Implementing Secure Grid Services using Globus Toolkit

A Thesis

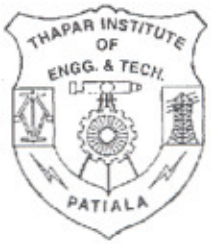
Submitted in partial fulfillment of the requirements

for the award of degree of

MASTER OF ENGINEERING

in

SOFTWARE ENGINEERING



Under the supervision of
Dr. SEEMA BAWA
Professor
Computer Science and Engineering Department
Thapar Institute of Engineering and Technology

Submitted By:
RAJWANT KAUR AHUJA
Roll No: 8033118

COMPUTER SCIENCE & ENGINEERING DEPARTMENT,

THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY
(DEEMED UNIVERSITY)
PATIALA- 147004
May, 2005

Abstract

In Collaborative environment, security is very important. Numbers of threats exists, when communicating over network of computers, through which security can be broken and utilization of resources is done by intruder. Distributed Systems are those in which multiple different components of an application communicate and coordinate their action by passing messages. Numbers of resources, huge computing power and data are available in distributed systems environment. Hackers are looking for such useful systems only. Different techniques are implemented to secure distributed systems. A Network cannot be 100 percent secured, achieving security in distributed system is bit difficult but somewhat achieved with the use of various security techniques. Grid is an example of distributed system.

"Grid enables the sharing, selection, and aggregation of geographically distributed resources dynamically at run time depending on their availability, capability, performance, cost, user quality-of –self-service requirement."

The heterogeneous nature of resources and their differing security policies are complicated and complex in the security schemes of a Grid Computing environment. The aim of thesis is to provide awareness about existing security techniques in distributed systems, and how they are not sufficient if there is need to secure Grid environment. This thesis focuses on the basic Grid security problem, its requirements, Grid Security Infrastructure (GSI) and GT3 model for Open Grid Service Architecture (OGSA).

Web services are distributed computing technology. They are used for client/server applications. Web Services are platform-independent and language-independent and use HTTP for transmitting messages. But include the overhead of transmitting all data in XML and lack of versatility. Extension of web service is grid service. Grid service fit in grid environment. Grid environment is set up on the Windows Operating System. Number of different software is required to set up the environment. Grid service is

developed on the grid environment and then made available to the number of grid users. Users can access the grid service once they have properly installed it in the grid service browser. Grid service Container contains all the services which are deployed by the developers of the grid service and are ready to be used by the users. There are proper steps to implement the grid service.

The available services can be used by the users but when we want our service to be secured enough and have limited number of users, then need to create secure grid service. There do exists ways to make grid service secured. The number of statements is included in the files to make it secure and login window is added to check the authority of the person that whether he do have right to access the particular service from the container. This thesis implements the secured grid service.

Candidate's Declaration

I hereby certify that the work which is being presented in the thesis entitled "IMPLEMENTING SECURE GRID SERVICE USING GLOBUS TOOLKIT", in partial fulfillment of the requirement for the award of degree of MASTER OF ENGINEERING in SOFTWARE ENGINEERING and submitted in Computer Science and Engineering Department of THAPAR INSTITUTE OF ENGINEERING & TECHNOLOGY (DEEMED UNIVERSITY), PATIALA is an authentic record of my own work carried out under the supervision and guidance of Dr. Seema Bawa.

The matter Presented in this thesis has not been submitted by me for the award of degree of any other degree of this or any other University.

Rajwant

(Rajwant Kaur Ahuja)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

Seema Bawa 26/05/05

(Dr. SEEMA BAWA)

Professor, Supervisor,
Computer Science & Engineering Department,
Thapar Institute of Engineering &Technology,
Patiala-147004.

Countersigned By

Mr. R.S. Salaria
Assistant Professor & Head,
Computer Science & Engineering Department,
Thapar Institute of Engineering &Technology,
Patiala.

Dr. D.S. Bawa
Dean (Academic Affairs),
Thapar Institute of Engg. and Tech.,
Patiala.

Acknowledgement

It is my proud privilege to acknowledge with respectful gratitude and heartiest thanks to my supervisor **Dr. Seema Bawa**, Professor, Computer Science & Engineering Department, whose help, stimulating suggestions and encouragement helped me in all the time of research for and writing of this thesis. I would like to thank her for her personal involvement in guiding me right from the start to the successful completion of thesis.

I am highly indebted to **Mr. Maninder Singh**, Assistant Professor, Computer Science & Engineering Department, Thapar Institute of Engineering and Technology, Patiala for his guidance that inspired me to prepare the thesis in the present form.

I also take the opportunity to thank **Mr. R. S. Salaria, HOD,** Computer Science and Engineering Department, TIET, Patiala, and the entire faculty and staff for their help, inspiration and moral support, which went a long way in successful completion of my thesis.

I would like to express my gratitude to my friends whose support has been a constant source of inspiration.

Rajwant

Place: Patiala (**Rajwant Kaur**)

Dated:

Contents

List of Figures

Organization of Thesis

The first chapter provides the introduction about the Grid computing and the architecture and infrastructure of Grid. It also includes the introduction about the security in distributed system. The second chapter includes various security techniques in distributed system and shows how they are not sufficient for large distributed system i.e. grid. This chapter also highlights the Grid Security requirements. Third chapter shows the steps to create the simple and secured grid service. Fourth chapter "Implementation of Grid Service" highlights the work carried out including the environment setup and implementing grid service i.e. simple BasicCalculatorService and Secured BasicCalculatorService. Finally, concluding thesis in Fifth chapter with future scope and directions.

Chapter 1

Introduction

1.1 GRID COMPUTING

With the advent of new technology, it has been realized that parallel sequential applications could yield faster results and at a lower cost. It was not possible for everyone to possess multiprocessor systems. It was noticed that large number of computers are used within a single organization; means huge processing power is available which remains underutilized, most of the time. A type of computing which make use of unutilized processing power then came into existence and was known as Distributed Computing. In Distributed computing, different component and object comprising an application can be located on different computers connected to a network. So, for example, a word processing application might consist of an editor component on one computer, a spell-checker object on a second computer, and a thesaurus on a third computer. In some distributed computing systems, each of the three computers could even be running a different operating system. In Distributed Computing, the problem to be solved is divided into numerous tasks that are then distributed to various computers for processing. The computers within this environment are connected through a Local Area Network (LAN). The tasks are then executed in parallel to each other. Parallelism can be either Data Parallelism or Function Parallelism. Data Parallelism means that each computer performs the same functions but for a different data set whereas in Functional Parallelism, each computer performs a different function for different or same data set.

As more and more resource intensive applications (compute-intensive and data-intensive) were developed, a need for larger amount of resource sharing was felt. The concept of using multiple distributed resources to cooperatively work on a single application has been around for several decades. As early as the late seventies work was being done toward "networked operating systems". In the late eighties and early nineties, this returned as distributed operating systems. Shortly thereafter the field of heterogeneous computing came to play: thousands of heterogeneous resources running

hundreds of tasks. The next derivative of this area was known as parallel distributed computing in which parallel codes ran on distributed resources. This became metacomputing, and then computing on the Grid [5].

"Grid is type of parallel and distributed system that enables the sharing, selection, and aggregation of geographically distributed resources dynamically at run time depending on their availability, capability, performance, cost, user quality-of –self-service requirement."

Grid Computing distinguishes from conventional distributed computing by its focus on large-scale resource sharing and in some cases high performance orientation [5].

- Grids focus on site autonomy. One of the underlying principles of the Grid is that a given site must have local control over its resources, which users can have an account, usage policies, etc.
- Grids involve heterogeneity. Instead of making every administrative domain adhere to software and hardware homogeneity, work on the Grid is attempting to define standard interfaces so that any resource speaking a defined set of protocols can be used.
- Grids involve more resources than just computers and networks. Grid computing today is as much about the data as it is about the compute cycles – where is the data, how to store terabyte data sets, how to replicate the needed pieces of data and access them through the network.
- Grids focus on the user. This is perhaps the most important, and yet the most subtle, difference. Previous systems were developed for and by the resource owner in order to maximize utilization and throughput. In Grid computing, the specific machines that are used to execute an application are chosen from the user's point of view, maximizing the performance of that application, regardless of the effect on the system as a whole.

The concept of computational Grid has been inspired by the "Electric Power Grid", in which a user could obtain electric power from any power station present on the electric grid irrespective of its location, in an easy and reliable manner. When a user

needed additional electricity, he was required to just plug into an Electric Power Grid and receive electricity on demand. Similar to electric Power grid, in a computational power grid, the user needs to plug into a computation grid in order to access additional computing power on demand. The user submits his job to the Grid resources through an interface, which is usually graphical or web enabled that serve as a portal to the grid. Special software for grid management does exist. This software is also known as Grid Middleware, accepts the job, locates best suitable and available resource and submits the job to that resource. Finally, it collects the results and gives back to the user [7].

In general, Grid Computing envisions the coupling of geographically distributed resources to offer consistent, inexpensive, location-agnostic access to a wide variety of resources, thereby enabling the aggregation and sharing of things such as supercomputers, computer cluster, distributed database, nodes on an arbitrary network and ultimately people.

