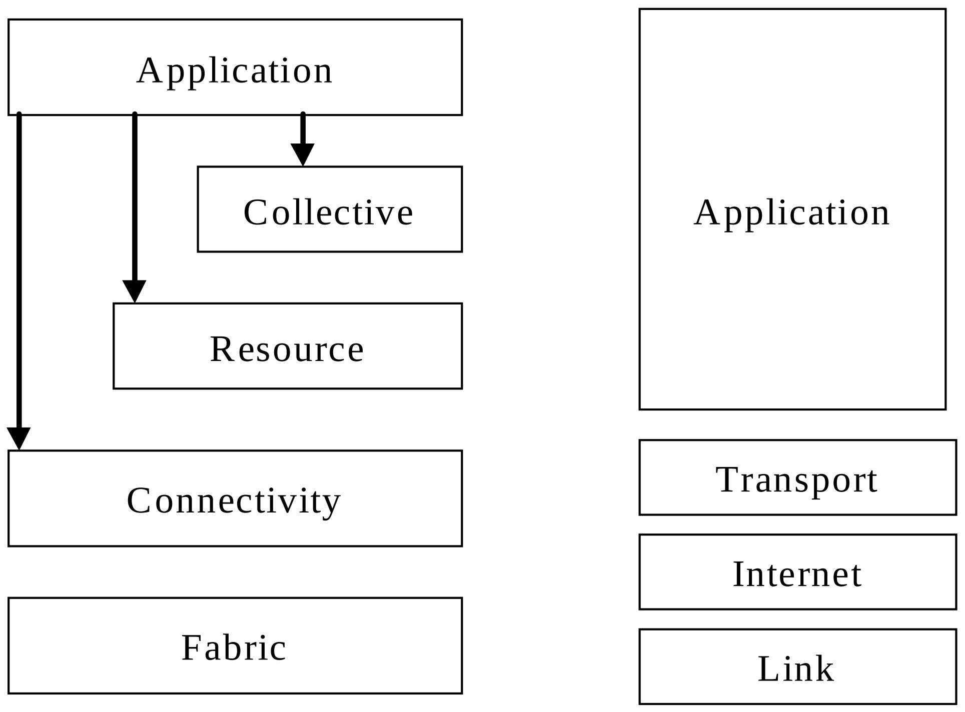


Figure 2.1: The Grid Architecture

1. A fabric layer, which interfaces to local control including physical and logical resources such as systems, files, or even a distributed file system.

2. A connectivity layer, which defines core communication and authentication protocols supporting Grid specific network transactions.

3. A resource layer, which allows the sharing of a single resource.

4. A collective layer, which allows resources to be viewed and operated on as collections.

5. An application layer, which uses the appropriate components of each layer to support applications.

Each of these layers may contain protocols, APIs, and Software Development Kits (SDK) to support the development of Grid applications. This general layered architecture of the Grid is shown in the left part of Figure 2.1.

2.2 Commodity Technologies

The past few years have seen an unprecedented level of innovation and progress in commodity technologies. Three areas have been critical in this development: the Web, distributed objects, and databases. Each area has developed impressive and rapidly improving software artifacts. Examples at the lower level include Hypertext Markup Language (HTML), Hypertext Transfer Protocol (HTTP), MIME, Internet Inter-ORB Protocol (IIOP), Common Gateway Interface (CGI), Java, JavaScript, JavaBeans, Common Object Request Broker Architecture (CORBA), Common Object Model (COM), ActiveX, Virtual Reality Markup Language (VRML), object broker ORBs, and dynamic Java servers and clients. Examples at the higher level include collaboration, security, commerce, and multimedia technologies. The most important contribution of commodity technologies is the set of open interfaces that enable large components to be quickly integrated into new applications.

2.3 Commodity Grid Kits

The advances in "Grid" and "commodity" technologies have evolved in parallel, with different goals leading to different emphases and technology solutions. For example the commodity technologies focus on issues of component, scalability and presentation while the Grid emphasizes on end-to-end performance, advanced network services, and support for unique resources such as supercomputers. These two technologies could complement each other but there is an obvious gap which needs to be bridged. The Commodity Grid Kit project is one such effort to bring the world of commodity and Grid computing together.

Definition: "A Commodity Grid Toolkit (CoG Kit) defines and implements a set of general components that map Grid functionality into a commodity environment/framework."

2.3.1 Java CoG Kit

The first activity that was undertaken in building a CoG kit was a Java CoG Kit [11, 39] at Argonne National Laboratory. The Kit currently provides most of the client-side functionality of Globus [13]. Some of these capabilities include: the Grid Security Infrastructure, which enables secure authentication and communication over an open network; the Resource Specification Language, a method for exchanging information about resource requirements between all of the components in a resource management architecture; the Globus Access to Secondary Storage module, which allows applications to access data stored in any remote file system; and the Metacomputing Directory Service, which provides the tools to build information infrastructures for computational grids.

2.3.2 Perl CoG Kit

A parallel effort is being taken at University of San Diego to develop a Perl Commodity Grid Kit [34]. It is intended to be used by any developer who wants to use the Perl programming language and also has a need to use Grid services in the application. For example web portals are good examples that can use grid services via the Perl CoG Kit. Similiar to the Java CoG Kit, the Perl CoG Kit provides interface to Globus. It contacts the Globus gatekeeper on remote machines to submit jobs and uses the Globus MDS to gather information. In some cases it uses the Globus client utilities to make request to the Globus server utilities. Some of the modules are wrappers around the binary executables such as 'globusrun' and 'globus-job-submit'. It uses the 'grid-proxy-init' to implement much of the security functionality. For installation purposes it provides a makefile and test script for each of the modules.

2.3.3 Python CoG Kit

The development of Python CoG Kit [22] is being undertaken at Berkeley National Laboratory. It provides a Python based high level interface to the Grid Services. The key advantages of Python are that it is an object oriented programming language with automatic memory management. It provides support to XML processing, SOAP, and MPI. The interfaces provide the use of native extension modules in Python which provide a clean interface to the Globus C code and mapping the underlying C code to a natural Python idiom. For example, C returns an int value as status code and uses pointers to return multiple values. In Python, functions return multiple values. The CoG Kit provides modules to GRAM, GridFTP, GassCopy and Secure IO. It has future plans to integrate replica catalog and replica management packages for data intensive projects.

2.3.4 JAVA Server Pages CoG Kit

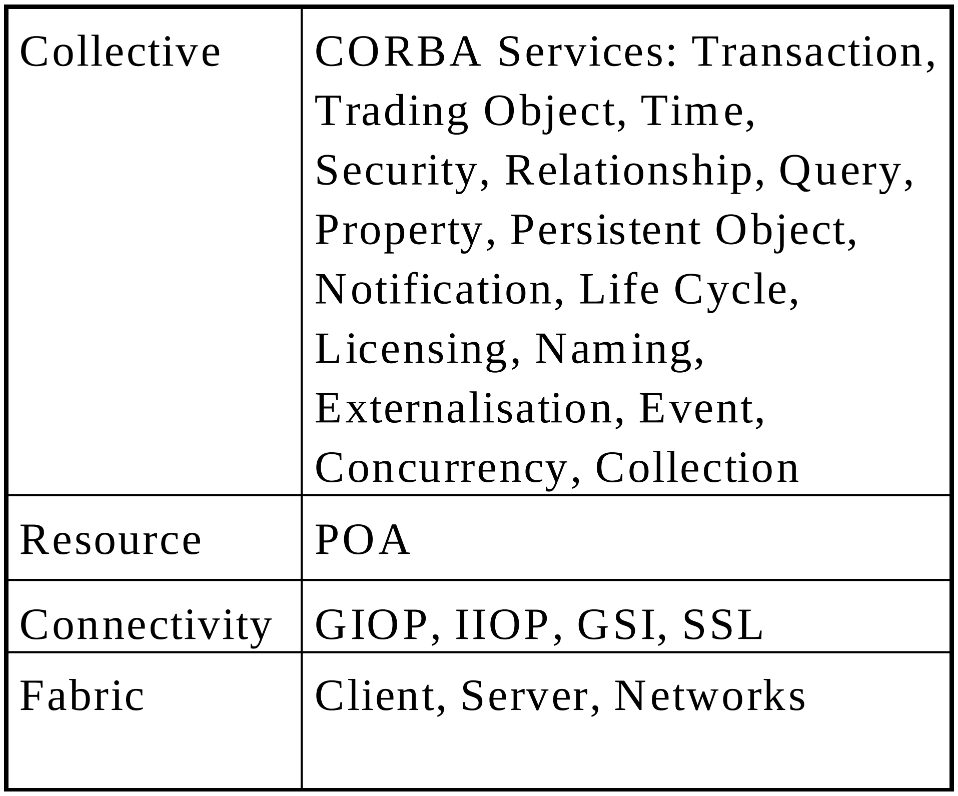
The development of JAVA Server Pages Commodity Kit [26] is a collaboration effort of NCSA, SDSC, and NASA IPG. It provides a common set of components and utilities that make portal development easier and allow various portals to interoperate by using the same core infrastructure.

2.4 CORBA Commodity Grid Kit

Researchers have expressed a growing interest in combining the functionality of Grid technologies and CORBA as CORBA services can support the Grid architecture as is obvious from Figure 2.1 and Table 2.1. With the initiative of Gregor von Laszewski [14] at Argonne National Lab, Illinoi we started looking into the development of a CORBA CoG Kit. One of the key benefits of CORBA is that it enables software objects to transparently call remote objects across networks. This is facilitated by the remote method invocations through an *Object Request Broker*(ORB) which hides the complexity of distributed applications. The ORB can be regarded as a "software bus", analogous to hardware bus which provides hardware devices with an abstract interface to the communications mechanism. The actual functionality is implemented by the ORB libraries. The methods that can be invoked on the remote objects are specified in a standardized *Interface Definition Language*(IDL) and objects can be located with *Interoperable Object References*(IORs). CORBA also provides other services such as naming service, event service, persistence service and security service.

There are additional reasons why CORBA appeals to users; some of the most important ones are as follows:

- High-level, modular programming model: The CORBA interaction model, as well as its services provides a convenient environment for distributed computation; CORBA hides the complexities of networking, and provides security mechanisms and other ready-made solutions. Programming in CORBA not only speeds the development process but also results in systems with a high reusability potential.



Collective	CORBA Services: Transaction, Trading Object, Time, Security, Relationship, Query, Property, Persistent Object, Notification, Life Cycle, Licensing, Naming, Externalisation, Event, Concurrency, Collection
Resource	POA
Connectivity	GIOP, IIOP, GSI, SSL
Fabric	Client, Server, Networks

Table 2.1: Mapping of various CORBA related technologies into the Grid layers

- Interoperability of heterogeneous components: Components implemented in different languages can interact by specifying interfaces in the Interface Definition Language (IDL).

- Location transparency: The CORBA distributed computing model hides the fact that two components may be interacting remotely.

- Open standard: CORBA is vendor independent, which results in many implementations over many diverse platforms.

- Interoperability: CORBA defines mechanisms that allow solutions from different vendors to interoperate.

- Legacy integration: Legacy applications can be cast as CORBA objects [36].

The interest in CORBA within the Grid community has led to a number of applications seeking [33, 16, 36] to combine the functionality of CORBA and Globus [13]. Although these solutions work well to solve specific problems encountered in individual applications, they lack generality and uniformity of approach. The different CORBA Grid solutions are not necessarily compatible with each other, and they require programmers to frame their solutions in terms of two different programming models that are not always consistent. The purpose of our 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 fulfill the needs of CORBA Grid applications. The Kit currently provides most of the client-side functionality of Globus. Some of these capabilities include: the Grid Security Infrastructure, which enables secure authentication and communication over an open network; the Resource Specification Language, a method for exchanging information about resource requirements between all of the components in a resource management architecture; the Globus Access to Secondary Storage module, which allows applications to access data stored in any remote file system; and the Metacomputing Directory Service, which provides the tools to build information infrastructures for computational grids.

We have identified two key scenarios in which users may want to combine the functionality of the Grid technologies and CORBA:

1. A CORBA programmer may want to combine the CORBA programming model and CORBA services with the functionality provided by the Grid.

2. A Grid programmer may want to access CORBA services not provided by the Grid.

While we eventually plan to address both scenarios in this thesis we will focus on providing a high-level CORBA interface to the Grid. We will begin by defining a CORBA Grid computing model and making Grid services accessible through the CORBA programming interface.

As CORBA defines both high-level interoperability through high-level bridges and low-level interoperability through IIOP and acknowledges their respective advantages and disadvantages, we believe that our work on creating interoperability between the Grid and CORBA architectures can benefit from such a dual approach. Therefore, we will pursue both two lines of investigation: a high-level approach and a low-level approach. In the former, adding Grid functionality to CORBA is achieved by wrapping CORBA interfaces around key Grid services as described in this thesis. The advantages of this approach are simplicity, modularity (i.e, the programmer can use a subset of Grid services and functionality fulfilling the application requirements), and the speed with which a system can be implemented and deployed. The disadvantages are that the approach is not exposing all features of CORBA. For example, let us consider the security service; in this approach the mechanisms for Grid security are implemented using security services present in CORBA and the security model of GSI, essentially requiring the presence of two largely overlapping security models. In the low-level approach the CORBA programming model is overlaid on a Grid-based implementation. In this case the CORBA security service (including its interface, if necessary) is extended to include Grid-based functionality. Within this approach we plan to experiment with uniting models for specific services rather than presenting them as external components. For example, rather than translate between two different security models we will consider if they can interoperate at a lower level presenting consistent interface to the user. Similarly, rather than present GRAM [7] as an external service, we will consider how it might fit within CORBA activation mechanisms. The advantages of this approach are that the programmer would deal with one consistent model available through familiar interfaces (CORBA mechanisms). The disadvantages are that this approach is harder to implement and may involve extending many of the CORBA facilities beyond the standard as defined today.

We believe that our final solution will incorporate both approaches to combining CORBA and Grid. The low-level approach will provide the best, and in many cases also the most efficient solution whenever critical functionality, or services present in both Grid and CORBA, need to be combined. On the other hand the high-level approach is appropriate when optional Grid-specific functionality (for example, a replica service) needs to be represented. We anticipate that our final solution will contain three kinds of services: pure CORBA services (for example, the Persistent State); combined Grid CORBA services (for example, the security, as well as ORB functionality, Object Adapters and other critical services with counterparts in both Grid and CORBA); and pure Grid services (for example, a Replica Service).

Chapter 3

CORBA Commodity Grid Kit

3.1 Architecture

A schematic view of the architecture of the CORBA CoG Kit is shown in Figure 3.1. The CORBA orb forms the middletier, providing clients access to CORBA objects that implement services on the Grid. The current implementation includes Grid services provided by the Globus project [13, 17]. The Globus project is developing fundamental technologies needed to build computational grids [18]. Grids are persistent environments that enable software applications to integrate instruments, display computational and information resources that are managed by diverse organizations in widespread locations. It provides an authentication service as part of the Grid Security Infrastructure (GSI) [10], an information service called Metacomputing Directory Service (MDS) [6, 9], a job submission service called Grid Resource Allocation Manager (GRAM) [7], and data storage and access service called Globus Access to Secondary Storage (GASS) [4]. A set of CORBA objects have been developed that interact implicitly with the appropriate Grid services. Clients access these CORBA objects using the CORBA naming service, which maps names to object references. The CORBA security service is used for authenticating the clients, and for enabling them to interact securely with the CORBA objects. The CORBA objects notify clients of any status changes through the CORBA event service. In the future the CORBA CoG Kit will be expanded to provide CORBA objects for other services on the Grid such as DISCOVER [8, 28], or NetSolve [2]. The goal is to provide uniform access to a pool of services that can be used and composed by the user applications.

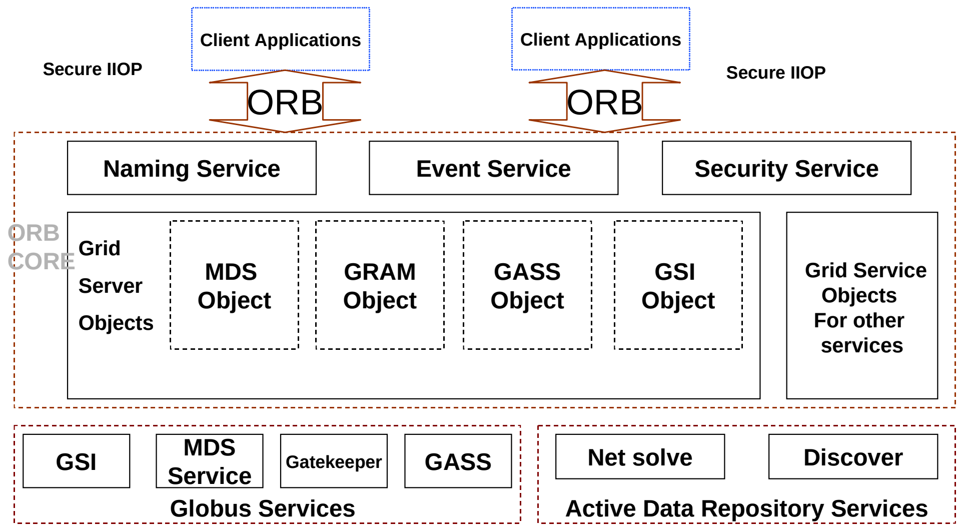


Figure 3.1: The CORBA CoG Kit Architecture

3.1.1 Corba Interfaces to Globus Grid Services

This section briefly describes the overall architecture and then the next few sections present the interfaces and mechanisms used by the CORBA CoG Kit to provide access to various Grid services.

Concepts and Terminology

Like all technologies CORBA has unique terminology associated with it. Understanding these terms and terminology will help in understanding the CORBA CoG Kit architecture.

- A *CORBA object* is a “virtual” entity which is capable of being located by an ORB and having client requests invoked on it. It is virtual in the sense that it does not really exist unless it is made concrete by an implementation written in a programming language. The realization of a CORBA object by programming language constructs is analogous to the way virtual memory does not exist in an operating system but is simulated using physical memory.
- A *target object*, within the context of a CORBA request invocation, is

the CORBA object that is the target of the request. The CORBA object model is a *single-dispatching* model in which the target object for a request is determined solely by the object reference used to invoke the request.

- A *client* is an entity that invokes a request on a CORBA object. A client may exist in an address space that is completely separate from the CORBA object or the client and the CORBA object may exist within the same application. The term client is meaningful only within the context of a particular request because the application that is the client for one request may be the server for another request.
- A *server* is an application in which one or more CORBA objects exist. As with clients this term is meaningful only in the context of a particular request.
- A *request* is an invocation of an operation on a CORBA object by a client. Requests flow from a client to the target object in the server, and the target object sends the results back in a response if the request requires one.
- An *object reference* is a handle used to identify, locate and address a CORBA object. To clients, object references are opaque entities. Clients use object references to direct requests to objects but they cannot create object references from their constituent parts nor can they access or modify the contents of an object reference. An object reference refers only to a single CORBA object.
- A *servant* is a programming language entity that implements one or more CORBA objects. Servants are said to incarnate CORBA objects because they provide bodies or implementations, for those objects. Servants exist within the context of a server application. For e.g. in C++ and Java servants are object instances of a particular class.

In the CORBA CoG Kit architecture, as shown in Figure 3.1 the CORBA objects that are interacting with the respective Grid Services are MDS, GRAM, GASS, and GSI objects. When the client application makes a request, the request flows through the orb and the server application receives it. The server dispatches the request to the respective servant of the CORBA object. After the servant CORBA object carries out the request, it returns the response to the client application. To obtain the object reference of these CORBA objects, the CORBA Naming Service is used. The CORBA naming service is the simplest and most basic of the standardized CORBA services. It provides a mapping from names to object references; given a name the service returns an object reference stored under that name. It is similiar to Internet Domain Service(DNS) which translates Internet domain names into IP addresses. The naming service provides a number of advantages to clients.

- Clients can use meaningful names for objects instead of having to deal with stringified object references.
- The naming service can be used to solve the problem of how the application components get access to the initial references for an application. Advertising these references in the Naming Service eliminates the need to store them as stringified references in files.

In the naming service, a name-to-reference association is called a *name binding*. A *Naming Context* is an object that stores name binding in form of a table that maps names to object references. A name in the table can denote either an object reference to an application object or another context object in the Naming Service. A collection of contexts and bindings is known as *naming graph*. The Figure 3.2 below shows the naming graph of the CORBA CoG Kit architecture.

Here the hollow nodes are the naming contexts and the solid nodes are the CORBA objects. These CORBA objects are always the leaf node. The root naming context is called Globus and each of the CORBA objects implementing the service is binded to the Globus context by creating a new name binding for e.g the CORBA object implementing the Gram Service is binded to the Globus context using the name binding called GramService. The code as shown in Figure **??** shows how the various CORBA objects are binded to the Naming Service.

The key advantage of this architecture is it's scalability. New CORBA objects interfacing various other Grid services can be developed and easily binded to the naming graph of

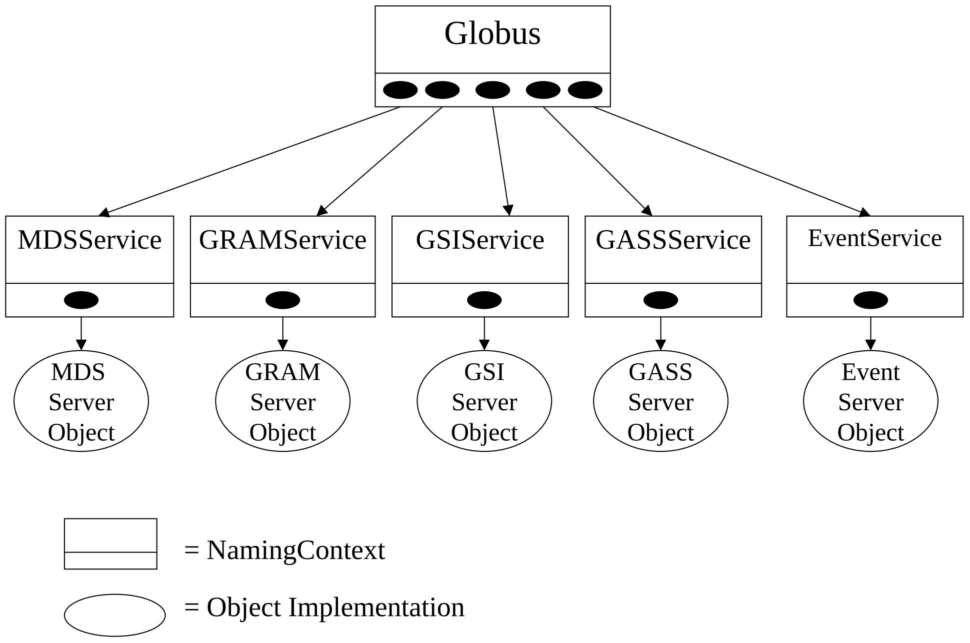


Figure 3.2: `Naming graph of CORBA CoG Kit architecture`

CORBA CoG Kit without any significant changes.

3.2 CORBA CoG Directory Service

3.2.1 Directory Service

High performance distributed computing applications often require the careful selection and configuration of computers, networks, application protocols, and algorithms. These requirements do not come up in traditional client server application where standard default protocols and interfaces are used. In high performance computing applications, the systems required are generally homogenous and can be manually configured. But in distributed computing applications information rich approach to configuration is used where decisions are made based upon the structure and the state of the system on which a program is to run.