The goal of grid computing, which gets its name from its grid like architecture, is to link surplus computing power and other spare IT resources with clients who have periodic needs beyond the capacity of their machines. It's a form of peer-to-peer computing. Grid computing software divides a task into subtasks, finds spare processors and other critical resources on the network, distributes the subtasks, monitors their progress and restarts any subtasks that fail. Finally, grid computing engines aggregate the results of the subtasks so the job or task can be completed.

Grid enables the selection and sharing of a wide variety of geographically distributed resources including supercomputers, storage systems, data sources, high Performance computers and other resources owned by different organizations for solving compute intensive and data intensive problems. The logic behind grid computing is to borrow the idle computational power of resources and not to buy new resources. The real basis of grid computing is coordinated resource sharing and problem solving in a dynamic, multi-organizational environment. Sharing in a grid is not just simple sharing of files but of hardware, software, data and other resources. Thus, a complex yet SECURE SHARING is at the heart of the Grid. The needs of the grid range from client-server to peer-to-peer architecture, from single user to multi-user systems, from sharing file to

sharing resources, etc and all these in a dynamic, controlled and secured manner. Current distributed computing technologies do not fulfill all these needs. As grid computing focuses on dynamic and cross-organizational sharing, it enhances the existing distributed computing technologies. Grid provides full security to their users. And therefore security is the main issue in grid computing. Application and storage service providers provide hardware and software accessing capabilities to the customer, but through a Virtual Private Network (VPN). The VPN again does not satisfy the security criteria and moreover has a static configuration, which means that once subscribed the user can use only those resources that are present in the VPN. Enterprise Computing Systems like CORBA, DCOM, support resource sharing and remote method invocation but the sharing is relatively static and restricted within a single organization.

1.2 VIRTUAL ORGANIZATION (VO) IN GRID

In Grids, Increasingly, independent institutions with similar goals and interests are forming loosely coupled *virtual organizations* for collaboration and resource sharing. Virtual Organizations provide a highly controlled environment to allow each resource provider to specify exactly what they want to share, who is allowed to share it and the conditions whereby this sharing occurs. The set of individuals and/or institutions that provide such sharing rules are collectively known as a virtual organization (VO).

A VO is a community of resource providers and users from multiple administrative domains, collaborating in order to achieve common objectives. Virtual organizations need to share resources such as data archives, computer cycles, and networks, underlying any sharing mechanism is the ability to *authenticate* the identity of the requestor of a resource, and to determine if the requestor is *authorized* to make the resource request.

The following VO example have a virtual organization (VO) comprising several institutions collaborating in a Grid project. These institutions can be Academic, Governments, and Industrial and Commercial institutions. We assume that these institutions are geographically distributed. Fig 1.1 illustrates the Grid Infrastructure and the collaborating partners' characteristics [13].

Each institution in the project has a local security policy that governs access to its local resources. Although these institutions are partners in the VO, not all their users are members of the VO (denoted Grid users in Fig 1.1). In addition, not all resources are shared with the VO (denoted Grid resources in Fig 1.1).

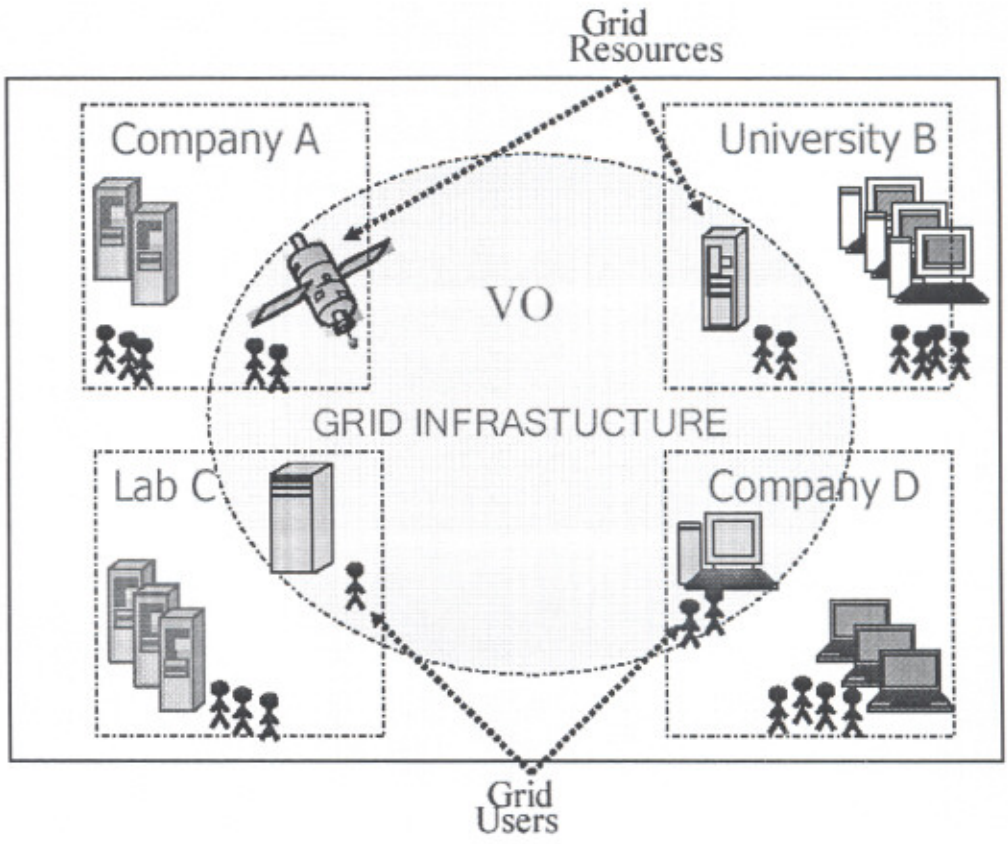


Fig 1.1: A typical Virtual Organization and Grid Infrastructure [13]

Some resources are restricted to the VO and only accessible by local users. Each institution has a local intranet security solution such as Kerberos or Public Key Infrastructure (PKI).

1.3 GRID ARCHITECTURE

1.3.1 Open Grid Services Architecture (OGSA)

The Open Grid Services Architecture (OGSA), developed by The Global Grid Forum, aims to define a common, standard, and open architecture for grid-based

applications. The goal of OGSA is to standardize practically all the services one finds in a grid application (job management services, resource management services, security services, etc.) by specifying a set of standard interfaces for these services. In other words, if OGSA (for example) defines that the JobSubmisionInterface has a submitJob method, there has to be a common and standard way to *invoke* that method if we want the architecture to be adopted as an industry-wide standard. Any distributed middleware (CORBA, RMI, or even traditional RPC) can act as *base* for the architecture. For reasons, Web Services were chosen as the underlying technology. Web Services Architecture was certainly the best option, but it still had several shortcomings which made it inadequate for OGSA's needs. OGSA overcame this obstacle by defining an extended type of Web Service called *Grid Service* (Shown in the Fig 1.3: Grid Services are defined by OGSA). A Grid Service is simply a Web Service with a lot of extensions that make it adequate for a grid-based application (and, in particular, for OGSA). In the diagram: Grid Services are an extension of Web Services. Finally, since Grid Services are going to be the distributed technology underlying OGSA, it is also correct to say that *OGSA is based on Grid Services.*

The grid infrastructure is mainly concerned with the creation, management and the application of dynamic coordinated resources and services. These dynamic and coordinated resources and services are complex. Even though the complexity and difference in resources and services may vary within every virtual organization, they all are agreed to deliver a set of Quality of Service (QoS) features including common security semantics, workflow, resource management, problem determination, failover, and service-level management. These QoS features require a well-defined architecture to achieve the desired level of service quality. This prompted for the introduction of OGSA to support the creation, maintenance and application of ensembles of services maintained by virtual organizations.

OGSA is proposed set of standards for ensuring quality of service across a grid computing network. The OGSA is a proposed Grid service architecture based on the integration of Grid and Web Services concepts and technologies. OGSA services provide a virtualization layer on top of the real computing resource, making it possible to create dynamic management and configuration of virtual organizations [12].

OGSA Architecture

OGSA Architecture is a layered architecture as shown in Fig 1.2, with clear separation of functionalities at each layer. The core architecture layers are OGSI, which provides the base infrastructure and OGSA core platform services, which are a set of standard services including policy, logging, and service-level management and so on. The high level applications and services use these lower layer core platform components and OGSI that become part of a resource-sharing grid.

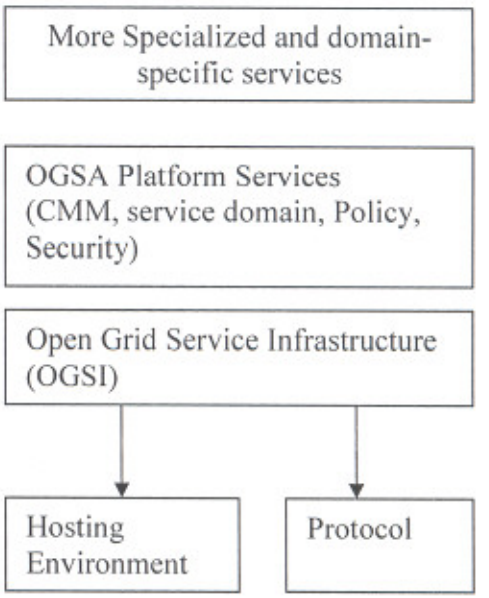


Fig 1.2: OGSA platforms Architecture [6]

OGSA defines standard Web service interfaces and behaviors that add to Web services the concepts of stateful services and secure invocation. With the release of Version 3 of the Globus Toolkit (GT3), the Globus Project offers an open source implementation of version 1 of the OGSI Specification, a key building block in the OGSA framework [6].