3.2.2 Metacomputing Directory Service

The Metacomputing Directory Service (MDS) in Globus provides the ability to manage and access information about the state of a Grid. As such it enables read access to entities such as computer, networks and people. The current implementation of MDS as part of the Globus toolkit is based on a distributed directory based on LDAP technology [31]. A high-end application can access information about the structure and state of the system through the uniform LDAP API. The information is organized in MDS as well defined collection called entries. The entries are organized in a hierarchical tree structured name space called Directory Information Tree (DIT). Any of the MDS entry can be referred by its unique name called Distinguished Name, which is constructed by specifying the path from the root to the entry being named. These entities represent an instance of an object. The information in an entry is represented by one or more attributes consisting of name and corresponding value. The attributes depend on the object type that the entity is representing. The object type information is encoded in the MDS Data Model. This data model specifies the data hierarchy and the object classes used to define each type of entry.

3.2.3 CORBA MDS Service

In order to support information services in the Grid an interface to MDS is developed. Although it is possible to develop a COS naming service to access objects stored within the MDS, it is problematic as object definition in the LDAP data model are created at time of instantiation. Thus, it is much easier to provide a direct interface to the MDS, returning objects in the same fashion as the Java CoG Kit. This approach is useful for those familiar with Grid services.

CORBA CoG MDS Server Object

The CORBA MDS server object implements a simple interface as shown in Figure 3.3 that provides the following functionality:

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

```
    module MDS
    {
      interface MDSServer
      {
        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);
      }
    }
end
```

Figure 3.3: `Interface to CORBA CoG MDS Service`

At the backend, the CORBA MDS server object accesses Globus MDS using JNDI (Java Naming and Directory Interface) [27] libraries, i.e. it replicates the approach used by the Java CoG Kit [11, 39]. JNDI is an interface rather than an implementation. As such it needs to access an existing naming service. JNDI performs all naming operations relative to a context. A *Context* in JNDI represents a set of bindings within the naming service that all share the same naming convention. A *Context* object provides the method for binding names to objects and unbinding names from objects, for renaming objects and for listing the bindings. To assist in finding a place to start, the JNDI specification defines an *InitialContext* class. This class is instantiated with properties that define the type of naming service in use and, for naming services that provide security, the ID and password to use when connecting. A new InitialDirContext object is created with default properties. The username and password is obtained from the client application. The getAttributes() method returns the attributes associated with the specified entry. The Attribute class represents a collection of attributes; it contains instances of the Attribute class, which by itself represents a single attribute. The getAttributes() method returns all attributes for the specified entry and getSelectedAttributes() method returns only those attributes which match the attributes specified in the function parameter. Similiarly for the search() method it returns NamingEnumeration class. The data types returned by the calls to the MDS server are very specific to the JNDI libraries. Since 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 MDS server object. For example, when the getAttributes() method is invoked on the CORBA MDS server object, the JNDI libraries returns an array of NamingEnumeration objects that have to be mapped into a customized 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() MDS methods.

3.3 CORBA CoG Grid Security Service

3.3.1 Security Requirements

Large scale distributed computing environments consist of computers, storage systems, advanced instruments and distributed applications. The Grid applications are distinct from traditional client-server applications by their use of large number of resources, dynamic resource requirements from multiple administrative domains, complex communication structures and stringent performance requirements. While heterogeneity is a desired goal for computational Grids, it leads to security problems which are not addressed by the current security technologies. For example a Grid application that requires multiple resources needs to establish security relationship with processes that span the administrative domains of those resources. As the Grid is dynamic it may not be possible to establish prior trust relationships between sites. As such the security policy should be able to interoperate with intradomain access technologies. It should provide authentication solutions that allow users, the processes that comprise a user's computation, resources used by processes to verify each other's identity. Some of the basic security functions should include authentication, access control, integrity, privacy and nonrepudiation.

In short the primary motivations behind the Grid Security Service are:

- The need for secure communication (authenticated and perhaps confidential) between elements of a computational Grid.
- The need to support security across organizational boundaries, thus prohibiting a centrally-managed security system.
- The need to support "single sign-on" for users of the Grid, including delegation of credentials for computations that involve multiple resources and/or sites.

3.3.2 Grid Security Infrastructure

The Grid Security Infrastructure(GSI) is the implementation of the security architecture in Globus toolkit. It provides single sign on by creating a *user proxy*. A user proxy is defined as a session manager process given permission to act on behalf of a user for a limited period of time. The interdomain security issues are handled by a *resource proxy*. The resource proxy is an agent which translates between interdomain security operations and local intradomain mechanism. GSI is implemented on top of Generic Security Services application programming interface (GSS-API). It provides security services to callers in a generic fashion and as such the services can be implemented by a range of underlying mechanism and technologies. The GSI-API bindings have been defined for two mechanism - one based for plaintext passwords and one based on X.509 certificates. The latter mechanism is used for wide area production use. The GSI implementation currently uses the authentication protocols defined by SSL (Secure Socket Library) [38].

3.3.3 CORBA Security Service

One of the services provided by CORBA is the security service. It provides abstraction to the application layer from the different underlying technologies. It tries to abstract the application from the underlying transport and security mechanisms. It provides the following security functionalities

- Authentication: Clients and targets can verify the identity of each other.

- Message Protection: Transit data can be protected from integrity and confidentiality attacks.
- Authorization: Access to objects and methods can be controlled.
- Audit: Logs can record which operations are invoked by which client.
- Non-Repudiation: This functionality is optional. Irrefutable evidence of method invocations can be generated and verified.

CORBASec [30] does not implement all the above functionalities itself but relies on the underlying security mechanism such as Kerberos v5 [29], SESAME [37] and SSL [38]. As such the functionalities offered are limited by the underlying technologies. The fundamental goals of CORBASec is to provide confidentiality, integrity, accountability, and availability. In addition the further requirements such as simplicity, consistency, scalability, transparency, easy administration, easy implementation of applications, certification, assurance, mechanism independence, reusability of the existing security infrastructure, flexibility, and interoperability are also specified in the specifications. The CORBA security specifications was first published in 1995 and has gone through a no of revisions. The current version is 1.7. Explaining the CORBA security architecture is not trivial. As such this thesis will only concentrate in explaining the authentication protocol which is used by the CORBA CoG toolkit. The CORBA security model for mutual authentication in a client server architecture is shown in Figure 3.4. It requires the client application to initially establish its right to access objects on a secure orb. The client is identified as a *User Sponsor* object in the security model. It passes its secure credentials, such as it public certificate and password, to the *CORBA Security Principal Authenticator* object, which then creates a *Credentials* object. The User Sponsor object then passes the Credential object reference to the *Current* object. This Current object represents the current execution context. To the client, a secure client invocation looks like any other method invocation, however the security service obtains the required client security information from the Current object. Similarly the Security Service at the server end can obtain required security information from the server's local Current object. The actual authentication between the client and the server however depends on the underlying authentication mechanism, e.g. Kerberos

V5, SESAME or SSL. All the communication between the objects during authentication takes place transparently. The only concern the application developer should address is the location of the public and Certificate Authority (CA) certificates and the private key. These certificates and the private key are obtained when Globus is installed on one of the machines. If the client is executed from the machine that does not have Globus installed then the certificates and the key should be securely copied to the machine.

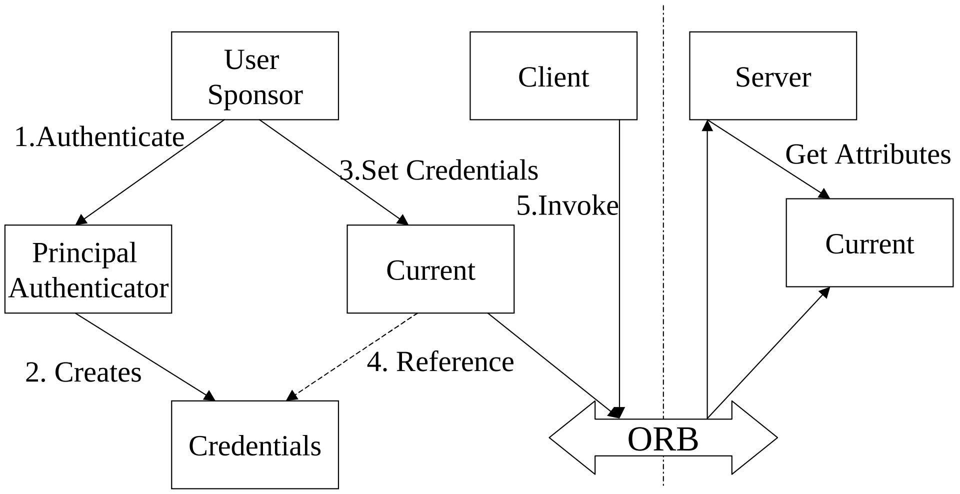


Figure 3.4: `CORBA Security Architecture`

3.3.4 CORBA CoG Security Service

Providing access to Grid security is an essential part of the CORBA CoG Kit. The implementation is based on the Globus Grid Security Infrastructure (GSI) [10]. It is designed in such a way that it combines the functionalities provided by CORBA security service with GSI implementation. To understand GSI implementation some of the concepts are explained below.

Certificates

A central concept in GSI authentication is the certificate. Every user and service on the Grid is identified via a certificate, which contains information vital to identifying and authenticating the user or service. A GSI certificate includes four primary pieces of information.

- A subject name, which identifies the person or object that the certificate represents.
- The public key belonging to the subject.

- The identity of a Certificate Authority (CA) that has signed the certificate to certify that the public key and the identity both belong to the subject.

- The digital signature of the named CA.

A third party (a CA) is used to certify the link between the public key and the subject in the certificate. In order to trust the certificate and its contents, the CA's certificate must be trusted. The link between the CA and its certificate must be established via some non-cryptographic means, or else the system is not trustworthy. GSI certificates are encoded in the X.509 certificate format, a standard data format for certificates established by the Internet Engineering Task Force [20]. These certificates can be shared with other public key-based software, including commercial web browsers from Microsoft and Netscape.

Mutual Authentication

If two parties have certificates, and if both parties trust the CAs that signed each other's certificates, then the two parties can prove to each other that they are who they say they are. This is known as mutual authentication. The GSI uses the Secure Sockets Layer (SSL) for its mutual authentication protocol, which is described below. (SSL is also known by a new, IETF standard name: Transport Layer Security, or TLS.)

Before mutual authentication can occur, the parties involved must first trust the CAs that signed each other's certificates. In practice, this means that they must have copies of the CAs' certificates–which contain the CAs' public key and that they must trust that these certificates really belong to the CAs. To mutually authenticate, the first person (A) establishes a connection to the second person (B). To start the authentication process, A gives B his certificate. The certificate tells B who A is claiming to be (the identity), what A's public key is, and what CA is being used to certify the certificate. B will first make sure that the certificate is valid by checking the CA's digital signature to make sure that the CA actually signed the certificate and that the certificate hasn't been tampered with. (This is where B must trust the CA that signed A's certificate.) Once B has checked out A's certificate, B must make sure that A really is the person identified in the certificate. B generates a random message and sends it to A, asking A to encrypt it. A encrypts the message using his private key, and sends it back to B. B decrypts the message using A's public key. If this results in the original random message, then B knows that A is who he says he is. Now that B trusts A's identity, the same operation must happen in reverse. B sends A her certificate, A validates the certificate and sends a challenge message to be encrypted. B encrypts the message and sends it back to A, and A decrypts it and compares it with the original. If it matches, then A knows that B is who she says she is. At this point, A and B have established a connection to each other and are certain that they know each others' identities.