1.3.2 Grid Service Infrastructure (OGSI)

OGSI refers to the base infrastructure on which OGSA is built. The OGSI is a grid software infrastructure standardization initiative, based on the emerging Web services standards that are intended to provide maximum interoperability among OGSA software components [6]. At its core is the "Grid Service Specification", which defines the standard interfaces and behaviors of a Grid service, building on a Web services base. OGSI is the concrete specification of OGSA infrastructure. It is the middleware, the "Java 2 Platform" for Grid services. It defines how to build a Grid service, outlining the mechanisms for creating, managing, and exchanging information for Grid services [12].

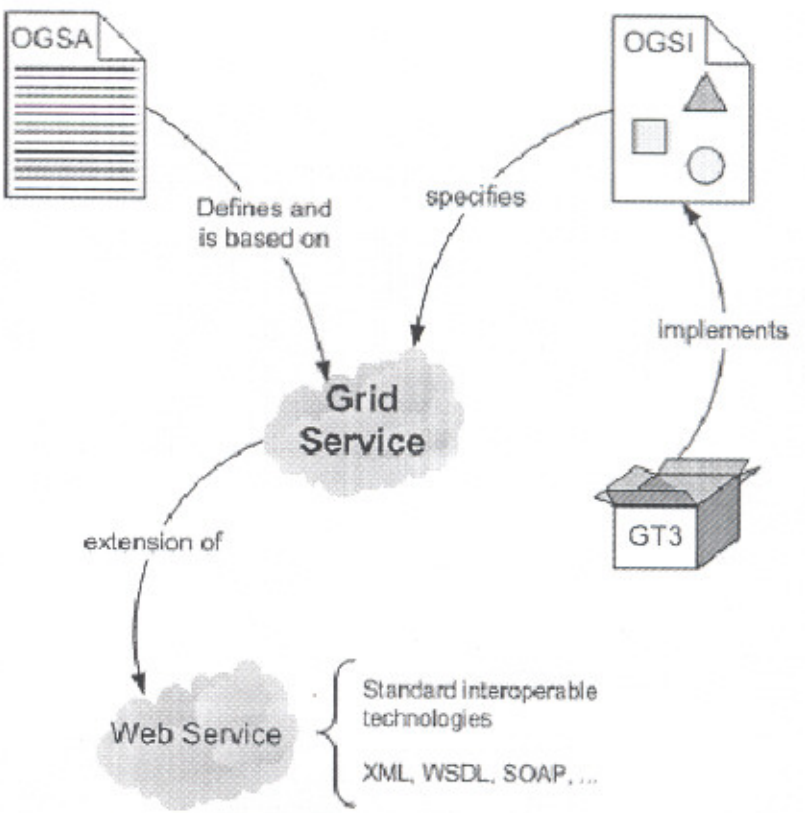


Fig 1.3: Relationship btw OGSA, OGSI and GT3 [20]

OGSI specification and other elements of the OGSA within the Global Grid Forum include new challenges and opportunities for Grid security. Integration of GSI with OGSA enables the use of Web services techniques to express and publish policy,

allowing applications to determine automatically what security policies and mechanisms are required. Implementing security in the form of OGSA services allows those services to be used as needed by applications to meet requirements. Advanced hosting environments enable security functionality to be implemented outside of the application, simplifying development. In terms of grid services, OGSA alone doesn't go into much detail when describing Grid Services. It basically outlines what a Grid Service should have (that Web Services don't) but little else. That is why OGSA spawned another standard called the OGSI which gives a formal and technical specification of what a Grid Service is. In Fig 1.3: Grid Services are specified by OGSI (as opposed to simply 'being defined' by OGSA) [20].

1.3.3 Globus Toolkit Version 3 (GT3)

The Globus Toolkit is a software toolkit, developed by The Globus Alliance, which is used to program grid-based applications. The third version of the toolkit (GT3) includes a complete implementation of OGSI (**GT3 implements OGSI**). However, it's very important to understand that GT3 isn't *only* an OGSI implementation. It includes a whole lot of other services, programs, utilities, etc. Some of them are built *on top of OGSI* and are called the *WS (Web Services) components*, while other are not built on top of OGSI and are called the *pre-WS components*.

Globus Toolkit 3.0 (GT3) is the reference implementation of the OGSI. It provides the foundation to build grid services based on the OGSA and Web services architecture. These services enable resource sharing, job scheduling and composing, and accessing various distributed applications on the grid [19]. For applications in grid environments, security is extremely important. It is necessary to restrict access to resources and data based on various policies, organizational and otherwise. Thus, grid services and clients need to be mutually authenticated and must have their access to the services authorized.

Globus is using OGSI as their infrastructure for their GT3 base services. They also add some management services. OGSI represents the reference open source standard implementation of the OGSI standard. At least one service of each pillar should be

included in most of Grid implementations. These key areas are resource management, information services, and data management [12].

Grid Services are only a small (but important!) part of the whole GT3 Architecture (Fig 1.4), which offers developers plenty of services with Grid programming.

- OGSI (i.e. "Grid Services") is the '**GT3 Core**' layer.
- **GT3 Security Services**: Security is an important factor in grid-based applications. GT3 Security Services helps to restrict the access to Grid Services, so only authorized clients can use them. For example, only limited location's office can access a service. The usual security measures (putting the web server behind a firewall, etc.) GT3 gives one more layer of security with technologies such as SSL and X.509 digital certificates.

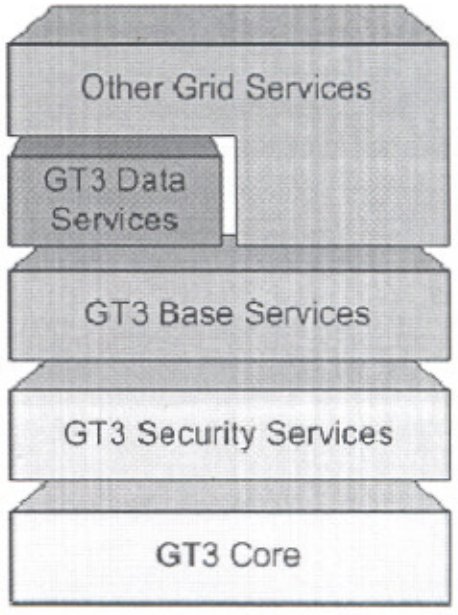


Fig 1.4: GT3 Architecture [20]

• **GT3 Base Services**: This layer actually includes a whole lot of interesting services:

  • **Managed Job Service**: Suppose some particular operation in MathService might take hours or even days to be done. Of course, we don't want to simply

stand in front of a computer waiting for the result to arrive (especially if, after 8 hours of waiting, all we get might simply be an error message!) We need to be able to check on the progress of the operation periodically, and have some control over it (pause it, stop it, etc.) This is usually called *job management.*

• **Index Service**: We usually know what type of Web Service we need, but we have no idea of where they are. This also happens with Grid Services: we might know we need a Grid Service which meets certain requirements, but we have no idea of what its location is. While this was solved in Web Services with UDDI, GT3 has its own Index Service. For example, we could have several dozen MathServices all around the country, each with different characteristics (some might be better suited for statistical analysis, while others might be better for performing simulations). Index Service will allow us to query what MathService meets our particular requirements.

• **Reliable File Transfer (RFT) Service**: This service allows us to perform large file transfers between the client and the Grid Service. For example, suppose we have an operation in MathService which has to crunch several gigabytes of raw data (ex, for a statistical analysis). Of course, we're not going to send all that information as parameters. We'll be able to send it as a file. As its name signifies, RFT guarantees the transfer will be reliable. For example, if a file transfer is interrupted (due to a network failure, for example), RFT allows us to restart the file transfer from the moment it broke down, instead of starting all over again.

• **GT3 Data Services**: This layer includes *Replica Management*, which is very useful in applications that have to deal with very big sets of data. When working with large amount of data, we're usually not interested in downloading the whole thing; we just want to work with a small part of all that data. Replica Management keeps track of those *subsets* of data we will be working with.

• **Other Grid Services**: Other non-GT3 services can run on top of the GT3 Architecture.

1.3.4 Difference between OGSA, OGSI, and GT3

Consider the simple example. Suppose we want to build a new house. The first thing we need to do is to hire an architect to draw up the blueprints, so we can get an idea of what our house will look like. Once we're happy with the architect's job, it's time to hire an engineer who will plan all the construction details (where to put the master beams, the power cables, the plumbing, etc.). The engineer then passes all his plans to qualified professional workers (construction workers, electricians, plumbers, etc) who will actually build the house. We could say that OGSA (the definition) is the blueprints the architect creates to show what the building looks like, OGSI (the specification) is the structural design that the engineer creates to support the architect's vision for the building, and GT3 is the bricks, cement, and beams used to build the building to the engineer's specifications.

1.4 WEB SERVICE

Web Services are *yet another* distributed computing technology (like CORBA, RMI, EJB, etc.) They allow us to create client/server applications. For example, develop an application for a chain of stores, these stores are all around the country, but my master catalog of products is only available in a database at central offices, yet the software at the stores must be able to access that catalog. Then *publish* the catalog through a Web Service called *ShopService*. The *clients* (the PCs at the store) would then contact the *Web Service* (in the *server*), and send a *service request* asking for the catalog.

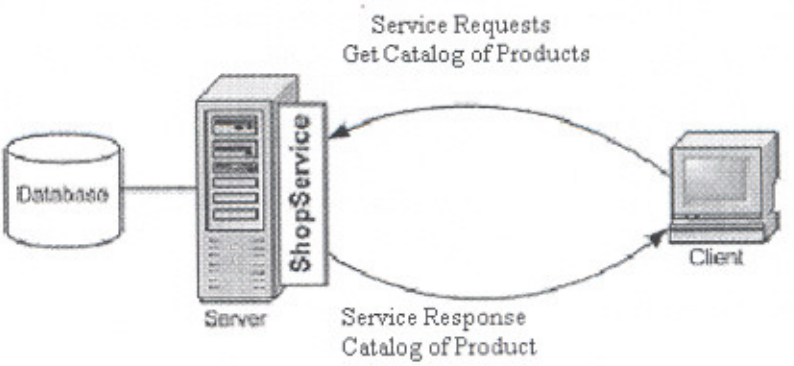


Fig 1.5: Client/Server Request/response

The server would return the catalog through a *service response*. Web Services have certain advantages over other technologies:

• Web Services are platform-independent and language-independent, since they use standard XML languages. This means that my client program can be programmed in C++ and running under Windows, while the Web Service is programmed in Java and running under Linux.

• Most Web Services use HTTP for transmitting messages (such as the service request and response). This is a major advantage if want to build an Internet-scale application, since most of the Internet's proxies and firewalls won't mess with HTTP traffic (unlike CORBA, which usually has trouble with firewalls)

Web Services also have some disadvantages:

• *Overhead.* Transmitting all data in XML is obviously not as efficient as using a proprietary binary code. What you win in portability, you lose in efficiency. Even so, this overhead is usually acceptable for most applications, but you will probably never find a critical real-time application that uses Web Services.

• *Lack of versatility.* Currently, Web Services are not very versatile, since they only allow for some very basic forms of service invocation. CORBA, for example, offers programmers a lot of supporting services (such as persistency, notifications, lifecycle management, transactions, etc.)

1.4.1 A Web Service Invocation

A web service is invoked in the following way:

1. A client may have no knowledge of what Web Service it is going to invoke. So, first step will be to *find* a Web Service that meets our requirements. For example, we might be interested in locating a public Web Service which can give me the temperature in US cities. This is done by contacting a UDDI registry Fig 1.6.

2. The UDDI registry will reply, telling the client what servers can provide the service that client require (e.g. the temperature in US cities)

3. Client now knows the location of a Web Service, but does not have idea of how to actually invoke it. Client need to know the temperature of a US city, but does not know what the actual service invocation is? So, client have to ask the Web Service to *describe* itself (i.e. tell us how exactly we should invoke it)

4. The Web Service replies in a language called WSDL.

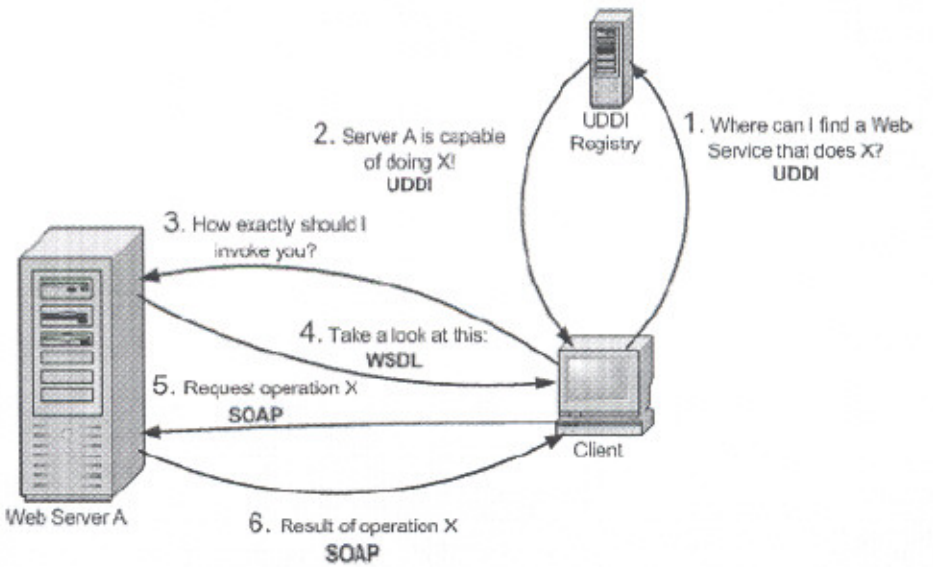


Fig 1.6: Web Service Invocation process [20]

5. We finally know where the Web Service is located and how to invoke it. The invocation itself is done in a language called SOAP. Therefore, client sends a *SOAP request* asking for the temperature of a certain city.

6. The Web Service will kindly reply with a *SOAP response* which includes the temperature client asked for, or maybe an error message if our SOAP request was incorrect.

1.5 GRID SERVICE

Grid Services are basically Web Services with improved characteristics and services.

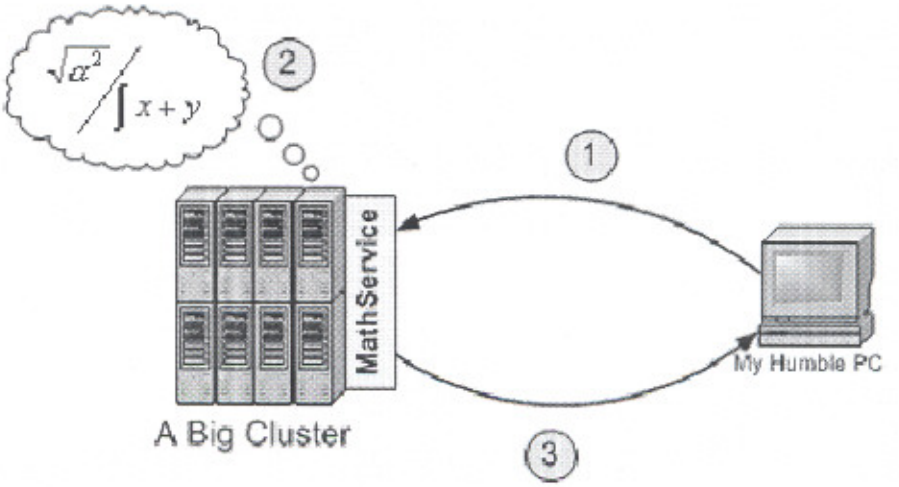


Fig 1.7: Web Service Invocations

We could implement a Math Web Service called *MathService* which offered operations such as SolveReallyBigSystem(), SolveFermatsLastTheorem(), etc. At first, perform typical Web Service invocations:

1. Invoke MathService, asking it to perform a certain operation.
2. MathService will instruct the cluster to perform that operation.
3. MathService will return the result of the operation.

Web Services are *stateless*. "Stateless" means that Web Services can't remember what you've done from one invocation to another. If we wanted to perform a chain of operations, we would have to get the result of one operation and send it as a parameter to the next operation. Furthermore, even if we solved the stateless problem, Web Services are still *non-transient*, which means that they *outlive* all their clients.

Web Services are also referred to as *persistent* because their lifetime is bound to the Web Services container (a Web Service is available from the moment the server is started, and doesn't go down until the server is stopped). Web Services are not suited for the collaborative environment like grid [20].

Grid Services solve both problems by allowing programmers to use a *factory/instance* approach to Web Services. Instead of having one big stateless MathService shared by all users, we could have a central MathService factory in charge of maintaining a bunch of MathService instances. When a client wants to invoke a MathService operation, it will talk to the instance, not to the factory. When a client needs

a new instance to be created (or destroyed) it will talk to the factory. Fig 1.8 shows that a client can share number of instances, and can have more than one instance at a time.

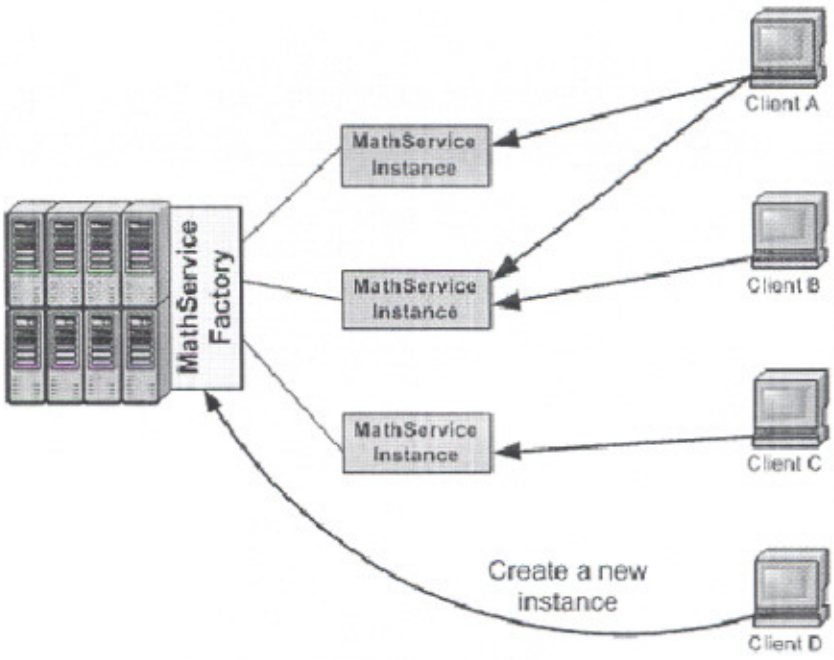


Fig 1.8: Creating Service Instance

These instances are *transient*, because they have a limited lifetime that isn't bound to the lifetime of the Grid Services' container. In other words, we can create and destroy instances whenever we need them (instead of having one persistent service permanently available). The actual lifecycle of an instance can vary from application to application. Usually, we'll want instances to live only as long as a client has any use for them. This way, every client has its own personal instance to work with. However, there are other scenarios where we might want an instance to be shared by several users, and to self-destruct after no clients have accessed it for a certain time. Grid Services are *potentially* transient. This means that not *all* Grid Services have to use (by definition) a factory/instance approach. A Grid Service can be persistent, just like a normal Web Service. Choosing between persistent Grid Services or factory/instance Grid Services depends entirely on the requirements of an application.