Confidential Communication

By default, the GSI does not establish confidential (encrypted) communication between parties. Once mutual authentication is performed, the GSI gets out of the way so that communication can occur without the overhead of constant encryption and decryption. The GSI can easily be used to establish a shared key for encryption if confidential communication is desired. Recently relaxed United States export laws now include encrypted communication as a standard optional feature of the GSI. A related security feature is communication integrity. Integrity means that an eavesdropper may be able to read communication between two parties but is not able to modify the communication in any way. The GSI provides communication integrity by default. (It can be turned off if desired). Communication integrity introduces some overhead in communication, but not as large an overhead as encryption.

Securing Private Keys

The core GSI software provided by the Globus toolkit expects the user's private key to be stored in a file in the local computer's storage. To prevent other users of the computer from stealing the private key, the file that contains the key is encrypted via a password (also known as a pass phrase). To use the GSI, the user must enter the pass phrase required to decrypt the file containing their private key. In our implementation the passphrase is available from the properties file of the toolkit.

Delegation and Single Sign-On

The GSI provides a delegation capability: an extension of the standard SSL protocol which reduces the number of times the user must enter his pass phrase. If a Grid computation requires that several Grid resources be used (each requiring mutual authentication), or if there is a need to have agents (local or remote) requesting services on behalf of a user,

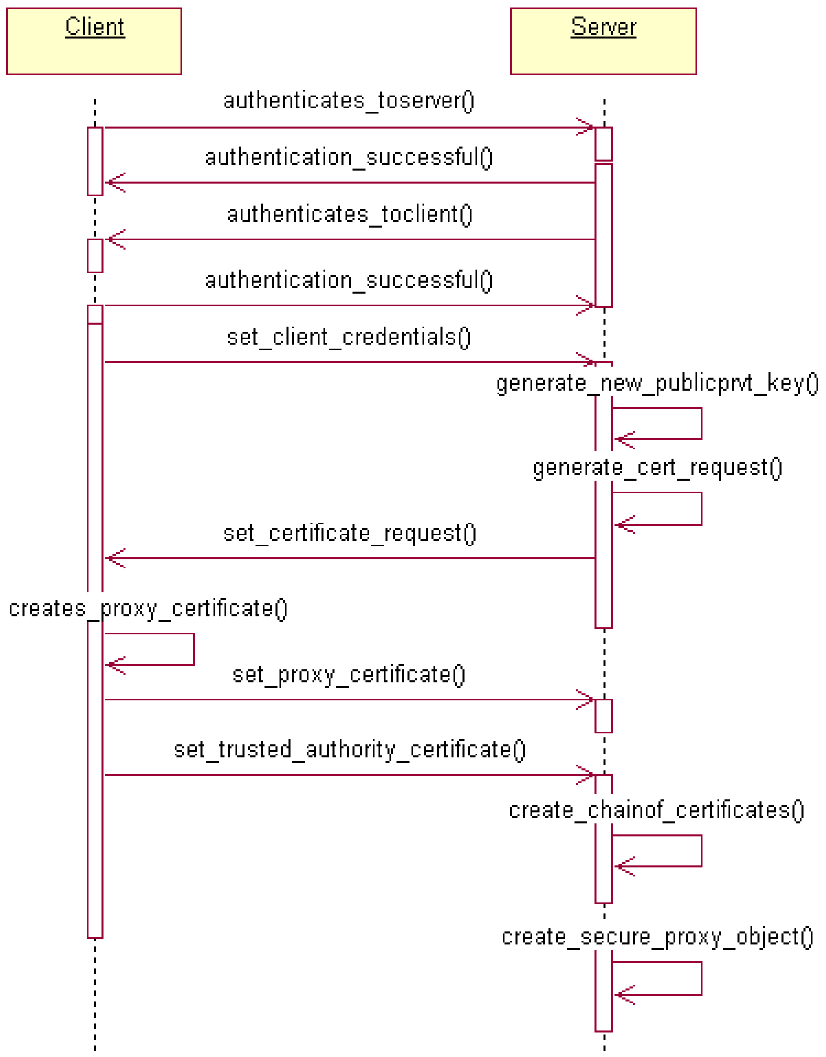

Client
Server
authenticates_toserver()
authentication_successful()
authenticates_toclient()
authentication_successful()
set_client_credentials()
generate_new_publicprvt_key()
generate_cert_request()
set_certificate_request()
creates_proxy_certificate()
set_proxy_certificate()
set_trusted_authority_certificate()
create_chainof_certificates()
create_secure_proxy_object()

Figure 3.5: `Delegation Model`

the need to re-enter the user's pass phrase can be avoided by creating a proxy. A proxy consists of a new certificate (with a new public key in it) and a new private key. The new certificate contains the owner's identity, modified slightly to indicate that it is a proxy. The new certificate is signed by the owner, rather than a CA see Figure 3.5. The certificate also includes a time notation after which the proxy should no longer be accepted by others. We yet have to implement the time duration of the proxy. Currently once the proxy is created it is only destroyed when the server application exits.

The proxy's private key must be kept secure, but because the proxy isn't valid for very long, it doesn't have to kept quite as secure as the owner's private key. It is thus possible to store the proxy's private key in a local storage system without being encrypted, as long as the permissions on the file prevent anyone else from looking at them easily. Once a proxy is created and stored, the user can use the proxy certificate and private key for mutual authentication without entering a password. When proxies are used, the mutual authentication process differs slightly. The remote party receives not only the proxy's certificate (signed by the owner), but also the owner's certificate. During mutual authentication, the owner's public key (obtained from her certificate) is used to validate the signature on the proxy certificate. The CA's public key is then used to validate the signature on the owner's certificate. This establishes a chain of trust from the CA to the proxy through the owner. The GSI and software based on it (notably the Globus toolkit, GSI-SSH, and GridFTP) is currently the only software which supports the delegation extensions to TLS (a.k.a. SSL). The Globus Project is actively working with the Grid Forum [12] and the IETF to establish proxies as a standard extension to TLS so that GSI proxies may be used with other TLS software.

3.3.5 CORBA CoG GSI Server Object

One can integrate Grid security at various levels of the CORBA architecture. In order to maintain portability across orbs, the protocol stack has not been modified but an intermediary CORBA object is placed between the CORBA client and Grid services called the CORBA GSI server object. This GSI server object creates a secure proxy object, which allows other CORBA objects, i.e. MDS server, GRAM server and GASS server 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) [30, 3] . 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 [7] and gasserver [4], only accept connections from clients with secure Globus credentials, the CORBA client delegates the GSI server 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 GRAM server and GASS server 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 GSI server object involves the following steps.

1. The client sends over its public certificate in an encoded form to the server object.

2. The GSI 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.

3. 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.

4. 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.

5. 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.

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

Figure 3.6: `Interface to CORBA CoG GSI Service`

The IDL interface for the GSI server object is shown in Figure 3.6. 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.

The above delegation process is executed only once in the entire client and server interaction. The client initiates the process by obtaining an handle to the GSI server object from the Naming Service and calls upon the delegation process. The secure proxy object which is created by the delegation process is stored by the GSI server object in a hashtable keyed on the subjectid. When the other service objects such as GASS server/GRAM server objects need the secure proxy object the GSI server object queries the hashtable and return the secure proxy object to the respective server object. By using the secure proxy object the GASS server/GRAM server objects communicate securely with it's respective Globus services.

3.4 CORBA CoG Resource Allocation Service

3.4.1 Globus Resource Allocation Manager

Metacomputing systems allow applications to be executed on various computational resources without regards to physical location. These systems need to address issues such as location and allocation of computational resources, authentication, process creation and other activities for resource usage. The challenging problems which any resource management architecture needs to address are:

- Site Autonomy: The resources will be geographically distributed, under different administrative domains and different use policy, security mechanisms and scheduling policies.
- Substrate: As the resources are scattered, the different sites may use different resource management systems such as Condor, EASY and CODINE.
- Policy Extensibility: The solution should be able to support the changes in the administrative domains.
- Co-allocation: As an application can have multiple resource requirements it is essential to have specialised mechanism for allocating multiple resources, initiating computation and monitoring the resources.

The resource management architecture in Globus toolkit addresses the issues of site autonomy and heterogenous substrate by introducing entities called resource managers which interface with local resource management tools. For online control and policy extensibility, a *resource specification language* is defined that supports negotiation between different components of a resource management architecture. The resource brokers handle the mapping of high level application requests into request of individual managers. The resource co-allocators address the problem of co-allocation. For our implementation we are particularly interested in the local resource managers. This entity is called Globus Resource Allocation Manager (GRAM) and is responsible for

1. Processing RSL requests by either denying the request or creating one or more processes to satisfy the request.

2. Monitoring and management of jobs that are created in response to the request.

3. Updating the MDS about the availability and the capabilities of the resource.

The GRAM implementation is kept general by mapping the resource specification into a request for local resource allocation mechanisms. When a job is submitted to GRAM a global unique *job handle* is returned that can be used for monitoring the status and completion of the job. The implementation consists of the following components [7] - a GRAM client library, gatekeeper, a Resource Specification Language (RSL), a parsing library, a job manager, and a GRAM reporter. GRAM uses GSI for authentication and authorization.

3.4.2 Job Submission in Globus

The process of remote job submission using GRAM involves the following steps. First the application authenticates with the gatekeeper at the remote site using the GSI libraries. Next the application specifies the resources it needs (using RSL), the location of the binary to be executed, and a callback function that will be invoked when the state of the job changes. The Globus gatekeeper at the remote end mutually authenticates the user and the resource, determines the user name for the remote user at the resource, and starts a job manager. The job manager is responsible for the creation of the actual process that is requested by the user. It submits the resource allocation request obtained from the RSL parsing library to the local resource management system. This can either be a batch queue or a simple process forked as part of the operating system management. In case it is not possible to start the job an error is returned. Once the process is created the job manager monitors the state of the process and uses the callback function to notify the user of a state transition or process termination. The responsibility of job manager ends once the process has terminated.

3.4.3 CORBA CoG GRAM Service

The CORBA CoG Kit GRAM server object implements an interface to GRAM and enables application developers to access its capabilities to submit jobs on a remote computer, bind to an existing job, get status updates (active, done, pending, failed and suspended) on submitted jobs, and cancel jobs. The state can also be monitored via the MDS. To understand fully the implementation of the CORBA CoG GRAM Service it is essential to have some understanding of the CORBA event service.

3.4.4 CORBA Event Service

In CORBA the event service represents a communication model where an application can send an event that will be received by any number of objects. The model provides two ways of initiating event communication and the communication can itself take up two forms. There are two main entities in the CORBA event service - the *suppliers* that initiate the event and the *consumers* that consume the event. Suppliers and Consumers are completely decoupled: a supplier has no knowledge of the number of consumers or their identities and consumers have no knowledge of which supplier generated the given event. In order to support this model the event service has a new architectural element called the *event channel*. An event channel mediates the transfer of events between the suppliers and consumers as follows:

1. The event channel allows consumers to register interest in events, and stores this registration information.
2. The channel accepts incoming events from suppliers.
3. The channel forwards supplier generated events to registered consumers.

Suppliers and consumers connect to the event channel and not directly to each other. From a supplier's perspective, the event channel appears as a single consumer and from a consumer's perspective, the event channel appears as a single supplier. In this way, the event channel decouples suppliers and consumers. Any number of suppliers can issue events to any number of consumers using a single event channel. There is no correlation between the

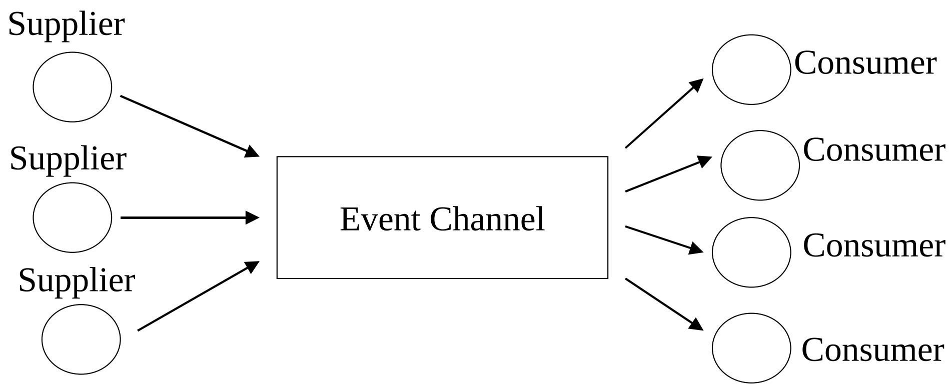


Figure 3.7: `CORBA Event Service`

number of suppliers and the number of consumers, and new suppliers and consumers can be easily added to the system. In addition, any supplier or consumer can connect to more than one event channel. CORBA defines the event service at a level above the orb architecture. Suppliers, consumers and event channels may be implemented as orb applications, while events are defined using standard IDL operation calls. Suppliers, consumers and event channels each implement clearly defined IDL interfaces that support the steps required to transfer events in a distributed system. CORBA specifies two approaches to initiating the transfer of events between suppliers and consumers. These approaches are called the Push model and the Pull model. In the Push model, suppliers initiate the transfer of events by sending those events to consumers. In the Pull model, consumers initiate the transfer of events by requesting those events from suppliers. The event communication can be 2 form, typed or untyped. In untyped event communication, an event is propagated by a series of generic push() or pull() operation calls. The push() operation takes a single parameter which stores the event data. The event data parameter is of type any, which allows any IDL defined data type to be passed between suppliers and consumers. The pull() operation has no parameters but transmits event data in its return value, which is also of type any. Clearly, in both cases, the supplier and consumer applications must agree about the contents of the any parameter and return value if this data is to be useful. In typed event communication, a programmer defines application-specific IDL interfaces through which events are propagated. Rather than using push() and pull() operations and transmitting data using an any, a programmer defines an interface that suppliers and consumers use for the purpose of event communication. The operations defined on the interface may contain parameters defined in any suitable IDL data type. In the Push model, event communication is initiated simply by invoking operations defined on this interface. The Pull model is more complex because event communication is initiated by invoking operations on an interface that is specially constructed from the application-specific interface that the programmer defines. As a consequence, the Push model and the Pull model can be used to transmit typed or untyped events.

In our design we have used the push model where the server pushes the data into the event channel and the client polls on the event channel and picks up the data as and when it arrives on the event channel. The data type passed within the communication is defined in the idl and consists of the jobid and the jobstatus. The details of the data type are explained in detail in the next section.

3.4.5 CORBA CoG GRAM Server Object

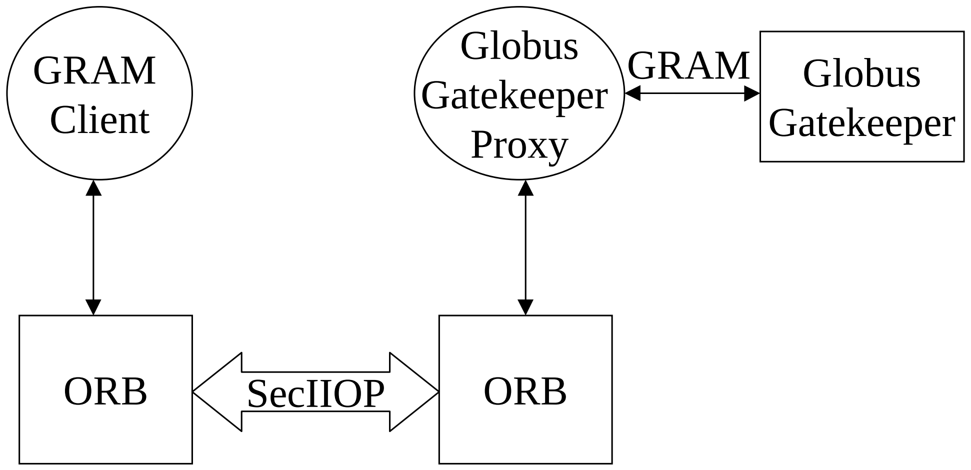


Figure 3.8: `CORBA CoG GRAM Service`

Job submission using CORBA CoG GRAM Service consists of the following steps: First, the client authenticates with the CORBA server object using CORBASec. For this the client and the server application should have access to their respective credentials such as private key, public certificate and trusted CA certificate. In our implementation we have used Globus credentials and Globus CA as the certificate authority. If the client and the server application are installed where Globus is not installed then the credentials especially the

private keys should be securely copied. After mutual authentication is successful, the client subscribes to the CORBA event channel on which the server is listening. The architecture design is such that the server does not set up an event channel till the client does not initiate it. This helps in restricting an event channel solely between the client and the server. To implement this design a new service is created called the EventService. The implementation object, called EventServerImpl, of EventService has a handle to the server object. The client obtains the reference of the event service from the naming service and then invokes a method on EventServerImpl. The EventServerImpl inturn directs the server to start an event channel. The client and server then bind to the same event channel thus ensuring a secure channel of communication between them. This design is robust and scalable as new event channel can be initiated for a new client application trying to connect to the server. Next the client gets a handle to the GSI server object from the naming service and delegates the CORBA GSI server object as described in the process in the earlier section. Once delegation is successful, the client obtains a reference to GRAM server 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 in *Resource Specification Language*. The syntax of an RSL language is based on the syntax for filter specifications in the Lightweight Directory Access Protocol(LDAP) and MDS. The RSL specification is constructed by combining simple parameter specifications and conditions by &, —, and +. For example the following specification
&(executable=/bin/echo)(—(&(count=6)(memory¿=64))(&(count=10)(memory=32)))
(resourcemanager=penn.rutgers.edu:2119)

requests 6 nodes with atleast 64MB of memory or 10 nodes of 32MB of memory to execute the program /bin/echo on the resource penn.rutgers.edu with portno as 2119.

On receiving the request, the GRAM server uses the secure proxy object created by GSI server during delegation to set up a secure connection with the GRAM gatekeeper. It then forwards the request to the gatekeeper. The gatekeeper returns the jobid and the status ACTIVE on successful initiation of the request. This jobid and jobstatus is wrapped in an object defined in the idl and communicated to the client as an any object through the event channel. As and when the job status changes the gatekeeper notifies the server and the server communicates it to the client. For e.g on completion of the job the server communicates to the client the jobid and the status as DONE. As multiple client applications can be connected to a GRAM server at one point of time the server has a hashtable keeping an account of jobids of each client and their corresponding communication channel. It is also possible for the client to suspend a job once its execution has begun with the jobid. On successful suspension the client will get back the status of SUSPENDED from the server.

```
module GRAMService
{
  exception GramException {short errorcode;};
  exception GlobusProxyException{short errorcode;};

  struct JobStatus        {
    string jobid;
    string currstatus;
  };
  interface GRAMServer
  {
    typedef sequence (octet) ByteSeq;
    void setProxyCredentials(in ByteSeq certificate);
    void jobRequest(in string rsl,in string contact, in boolean batchjob);
  };
};
```

Figure 3.9: `Interface to CORBA CoG GRAM Service`

The implementation of the GRAM server object in the CORBA CoG Kit provides a simple interface with the following methods (see Figure 3.9)

1. setProxyCredentials() : This method sets the reference to the proxy secure object created by the GSI server object.

2. jobRequest(): This method is used by the client to request a job submission on a remote resource.

3. jobBind(String jobid) : This method is used by the client to bind to an existing job identified by the jobid.

4. jobUnbind(String jobid) : This method is used by the client to unbind to an existing

job identified by the jobid.

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.

3.5 CORBA CoG GASS Service

3.5.1 Global Access to Secondary Storage

In high performance distributed applications there arises a need to access data which is not colocated with the site at which the computation is performed. This problem is challenging because the solution should not require a lot of changes to the application program and to the resource provider so that new resources can be easily incorporated into the grid environment. The current technologies such as distributed file systems provide convenient access to remote data but require substantial technology deployment and interorganizational cooperation. Web based file systems provide transparent access to remote resources but require special kernel capabilities in the target systems. Condor [32] avoids the need for kernel services as it has its own specialized versions of I/O libraries but only provides access to data on a user's "home" machine. Legion [15] provides access to Legion objects but not to data stored in the conventional file systems. The data movement and access service provided by the Globus toolkit provides a new mechanism to access remote data.

- It provides mechanism for common grid I/O patterns, such as executable staging, reading of configuration files, error/diagnostic output, and simulation output.
- It provides mechanism which can be implemented at the participating site without specialized services.
- It provides mechanism which allow programmers to guide or override default data movement strategies by controlling data source selection, staging, caching and filtering of data before transfer.

3.5.2 CORBA CoG GASS Service

The objective of the CORBA GASS server object is to provide an interface to the Globus GASS service as shown Figure 3.10.

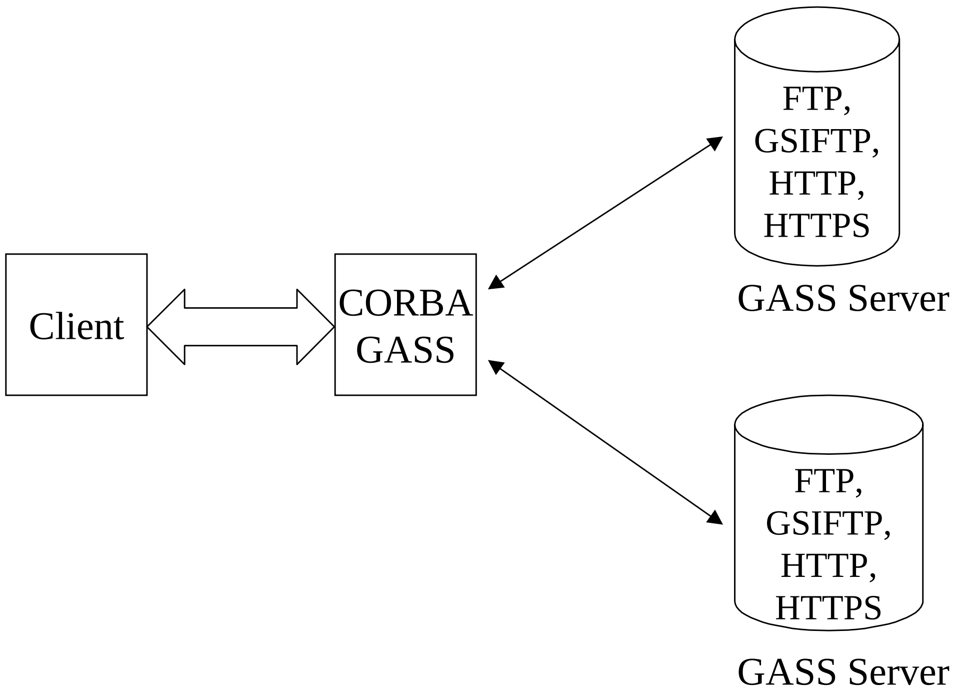


Figure 3.10: `CORBA CoG GASS Service`

The client gets a handle to the GASS server 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 GASS server object is defined in the IDL as shown in Figure 3.11.

The client initiates the process of the file copy by getting a handle to the GASS server object from the naming service. It then specifies the current location of the file by invoking the setSourceURL function on the server object. Similiarly it specifies the destination location of the file by calling setDestinationURL. When it initiates the copy function, the

```
module GASSService
{
  interface GASSServer
  {
    void setSourceURL(in string url);
    void setDestinationURL(in string url);
    void allowThirdPartyTransfer(in boolean value);
    void setProxyCredentials();
    void URLCopy();
  };
};
```

Figure 3.11: `Interface to CORBA CoG GASS Service`

GASS server object uses the secure proxy object created by the GSIService to authenticate with the back end GASS Server. Once the mutual authentication is successful a secure socket connection is created between the the CORBA GASS server object and the backend GASS Server. The file is copied to the destination file using this connection. The implementation of CORBA GASS server object is based on JAVA CoG Kit. It provides a wrapper class around the JAVA CoG Kit classes which provide the protocol implementation for communication between the servers.