1.6 EXAMPLE OF A GRID APPLICATION

A typical example of a Grid application is "weather prediction". This involves collaboration between several partners: TV stations that produce regular weather news reports, a Satellite Company that regularly provides space images of the earth, a super computing centre that rapidly analyses the images and a visualization centre that produces visual interpretations of the weather analysis (Fig 1.9).

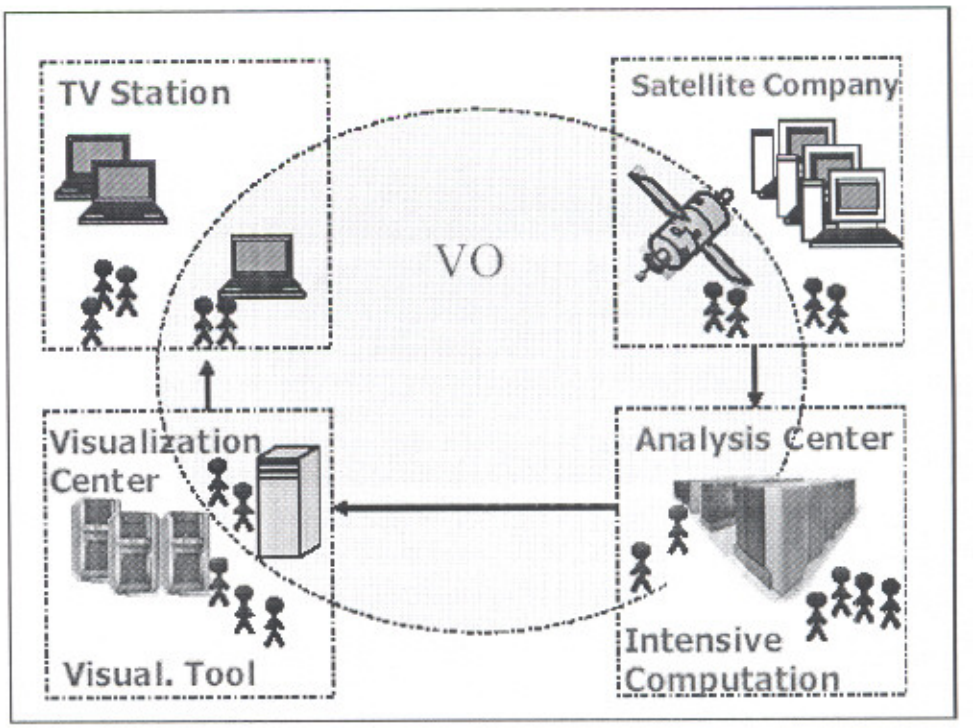


Fig 1.9 Example of Grid Application [13]

The smooth running of this project for the timely production of regular weather reports crucially depends on appropriate schemas for securely sharing, exchanging, and coordinating information between these partners [13].

The power of Grid is particularly useful in arenas involved in intensive processing such as life science research, financial modeling, industrial design, and graphics rendering. Many governments have recently initiated special programs to support the Grid: UK e-science programme is funding project such as Reality Grid. EU is funding

projects such as European Data Grid, EuroGrid, and in US, NASA is funding an Information Power Grid and department of Energy is funding Globus project [13].

The Enabling Grids for E-sciencE (EGEE) project is funded by the European Commission and aims to build on recent advances in grid technology and develop a service grid infrastructure which is available to scientists 24 hours-a-day. The MONSOON project, a Euro-India ICT Co-operation initiative, which EGEE is supporting as Champion Leader, officially started on 1 Feb 2005.

1.7 SECURITY

Security is the main issue which need to be focused everywhere, either is a question of house or PC. With the introduction of the computer, the need for automated tools for protecting files and other information stored on the computer become evident. The generic name for the collection of tools designed to protect data and to thwart hackers is computer Security [3]. Network security measures are needed to protect data during their transmission. Once PC is connected to Internet, there are very less chances that it is secured from unwanted attacks. If one has satisfied his or her customer that particular lock will provide full security, then everyone will buy that lock. Similarly, if one has won the confidence of users to use distributed systems or Internet, then everyone can and will use easily. But full security is and cannot be provided. So, main aim is to provide such tools or technologies that can secure our computer from intruders. Security is a vital part of our operations, not a luxury.

Classic security concerns deal more with data

- *Confidentiality*: Data is only available to those authorized persons.
- *Availability*: An authorized person can get data when he wants it.
- *Integrity*: Any intruder hasn't changed Data.

Additional concerns deal more with people and transactions

- *Trust*: It ensures "Who you are" and "what you are authorized to do".
- *Non-repudiation:* One can't deny doing something what he did.
- *Auditability*: One can check what other did to the data.
- *Reliability:* The system does what one want when one wants it to.

- *Privacy:* Within certain limits no one should know who one is or what he is doing.

1.8 SECURITY IN DISTRIBUTED SYSTEMS

Security and trust are fundamental issues, which must be considered in the management and operation of distributed systems. The security of distributed systems can be achieved by securing the processes and the channels used for their interactions and by protecting the objects that they encapsulate against unauthorized access.

Since hackers constantly find new ways of exploiting security vulnerabilities in operating systems and applications, considerable time and effort must be spent to counter those threats. Perfect security cannot be achieved, but well thought-out measures can go a long way both in reducing the risk of being broken in to and minimizing the impact a potential breaking would have.

To access data and applications, a user first need to be authenticated, and then needs to be authorized to perform the operation. *Authentication Procedures* perform the former task, and *Access Control Decision* functions perform the later task.

The authentication procedures are responsible for the verification of the identity of the user - if (s) he really is who (s) he says (s) he is - because the Access Control functions depend on this. Three different types of information may be used:

- Something the user knows
- Something the user possesses
- Something the user is

The rise of the Internet has made everyone aware of the need for system security. A Communication channel is misused by following types of attacks:

- Eavesdropping: Obtaining copies of messages without authority.
- Masquerading: Sending or receiving messages using the identity of another principal without their authority.
- Replaying: Storing in intercepted messages and sending them at later date. This is sometimes, very effective.

- Denial of Service: Flooding a channel or other resource with messages in order to deny access for others.
- Message tempering: incepting messages and then alter their content before passing.

Hacker's intention is to attack distributed systems as it includes not only huge amount of resources, but also huge amount of data. Threats must be considered to make secured network of distributed systems.

Chapter 2

Literature Survey

2.1 SECURITY TECHNIQUES IN DISTRIBUTED SYSTEMS

Numbers of techniques are applied to make communication secure enough so the client's information is not hacked by any intruder. Information when transferred from one machine to another may include number of threats. To provide security in distributed systems, numbers of different techniques are used.

1 SSH
2 PGP
3 Kerberos
4 Public Key Infrastructure

2.1.1 SSH (Secure Shell)

SSH (Secure Shell) is a program to log into another computer over a network, to execute commands in a remote machine, and to move files from one machine to another. It provides strong authentication and secure communications over insecure channels. It is intended as a replacement for rlogin, rsh, and rcp. SSH provides a secure and efficient way to interface with a remote host. Much like telnet SSH allows you to log on to computer over a network and execute commands on it. Unlike Telnet, SSH encrypts all of the data, including password which is passed between computer and the remote host. This makes personal account information and any other information that is passed over the network less susceptible to hackers. TCP/IP is used as the transport. Users like SSH because it provides basic remote login and file copy capabilities without a lot of complexity [8].

SSH protects against:

- IP spoofing, where a remote host sends out packets, which pretend to come from another, trusted host. SSH even protects against a spoofer on the local network, who can pretend he is your router to the outside.
- IP source routing, where a host can pretend that an IP packet comes from another, trusted host.
- DNS spoofing, where an attacker forges name server records.
- Interception of clear text passwords and other data by intermediate hosts.
- Manipulation of data by people in control of intermediate hosts.

In other words, SSH never trusts the net; somebody hostile who has taken over the network can only force SSH to disconnect, but cannot decrypt or play back the traffic, or hijack the connection [16].