3.6 Conclusion

The above interfaces in CORBA to the Grid services is one of the attempts to create a bridge between the commodity technology CORBA and Grid. In the next chapter we show how using these services it is possible to submit jobs on remote resources and create a Grid enabled application.

Chapter 4

Experiments and Evaluation

4.1 Experiments

The following experiment illustrates the capability of the CORBA CoG toolkit in the Grid computing environment. The experiment uses the Tportamr application. This application adopts the adaptive mesh refinement technique to solve a simple 2-dimensional transport equation of the form

u,t + u,x + u,y = 0

where u represents some physical quantity and u,t represents the partial derivative of u with respect to t, and so on. The computational domain is a rectangle. The initial values of u are chosen from a gaussian distribution according to the initial parameters chosen by the user. Generally a PDE is solved by choosing a discrete domain where the algebraic analogues of the PDEs are solved. One standard method is to introduce a grid and estimate the values of the unknowns at the grid points through the solutions of these algebraic equations. The spacing of the grid points determines the local error and hence the accuracy of the solution. The spacing also determines the number of calculations to be made to cover the domain of the problem and thus the cost of the computation. For well behaved problems a grid of uniform mesh spacing (in each of the coordinate directions) gives satisfactory results. However, there are classes of problems where the solution is more difficult to estimate in some regions (perhaps due to discontinuities, steep gradients, shocks, etc.) than in others. In such cases the adaptive mesh refinement technique helps. It starts with a coarse grid and as the solution proceeds the regions are identified which require more resolution by some parameter characterizing the solution, for e.g. the local truncation error. Finer subgrids are imposed only on these regions. Finer and finer subgrids are added recursively until either a given maximum level of refinement is reached or the local truncation error has dropped below the desired level. Thus in an adaptive mesh refinement computation grid spacing is fixed for the base grid only and is determined locally for the subgrids according to the requirements of the problem. This parallel application can be configured to run on multiple processors by specifying the no of processors by the user.

4.1.1 Scenario

The test environment consisted of four machines each running a respective part of the software. The client applications was deployed on Tassl-pc2 and Tassl-pc1 running on WinNT, server application on discover.rutgers.edu running Linux and Globus toolkit on grid1.rutgers.edu running on linux. The JacORB [23] orb was installed on both the client and the server machines. Figure 4.1, Figure 4.2, and Figure 4.3 illustrates the different setups of the experiment for testing CORBA CoG toolkit. In the second setup we had two clients submitting different jobs to the same server and the last setup had one client submitting two consequetive jobs on the same server and monitoring the progress of the job with the help of jobid.

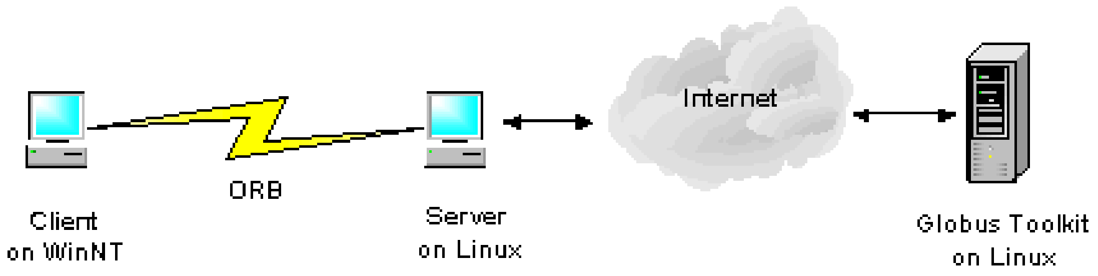


Figure 4.1: The CORBA CoG Kit Testing Scenario 1

4.1.2 Setup

A couple of setup steps were required before CORBA CoG kit could be used for running the application on a remote resource.

- The installation of an ORB at the client and the server end. In our implementation we have used the freely available Java [25] orb called JacORB available at www.inf.fu-berlin.de. JacORB is an open source software. To install it, it was just required to

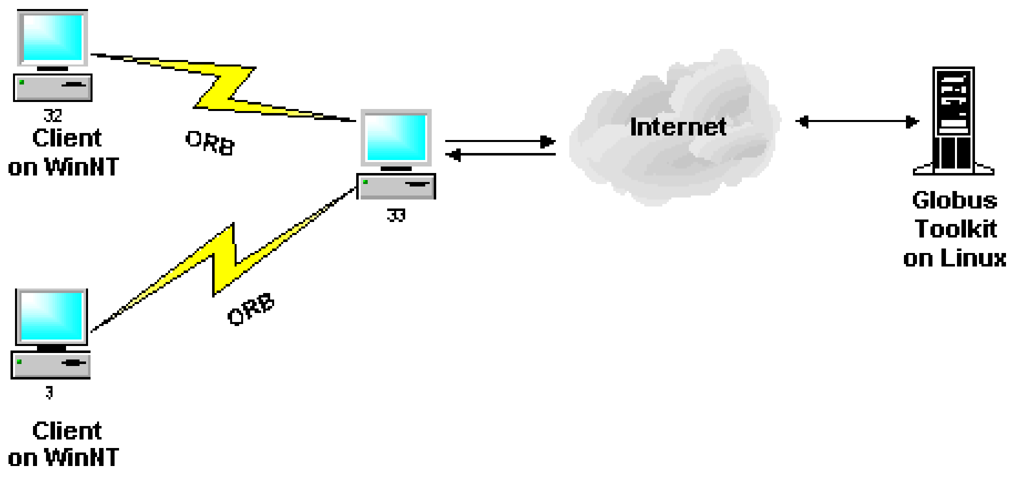


Figure 4.2: The CORBA CoG Kit Testing Scenario 2

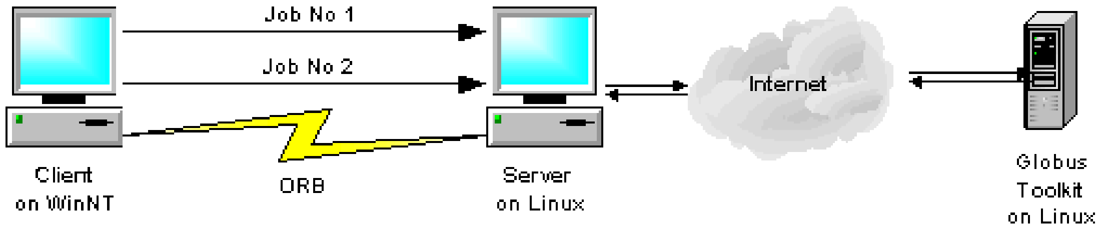


Figure 4.3: The CORBA CoG Kit Testing Scenario 3

unzip and untar the archive which resulted in a new directory called JacORB1.3.

- The installation of cryptography and SSL libraries by IAIK [19] viz. IAIK-JCe2.5 and iSaSiLk3.0.
- Made changes to the classpath to include 'jacorb.jar' and added 'bin' to the search path.
- Made changes to the various properties file which are provided by JacORB. The property file 'jacorb.properties' comes with a number of configuration options and is either stored in the home directory or in the current directory. If JacORB finds multiple files it loads all of them and if there are different settings for the same property then the last loaded one takes the most precedence. The only property which needed to be changed was the ORBInitRef.NameService. The string value of this property could be a file or a url. This value would be used by the ORB to locate the file used to store the name server's object reference.

- Created a keystore which will store the Globus certificates and private keys as JacORB uses keystore during authentication process. A keystore is simply a file that contains public key certificates and the corresponding private key. It also contains other certificates that can be used to verify the public key certificates. All cryptographic data is protected using passwords and accessed using names called aliases. JacORB provides a GUI tool which allows the easy creation of keystore files. 2 keystore files were created, one for client and another for the server. Each of these keystore contained a Globus public certificate, a Globus private key and the Globus Certificate Authority.

- The respective keystore file names were added to the client and server property file. The property to be defined were
  jacorb.security.keystore=keystorefilename
  To avoid typing in lot of aliases and passwords (one for the keystore and one for each entry that is used) the aliases and the password can be specified in the property file as
  jacorb.security.default_user=brose
  jacorb.security.default_password=jacorb
  #the name and location of the keystore relative to the home directory
  jacorb.security.keystore=.keystore.

4.1.3 Execution Process

The different steps for the successful execution of the Tportamr application on a remote resource are outline below:

- The CORBA naming service was started on the server machine called ‘discover’ running linux using the command
  ns ns.ior
  where ns.ior is the filename which the client application references to obtain the location of the naming server.

- Next the CORBA CoG server application was started on the same server machine from the following command

jaco -Dcustom.props = server_props org.globus.CORBACoG.Server

The orb running the server application reads in the properties file and from the value of the property ORBInitRef.NameService it gets the reference to the CORBA naming service. For a secure connection it first authenticates to the naming service using the keystore filename mentioned in the server properties file and jacorb properties file. It then binds the various CORBA CoG server objects to the naming server and waits for request from the client.

- The client application was started on Tassl-pc-2 running WinNT using the command

  jaco -Dcustom.props=client_props org.globus.CORBACoG.Client

  The client in the similiar manner to the server first binds itself securely to the naming service and then obtains local references to the various CORBA CoG server objects. On obtaining reference to the CORBA CoG GSIServer object the client initiated the process of delegation where the client and the server mutually authenticate each other using CORBASec and their respective keystore. The successful completion of the delegation process resulted in the creation of a secure proxy object which the server objects will use to forward request to the respective Globus services. Only when the delegation was completed the client initiated the creation of a secure event channel with the server on the CORBA CoG EventServer object. The EventServer object created a push event channel between the client and the server. Finally the client was in the state for requesting a job on a remote resource.

- To obtain the reference to the remote resource which satisfy the resource requirements the CORBA CoG MDSServer object was queried. For e.g if the tportamr application needs to be executed on a resource having four nodes with more than 64MB memory then the query is passed to the MDSServer object which in turn queries the Globus MDS to obtain the list of resources. In this case the query was kept simple and just the resource name was sent as the parameter in the query. The CORBA CoG MDSServer object returned with the resourcename "penn.rutgers.edu" and its various resource properties.

- The tportamr application was then copied to "penn.rutgers.edu" using the CORBA CoG GASSServer object. The parameter for setSourceURL was the location of the tportamr application and the parameter for setDestinationURL was the location where the application will be copied in the remote resouce. The protocol specified in the source url was used for copying the file.

- The client requested the execution of the job on CORBA CoG GRAMServer object by specifying the application, the resource name, the portno, the arguments for the application in the rsl language. On receiving the request the GRAMServer object used the secure proxy object created by GSI for authenticating with the resource gatekeeper. On successful authentication the gatekeeper started the process on the resource and returned a jobid. This jobid was returned by the server object to the client using the event channel. The client could use the jobid for obtaining periodic status update of the job.

- On the completion of the job, the gatekeeper notified the server with job status as DONE. This status was propogated by the server to the client using the event channel.

- The output of tportmar application was logged into a file called "out" which was copied back to the client side by using the CORBA CoG GASSService. This time the sourceurl and destinationurl have to be interchanged.

The interaction diagram of the above scenario is shown in Figure 4.4. The above experiment displays how an application can be executed remotely using the four basic services of CORBA CoG toolkit.

4.2 Timing Results

The above experiments were repeated about 8 times to get an average value of the time to execute each process for each scenario. The Figure 4.5 shows the execution time in milliseconds for the processes Resolve Services, Delegation, Event Channel Creation and Job Execution. These timing result are for the scenario where the client, server and the Globus Toolkit are located on the same machine. The results project the exact time to

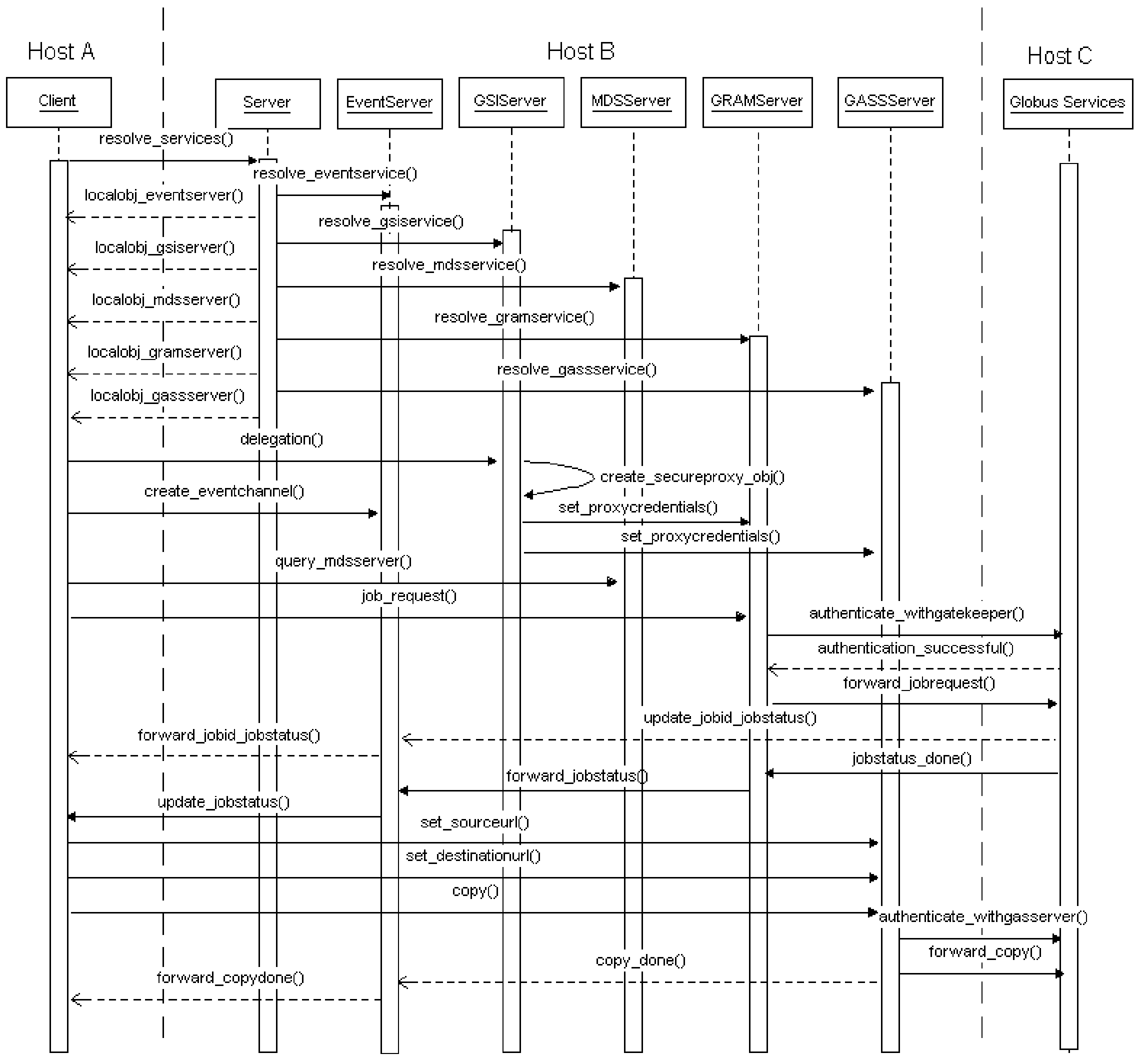


Figure 4.4: The CORBA CoG Kit Interaction Diagram

execute the processes by eliminating the network.

The next Figure 4.6 shows the change in the execution time of the different processes with change in the setup. In the case of Resolving Services as a search is performed on Globus Toolkit the execution time increases when the server is not located on the same machine as the Globus Toolkit. The time increase is attributed to the network. For Delegation process the interaction is mainly between the client and the server and as such the time increase is very small. Similiar explanation can be attributed to Event Channel Creation process. For Job Execution process when 2 clients are connected to one server the extra time is logged as the server is processing one job and then it executes the second job.

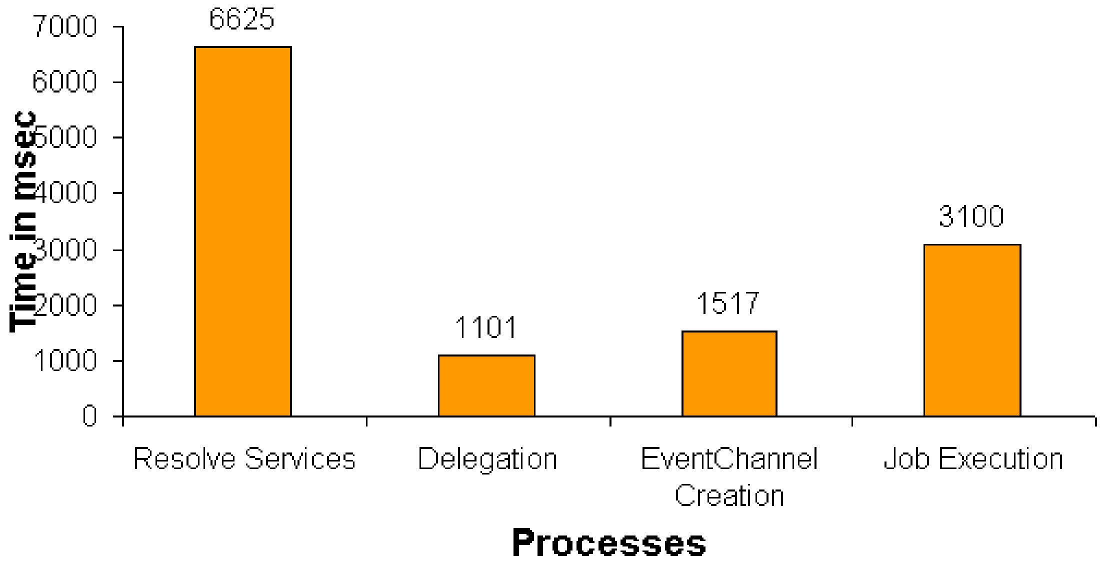


Figure 4.5: Execution Time

The Figurereffig:memusage shows the increase in memory utilization when the no of the clients connecting to the server increases. We could only experiment with at most 2 clients as we had Globus certificates for only 2 users.

The most interesting experiment was the comparison of Java and CORBA CoG Kit. The comparison could only be done for 3 processes as the event channel creation is very unique to CORBA CoG and it is handled as a Listener by the Java CoG Kit. Initially the Delegation process for the Java CoG Kit took about 6000 msec which showed that the CORBA CoG Kit was faster than Java CoG Kit. Logically it did not justify as the whole process of delegation in Java CoG Kit takes place on one machine unlike the CORBA CoG Kit where the client and server interact to send its certificate and certificate request. With input from Jarek we realised that the seed generator which is used for creating the new public and private key was taking the majority of the time. On initialising the seed generator the execution time came down to 562 msec which is surely better than 922 msec. When I applied this initialization for the CORBA CoG Kit no effect was observed as I believe the CORBA Securlty layer already does the intialization of the seed generator during the setup of the security layer. The execution time of other processes showed that the CORBA CoG Kit took a longer time due to the overhead of the client-server architecture and the presence of the ORB layer.

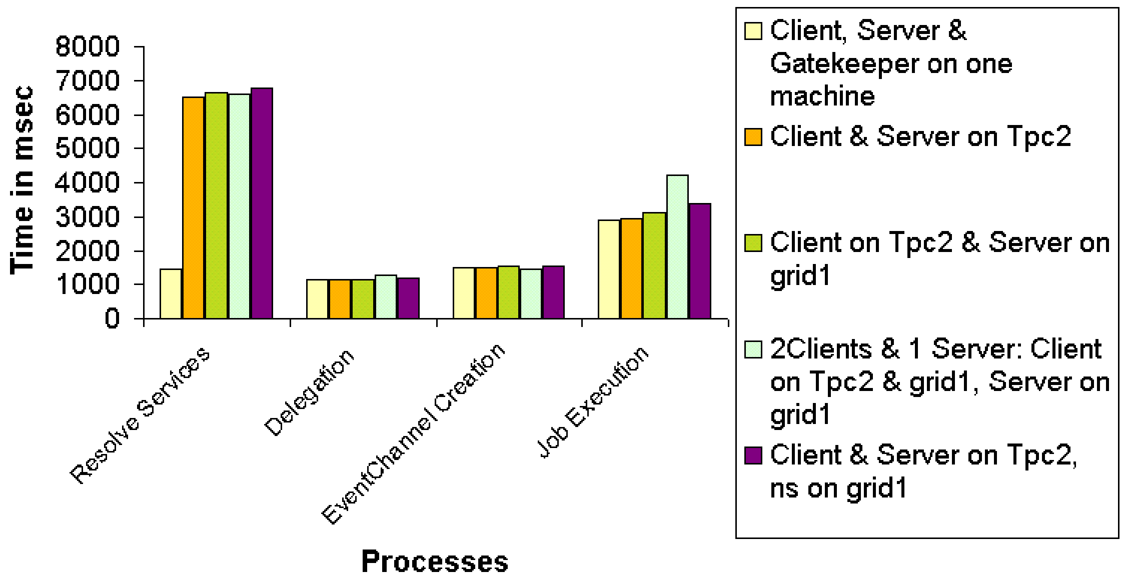


Figure 4.6: Execution Time for Different Scenarios

4.3 Applications

Many other applications can benefit from a CORBA CoG Kit. One example is the Numerical Propulsion System Simulation (NPSS) [35], which is a part of the NASA IPG and provides an engine simulation using computational fluid dynamics. It consists of 0- to 3-dimensional engine component models responsible for examining aerodynamics,structures, and heat transfer. Previous studies show that the NPSS's engine components can be encapsulated using CORBA in order to provide object access and communication from heterogeneous platforms while at the same time coordinating the modelling runs across Globus. As part of this task, a large number of NPSS jobs (1000+) are submitted from a desktop interface returning the output to that same interface using the CORBA CoG Kit. This application can be also integrated with other services such as DISCOVER [8]. DISCOVER service allow users to collaboratively monitor and control application, access, interact, and steer individual component objects; manage object dynamics and distribution; and schedule automated periodic interactions. The 0-D or 1-D engine components that are submitted for execution via the CORBA CoG GRAM server can be interactively steered and collaboratively monitored using DISCOVER. Other examples include the control of advanced scientific instruments such as radio telescopes and synchrotron rings, via their commercially

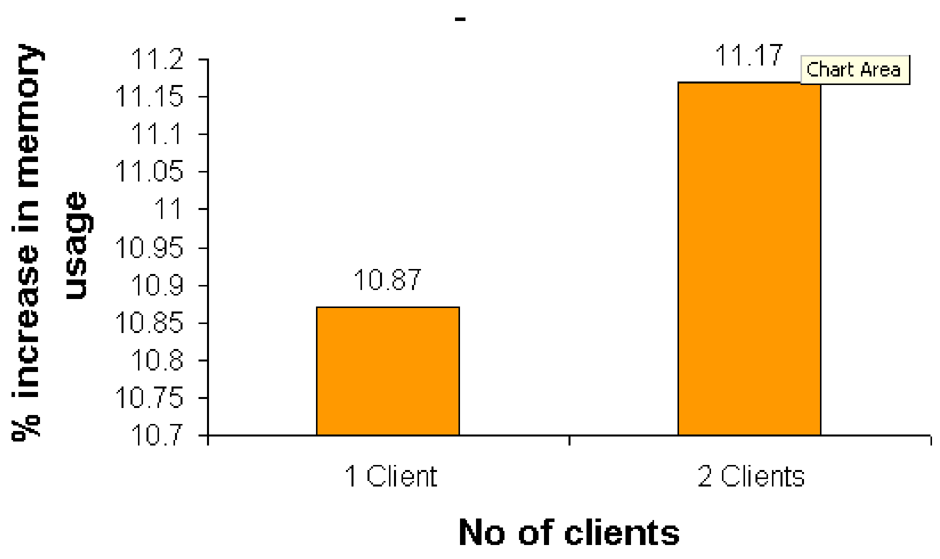


Figure 4.7: Increase in Memory Utilization on Server Side

available control infrastructure, using access through CORBA objects. At many of these installations it will not be possible to install Globus server side software but only to interface to it as a client.