SSH Technology

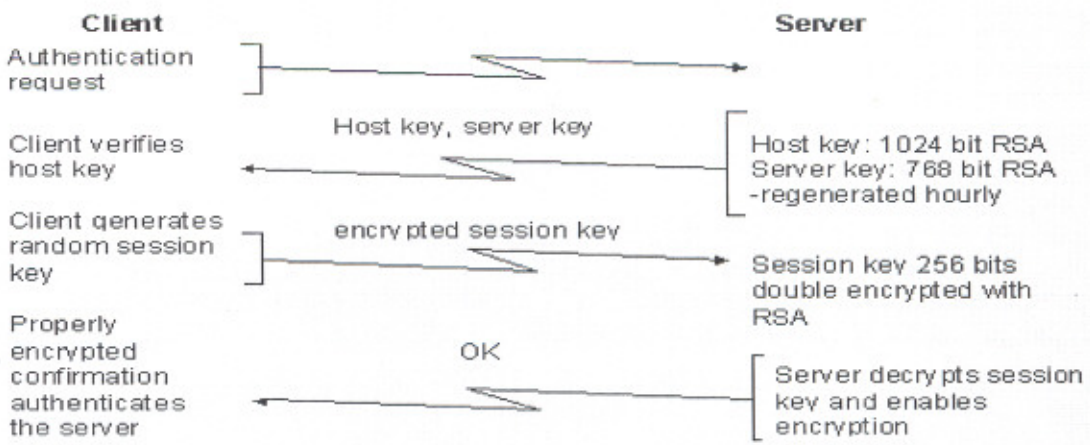


Fig 2.1: SSH Technology

2.1.2 Pretty Good Privacy (PGP)

PGP is a public-key encryption application for exchanging files or messages with confidentiality and authentication. *Privacy (confidentiality)* means that only the intended recipient of a message can read it. By providing the ability to encrypt messages, PGP

provides protection against anyone eavesdropping on the network. Even if the information is intercepted, it is completely unreadable to the snooper [8].

Authentication identifies the origin of the information, certainty that it is authentic, and that it has not been altered. Authentication also provides an extremely valuable tool in network security: verification of the identity of an individual. In addition to secure messaging, PGP also provides secure data storage, helps to encrypt files stored on computer.

Version 6.5.8 of PGP includes PGPnet - a powerful VPN client that enables secure peer-to-peer IP-based network connections - and Self-Decrypting Archives (SDAs) which allows exchanging information securely even with those who do not have PGP. It is used to protect email and files by scrambling them so others cannot read them. Encryption is the translation of data into a secret code. Encryption is the most effective way to achieve data security. To read an encrypted file, one must have access to a secret key or password that enables to decrypt it. Unencrypted data is called plain text. Encrypted data is referred to as cipher text.

Every PGP user generates a pair of unique keys, a private key (secret) and a public key (shared). The public key can be given to anyone. It is used to encrypt messages being sent. Encrypted messages being sent to you can only be decrypted using your private (secret) key. You publish your public key by sending it to a PGP key server on the Internet. There is no security risk to give away public key.

2.1.3 Kerberos

Kerberos is a distributed authentication service/protocol that allows a client to verify its identity to a server without sending data across the network that would be of value to a hacker [3]. Kerberos can also be used to ensure the confidentiality and integrity of data transmitted across a network. Kerberos provides security on physically insecure networks. Kerberos is now becoming an IETF standard for a standard authentication protocol.

Kerberos uses a standard encryption-based authentication technique with a few variations designed to increase ease of use across administrative entities and reduce the number of possible 'attacks' on the system [8]. The system uses cryptographically sealed tickets and authenticators, which may be passed over the network and decrypted only by a user or machine that knows the appropriate encryption/decryption key. Kerberos can be used to satisfy the need for an authentication service and ensure the confidentiality and integrity of network data.

System Architecture of Kerberos

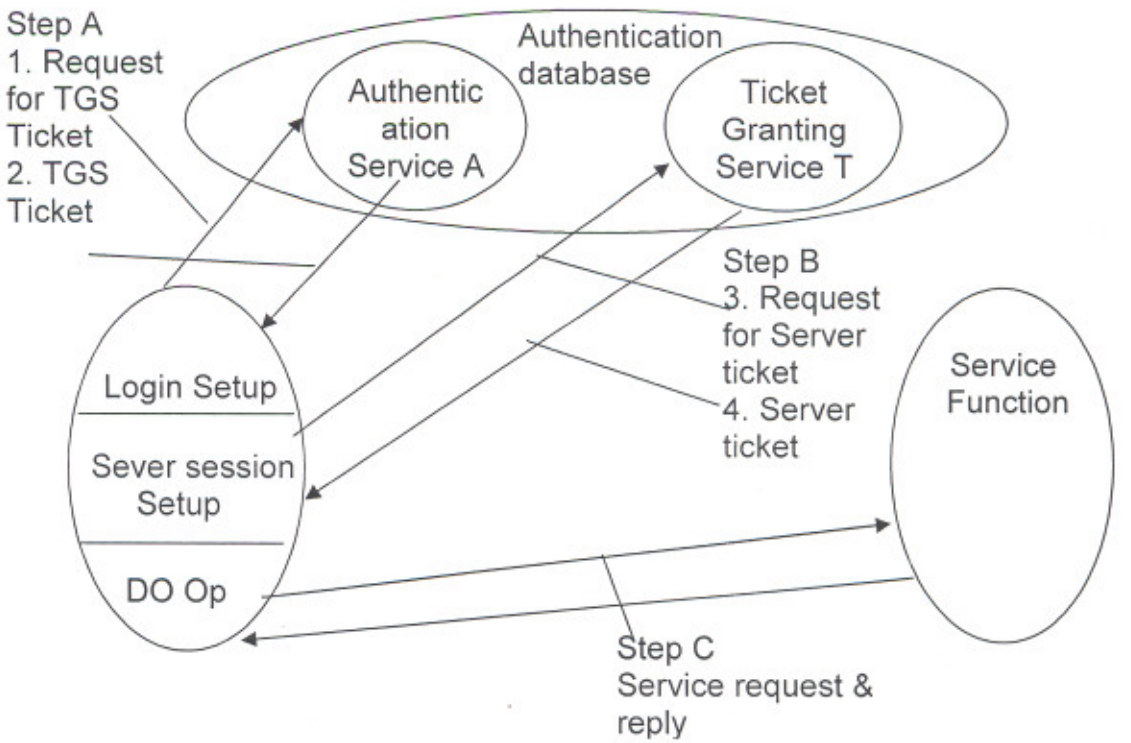


Fig 2.2: Kerberos Architecture

Kerberos deals with three kinds of security Objects [3]:

- *Tickets*: A token issued to a client by Kerberos ticket granting service for presentation to a particular server, verifying that sender has recently been authenticated by Kerberos. Tickets include expiry time and newly generated session key for user by the client and the server.

- *Authentication*: A token constructed by client and sent to server to prove the identity of the user and the currency of any communication with a server. It contains client's name, a timestamp and is encrypted session key, An Authenticator can be used only once.
- *Session Key*: A secret key is generated by Kerberos and issued to a client for use when communicating with a particular server. Encryption is not mandatory for all communication with servers; the session key is used for encrypting communications with servers.

A Kerberos server is known as Key Distribution Centre (KDC). Each KDC offers Authentication Service (AS) and Ticket Granting Service (TGS).

2.1.4 Public Key Infrastructure (PKI)

PKI is a security architecture that has been introduced to provide an increased level of confidence for exchanging information over an increasingly insecure Internet. A robust, flexible, standard, and open Public-Key Infra-structure architecture is critical to the success of secure systems based on Public-Key technology [8].

PKI include the use of encryption, digital signatures, keys, certificates, Certificate Authorities, certificate revocation and storage. PKI supports the use of Public Keys for authentication and identifying users, services, and confirming digital signatures. Public keys usually conform to the X.509 standard for certificates, and usually are based on the RSA public/private key encryption algorithm [17]. PKI means the methods, technologies and techniques that together provide a secure infrastructure and the use of a public and private key pair for authentication and proof of content. A PKI infrastructure is expected to offer its users the following benefits:

- Certainty of the quality of information sent and received electronically
- Certainty of the source and destination of that information
- Assurance of the time and timing of that information (providing the source of time is known)

- Certainty of the privacy of that information
- Assurance that the information may be introduced as evidence in a court or law

These facilities are delivered using a mathematical technique called public key cryptography that uses a pair of related cryptographic keys to verify the identity of the sender (signing) and/or to ensure privacy (encryption) [26]. At a high level, the PKI model involves certificate authorities issuing certificates with public asymmetric keys and authorities that assert properties other than key ownership. Owners of such certificates may use the associated keys to express a variety of claims, including identity. The Web services security model supports security token services issuing security tokens using public asymmetric keys. PKI is used here in the broadest sense and does not assume any particular hierarchy or model. The public keys must be stored in a directory, to ensure their worldwide availability. As they are accessible via unsecured networks (Internet), an infrastructure must be set-up to allow them to be undoubtedly trusted. This is the main objective of the Public Key Infrastructure [26].

2.2 WORKING OF A GRID

Grid working is much similar to the Internet where request and response is expected from user and server, respectively. Figure 2.3 shows the steps involved when a user requests a batch job to be run on a remote GRID system [11]. One goal of automation in a Grid system is for users to have a single sign-on, rather than having to log-on and supply a password to each resource used. This is achieved by means of a user proxy (step 1). The user proxy is given a new certificate with its own key pair. The user signs the user proxy's certificate with his private key, allowing the user proxy to act on his behalf while the certificate is valid, usually 24 hours.