4.4 Availability

The CORBA CoG toolkit is available as an open source code from
http://www.caip.rutgers.edu/TASSL/Projects/CorbaCoG/
It is required for first time users to register and henceforth they have to just enter the name and password to obtain a copy of the toolkit.

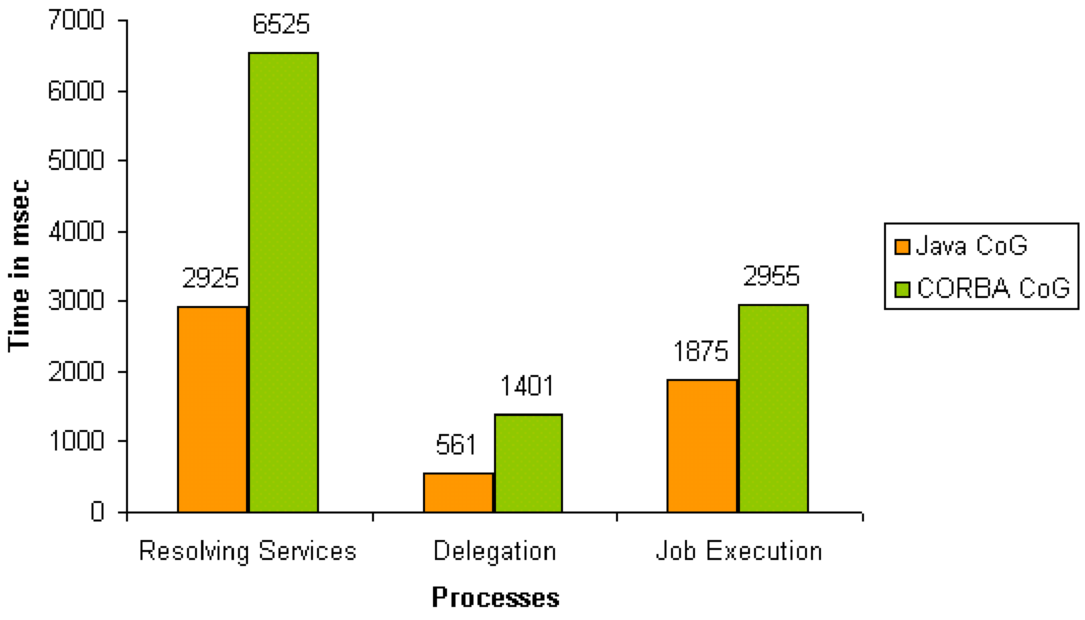


Figure 4.8: Comparison of Execution Time between Java and CORBA CoG

Chapter 5

Conclusion & Future Work

5.1 Contribution of the Thesis

While commodity distributed computing technologies enable the development of sophisticated client server applications, Grid technologies provide advanced network services for large-scale, wide area, multi-institutional environments and for applications that require the coordinated use of multiple resources. The Commodity Grid Kits bridge these two worlds so as to enable development of advanced applications that can benefit from both grid services and sophisticated commodity development environments. A lot of effort has already gone into the development of a Java, Perl and Python CoG Kit. Recently there has been an increasing demand for the creation of a CORBA CoG toolkit as various projects ranging from the creation of CORBA based control systems for advanced instruments to the computational steering of fluid dynamics codes feel the need for integrating with the Grid.

The CORBA Commodity Grid Kit presented in this thesis was one such effort at creating such a bridge. This thesis presents the design, implementation and deployment of the CORBA CoG Kit which will allow Grid application developers to exploit commodity technologies wherever possible and identifying modifications or extensions to commodity that can render them more useful for Grid Applications.

The overall goal of this thesis was to provide a framework to enable existing Grid computing environments and CORBA service providers to interoperate. CORBA targets distributed environment, is supported by many vendors, and provides transparency on many levels including languages, operating systems, networks, and protocols. It is an ideal candidate for application programmers to develop Grid-based applications. Providing a CORBA Grid domain will allow an easy integration of additional Grid services and functionality within these applications.

5.2 Challenges Faced

- The Grid is a very complex infrastructure and it took some time to grasp the complexities of the Grid. Installation of Globus was not an easy task though the Globus people are trying to make the upgrades as simple as possible. I was greatly helped on this by Paul D. Long and Viraj Bhat.
- Our approach to the toolkit design was to start with providing interface to simple services such as MDS and then move onto complex services such as GRAM, GSI. A lot of idea about the interface design was obtained from Java CoG kit. I was helped in this effort by Gregor von Lazweski.
- The toughest and the longest development time was spent on the development of the GSI Service. As the security area is new and pretty difficult to comprehend it took me a long time to understand the proxy architecture. I am extremely thankful to Jarek Gawor [24] for spending time and clearing all my queries about the proxy architecture in Globus. Added to it the different orbs came with different security features. We intially started with Visibroker [40] and then had to switch to IONA [21] as the security features integration in ORBIX was more pronounced than Visibroker. In the end we finally switched to JacORB from Adiron Ltd [1] as it was freely available open source software.

5.3 Future Work

Our future effort will concentrate on enabling applications to combine services developed by the Globus project, with the collaborative monitoring, interaction, and steering capabilities distributed with DISCOVER [8]. Our objective is to enable a scientific simulation application using the CORBA CoG Kit to discover the available resources on the network, use the GRAM service provided to run simulations on the desired high-end resources; and use DISCOVER Web-portals to collaboratively monitor, interact with, and steer the application.

References

[1] Adiron Secure System Design. http://www.adiron.com/.

[2] D. C. Arnold and J. Dongarra. The Netsolve Environment: Progressing Towards The Seamless Grid. In *Proc. of the International Workshop on Parallel Processing*, pages 199–206, 2000.

[3] B. Blakley and R. Blakley and R. M. Soley. *CORBA Security: An Introduction to Safe Computing With Objects.* Addison Wesley, Reading, Massachusettes, 1999.

[4] J. Bester, I. Foster, C. Kesselman, J. Tedesco, and S. Tuecke. Gass: A Data Movement and Access Service for Wide Area Computing Systems. In *Proc. of the 6th Annual Workshop on I/O in Parallel and Distributed Systems*, pages 77–78, 1999.

[5] Common Object Request Broker Architecture. http://www.omg.org.

[6] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid Information Services for Distributed Resource Sharing. In *Proc. of the 10th IEEE International Symposium on High Performance Distributed Computing*, pages 181-184, August 2001.

[7] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A Resource Management Architecture for Metacomputing Systems. In *Proc. of the IPPS/SPDP'98 Workshop on Job Scheduling Strategies for Parallel Processing*, pages 62–82, 1998.

[8] DISCOVER. http://www.discoverportal.org.

[9] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, and S. Tuecke. A Directory Service for Configuring High-Performance Distributed Computation. In *Proc. of the 6th IEEE Symp. on High-Performance Distributed Computing*, pages 365–375, 1997.

[10] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. A Security Architecture for Computational Grid. In *Proc. of the 5th ACM Conference on Computer and Communications Security Conference*, pages 83–92, 1998.

[11] G. v. Laszewski and I. Foster and J. Gawor and P. Lane. A Java Commodity Grid Kit. *Concurrency and Computation: Practice and Experience*, 13(8-9):643–662, 2001.

[12] Grid Forum. http://www.gridforum.org.

[13] Globus. http://www.globus.org.

[14] G. v. Laszewski. www-fp.mcs.anl.gov/g̃regor/.

[15] A. Grimshaw, W. Wulf, J. French, A. Weaver, and P. Jr. Reynolds. The Next Logical Step Toward a Nationwide Virtual Computer. In *Technical Report CS-94-21, Department of Computer Science, University of Virginia*, 1994.

[16] H. Prot and M. Bouet and V. Breton and S. Du and N. Jacq and Y. Legre and R. Medina and R. Metery and J. Montagnat. *A Virtual Laboratory for Bioinformatics on the GRID.* European Community document, 2001.

[17] I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *International Journal of Supercomputer Applications*, 11(2):115–128, 1997.

[18] I. Foster and C. Kesselman and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of Supercomputer Applications*, 15(3), 2001.

[19] Iaik. http://www.jcewww.iaik.tu-graz.ac.at/.

[20] Internet Engineering Task Force. http://www.ietf.org/.

[21] Iona Technologies. http://www.iona.com/docs/manuals/orbix.

[22] K. R. Jackson. Pyglobus:A Python Interface to the Globus Toolkit http://www.cogkits.org/papers/c545python-cog-cpe.pdf.

[23] Jacorb. http://www.jacorb.org.

[24] J. Gawor. www.mcs.anl.gov/g̃awor.

[25] Java. http://www.java.sun.com.

[26] The Grid Portal Development Kit. http://dast.nlanr.net/Projects/GridPortal.

[27] JAVA Naming and Directory Interface. http://java.sun.com/products/jndi.

[28] S. Kaur, V. Mann, V. Matossian, R. Muralidhar, and M. Parashar. Engineering a Distributed Computational Collaboratory. In *Proc. of the 34th Hawaii Conference on System Sciences*, page 6, January 2001.

[29] Kerberos: The Network Authentication Protocol. http://web.mit.edu/kerberos/www/.

[30] U. Lang, D. Gollmann, and R. Schreiner. Security Attributes in Corba. *Submitted to IEEE Symposium on Security and Privacy*, 2001.

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

[32] M. Litzkow, M. Livny, and M. Mukta. Condor A Hunter of Idle Workstations. In *Proc. of the 8th International Conference on Distributed Computing Systems*, pages 104–111, 1998.

[33] I. Lopez, G. J. Follen, R. Gutierrez, I. Foster, B. Ginsburg, O. Larsson, and S. Tuecke. Using Corba and Globus to Coordinate Multidisciplinary Aeroscience Applications. In *Proc. of the NASA HPCC/CAS Workshop*, 2000.

[34] S. Mock, M. Thomas, and G. v. Laszewski. The Perl Commodity Grid Toolkit. http://www.cogkits.org/papers/CPE_Perl_CoG_submitted.pdf.

[35] NPSS. http://www.nas.nasa.gov/SC2000/GRC/npssseat.html.

[36] J. Sang, C. Kim, and I. Lopez. Developing Corba Based Distributed Scientific Applications from Legacy Fortran Applications. In *Proc. of the HPCC Computational Aerosciences (CAS)*, pages 13–30, 2000.

[37] SESAME V4 - Overview. http://www.sesame.com.

[38] SSL. http://openssl.org.

[39] G. v. Laszewski, Ian Foster, and Jarek Gawor. Cog kits: A Bridge between Commodity Distributed Computing and High-Performance Grids. In *Proc. of the ACM 2000 Java Grande Conference*, pages 97–106, 2000.

[40] Visibroker. http://www.borland.com/bes/visibroker/.

[41] Y. Wang and F. D. Carlo and D. Mancini and I. McNulty and B. Tieman and J. Bresnahan and I. Foster and J. Insley and P. Lane and G. v. Laszewski and C. Kesselman and M. H. Su and M. Thiebaux. A High-Throughput x-ray Microtomography System at the Advanced Photon Source. *Review of Scientific Instruments*, pages 2062–2068, 2001.