In addition to the time limitation, the user may put further restrictions on what the user proxy can do by listing them inside the certificate. In order for the user to access his private key to sign the user proxy, he will have to enter his password, however this should be the only time the password is required, since the proxy's private key is not encrypted with any password. Thus, single sign-on is achieved. Once the user proxy has

been created, the user can launch the job he wants to run on the Grid. From this point forward, the Grid software will do all of the work with the user proxy acting on behalf of the user. The user proxy now connects to the Grid gatekeeper of the remote site (step 2). The gatekeeper is a process on each Grid site that listens for incoming requests from users and sends them to the appropriate resource. First, the gatekeeper and the user proxy perform mutual authentication. They challenge each other using a nonce. The user proxy then checks the host's certificate so it can trust the host and its gatekeeper. Similarly the gatekeeper checks the user proxy's certificate; however, the user and not a CA signed its certificate [11].

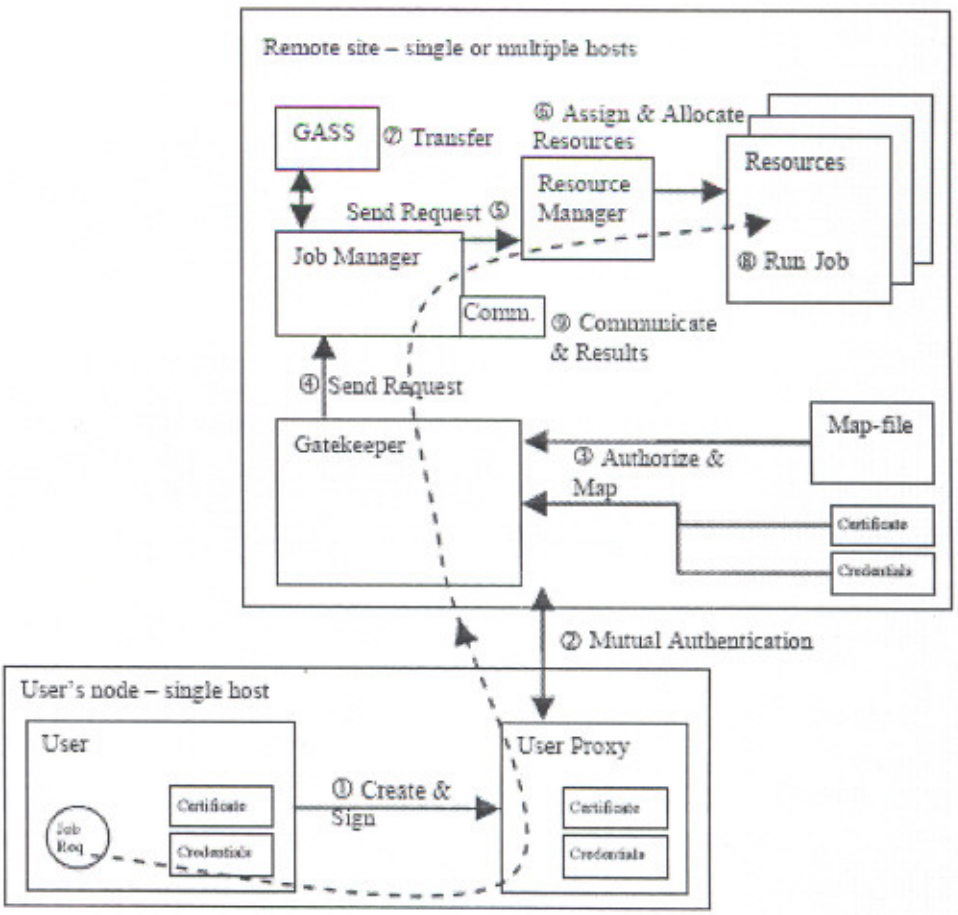


Fig 2.3: Running a job on the grid [11]

Therefore the gatekeeper first checks the user proxy's certificate against the user's certificate. Then it checks the user's certificate against the CA, which it trusts. After

mutual authentication, both the user and gatekeeper can be sure they are talking to each other and not a malicious third party.

Next the gatekeeper decides whether or not the user is allowed to use its services. This is determined by an entry in the Grid map file with the user's Grid subject (step 3). Once the user is authorized, the gatekeeper uses the entry in the map file to determine which local account the user will be mapped to. After the user is authorized and mapped to a local user, the gatekeeper sends the user's request to the job manager (step 4).

The job manager submits the user's request to the resource manager (step 5). The resource manager will determine which local resources are available to fulfill the user's request and will allocate them for the user (step 6). The job manager is responsible for all future communication with the user. The job manager will send the user a contact string that can be used to check on the status of the job or to cancel it in progress. The job manager allows the user proxy to upload data or executables that are needed. This is done with a facility called Global Access to Secondary Storage (GASS). If GASS is used, the job manager contacts the GASS server on the user's node and gets the files before starting the job (step 7). GASS is also used to keep track of the output of the user's jobs.

When all of the data and executables are present on the resources of the remote site, the job manager submits the job to the resource manager to start the job for the user (step 8). When it completes, the output is sent back to the user (step 9) [11].

2.3 SECURITY IN GRID

A major requirement for grid computing is security. At the base of any grid environment, there must be mechanisms to provide security including authentication, authorization, data encryption, and so on. The Grid Security Infrastructure (GSI) component of the Globus Toolkit provides robust security mechanisms. It also provides a single sign-on mechanism, so once a user is authenticated; a proxy certificate is created and used when performing actions within the grid. When designing grid environment, GSI sign-in may be used to grant access to the portal or may have security for the portal. The portal would then be responsible for signing into the grid, either using the user's credentials, or using a generic set of credentials for all authorized users of the portal (Fig

2.4). The heterogeneous nature of resources and their differing security policies are complicated and complex in the security schemes of a Grid Computing environment. These computing resources are hosted in differing security domains and heterogeneous platforms [6].

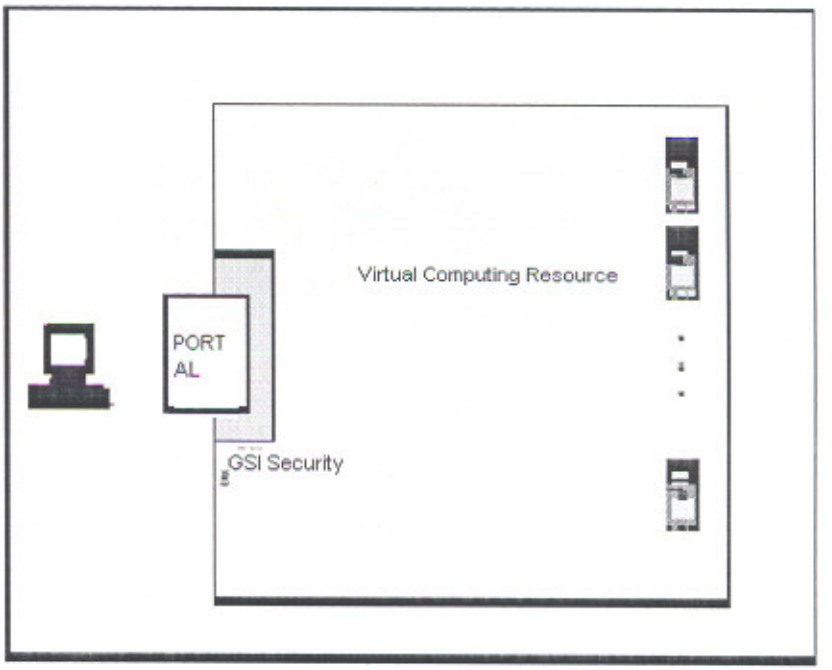


Fig 2.4: Security in Grid Environment

So, middleware solutions must address local security integration, secure identity mapping, secure access/authentication, secure federation and trust management.

Security in grid is not same as security in distributed systems. There are more issues that are to be considered when providing security in grid. Main challenge in Grid security is to establish security relationships not simply between a client and a server, but among potentially hundreds of processes that collectively span many administrative domains. Furthermore, the dynamic nature of the grid can make it impossible to establish trust relationships between sites prior to application execution. The interdomain security solutions used for grids must be able to interoperate with the diverse intradomain access control technologies inevitably encountered in individual domains [4]. To secure

computational grid, different architecture is developed including all grid security issues and requirements.

The Grid computing data exchange must be protected using secure communication channels, including SSL / TLS and in combination with secure message exchange mechanism such as WS-Security. WS-Security describes enhancements to SOAP messaging to provide quality of protection through message integrity, message confidentiality, and single message authentication. These mechanisms can be used to accommodate a wide variety of security models and encryption technologies [6].

The most notable security infrastructure used for securing grid is the Grid Security Infrastructure (GSI). GSI provides capabilities for single sign-on, heterogeneous platform integration and secure resource access/authentication [6].

The latest and most notable security solution is the use of WS-Security standards. This mechanism provides message-level, end-to-end security needed for complex and interoperable secure solutions. A combination of GSI and WS-Security provide secure message exchanges.

While scalability, performance and heterogeneity are desirable goals for any distributed system, the characteristics of computational grids lead to security problems that are not addressed by existing security technologies for distributed systems [4].

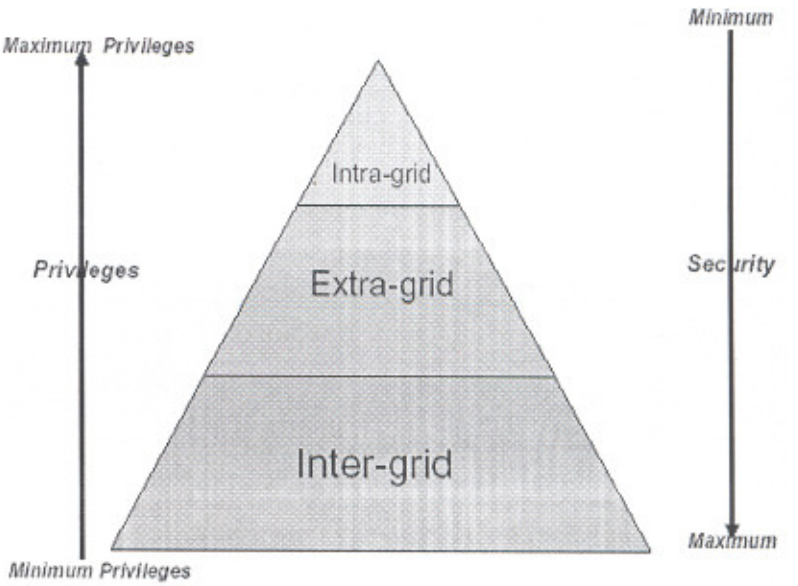


Fig 2.5: Grid Security Levels [14]

Security needs to be customized. Different kinds of systems and/or applications require different solutions. Main challenge in Grid security is to establish security relationships not simply between a client and a server, but among potentially hundreds of processes that collectively span many administrative domains.

Fig 2.5 shows the level of security required when moves from intra grid to the inter grid.

2.3.1 Security Challenges in a Grid Environment

The security challenges faced in a Grid environment can be grouped into three categories: integration with existing systems and technologies, interoperability with different "hosting environments" (e.g., J2EE servers, .NET servers, Linux systems), and trust relationships among interacting hosting environments [18].

The Integration Challenge

For both technical and pragmatic reasons, it is unreasonable to expect that a single security technology can be defined that will both address all Grid security challenges and be adopted in every hosting environment. Existing security infrastructures cannot be replaced overnight. For example, each domain in a Grid environment is likely to have one or more registries in which user accounts are maintained (e.g., LDAP directories); such registries are unlikely to be shared with other organizations or domains. Similarly, authentication mechanisms deployed in an existing environment that is reputed secure and reliable will continue to be used. Each domain typically has its own authorization infrastructure that is deployed, managed and supported. It will not typically be acceptable to replace any of these technologies in favor of a single model or mechanism. Thus, to be successful, Grid security architecture needs to step up to the challenge of integrating with existing security architectures and models across platforms and hosting environments. This means that the architecture must be *implementation agnostic*, so that it can be instantiated in terms of any existing security mechanisms (e.g., Kerberos, PKI); *extensible*, so that it can incorporate new security services as they become available; and *integratable* with existing security services.

The Interoperability Challenge

Services that traverse multiple domains and hosting environments need to be able to interact with each other, thus introducing the need for interoperability at multiple levels [18]:

• At the *protocol level*, we require mechanisms that allow domains to exchange messages. This can be achieved via SOAP/HTTP, for example.

• At the *policy level*, secure interoperability requires that each party be able to specify any policy it may wish in order to engage in a secure conversation— and that policies expressed by different parties can be made mutually comprehensible. Only then can the parties attempt to establish a secure communication channel and security context upon mutual authentication, trust relationship, and adherence to each other's policy.

• At the *identity level*, mechanisms are required for identifying a user from one domain in another domain. This requirement goes beyond the need to define trust relationships and achieve federation between security mechanisms (e.g., from Kerberos tickets to X.509 certificates). Irrespective of the authentication and authorization model, which can be group-based, role-based or other attribute-based, many models rely on the notion of an identity for reasons including authorization and accountability.

The Trust Relationship Challenge

Grid service requests can span multiple security domains. Trust relationships among these domains play an important role in the outcome of such end-to-end traversals. A service needs to make its access requirements available to interested entities, so that they can request secure access to it. Trust between end points can be *presumed*, based on topological assumptions (e.g., VPN), or *explicit*, specified as policies and enforced through exchange of some trust-forming credentials. In a Grid environment, presumed trust is rarely feasible due to the dynamic nature of VO relationships. Trust establishment may be a one-time activity per session or it may be evaluated dynamically on every request. The dynamic nature of the Grid in some cases can make it impossible to establish trust relationships among sites prior to application execution. Given that the participating domains may have different security technologies in their infrastructure

(e.g., Kerberos, PKI) it then becomes necessary to realize the required trust relationships through some form of federation among the security mechanisms.

The trust relationship problem is made more difficult in a Grid environment by the need to support the dynamic, user-controlled deployment and management of *transient services*. End users create such transient services to perform request-specific tasks, which may involve the execution of user code. For example, in a distributed data mining scenario, transient services may be created at various locations both to extract information from remote databases and to synthesize summary information. Challenges associated with user-created transient services include the following [18].

- *Identity and authorization*. It must be possible to control the authorization status (e.g., identity) under which transient services execute.
- *Policy enforcement*. Users may want to establish policies for services that they "own," to control, for example, who can access them and what actions they can perform.
- *Assurance level discovery*. A user may want to take into account the assurance level of a hosting environment when deciding where to deploy services. Thus, this information must be discoverable. Issues of concern may include virus protection, firewall usage for Internet access, and internal VPN usage.
- *Policy composition*. Security policy on instantiated services can be generated dynamically from multiple sources: not just the resource owners, but from the entity whose request created the service and the VO in which the entity's membership entitles them to do so.
- *Delegation*. Transient services may need to be able to perform actions on a user's behalf without their direct intervention. For example, a computational job running overnight might need to access data stored in a different resource. Since there may be no direct trust relationship between the VO in which the service is running and the VO in which it wishes to make a request, the service needs to be able to delegate authority to act on the user's behalf.

2.3.2 Problem with existing techniques

Security techniques in distributed system are not sufficient with Grid environment. It was reviewed that they do not meet the requirements for Security of Grid [8].

Problems With SSH

SSH provides a strong system of authentication and message protection but has no support for translation between different mechanisms or for creation of dynamic entities. SSH also has two significant drawbacks as an authentication solution for virtual organizations [13].

First, it requires that users manage their own cross-site authentication relationships, by copying public keys (or keeping track of passwords) to each site for which remote access is required. This task can become burdensome if users wish to access many remote resources; moreover, SSH denies sites control over authorization, making it difficult, for example, to deny access to a particular user without being invasive of user privacy.

A *second* disadvantage is that SSH supports only a limited set of capabilities (remote shell and file transfer). Other applications (e.g., a collaborative environment) that require authentication cannot benefit from SSH [8].

Problem with PGP

PGP works well, but only with others who also run PGP. It cannot support encrypted exchange of information with people who do not use PGP.

Problems with Kerberos

Kerberos meets many of the basic requirements for virtual organization authentication, but it presents three problems [8].

First, using Kerberos for intersite authentication would require that it also be used for *intrasite* authentication that is simply not feasible.

Second is the practical difficulty encountered in negotiating cross-realm authentication agreements: sites often feel that they surrender too much control over local

policy when they agree to accept tickets issued by other sites? In addition, the number of such agreements that must be negotiated is large.

Third, Kerberos requires the explicit involvement of site administrators to establish interdomain trust relationships or to create new entities.

2.4 GRID'S SECURITY REQUIREMENTS

Grid systems and applications may require any or all of the standard security functions, including authentication, access control, integrity, privacy, and non-repudiation. . Main focus is on the issues of authentication and access control.

The goal and purpose of Grid technologies is to support the sharing and coordinated use of diverse resources in dynamic, distributed VOs: in other words, to enable the creation, from distributed components, of virtual computing systems that are sufficiently integrated to deliver desired qualities of service. Security is one of the characteristics of an OGSA-compliant component [18]. The basic requirements of a Grid security model are that security mechanisms be *pluggable* and *discoverable* by a service requestor from a service description. This functionality then allows a service provider to choose from multiple distributed security architectures supported by multiple different vendors and to plug its preferred one(s) into the infrastructure supporting its Grid services. Grid security must be seamless from edge of network to application and data servers, and allow the federation of security mechanisms not only at intermediaries, but also on the platforms that host the services being accessed.

Need is to develop a security architecture that meet requirements and include the characteristics of the grid environment and grid applications [6]. The basic Grid security model must address the following security disciplines:

• *Authentication*. Provide plug points for multiple authentication mechanisms and the means for conveying the specific mechanism used in any given authentication operation. The authentication mechanism may be a custom authentication mechanism or an industry-standard technology. The authentication plug point must be agnostic to any specific authentication technology.

• *Delegation*. Provide facilities to allow for delegation of access rights from requestors to services, as well as to allow for delegation policies to be specified. When dealing with delegation of authority from an entity to another, care should be taken so that the authority transferred through delegation is scoped only to the task(s) intended to be performed and within a limited lifetime to minimize the misuse of delegated authority.

• *Single Logon*. Relieve an entity having successfully completed the act of authentication once from the need to participate in re-authentications upon subsequent accesses to OGSA-managed resources for some reasonable period of time. This must take into account that a request may span security domains and hence should factor in federation between authentication domains and mapping of identities. These single sign-on sessions may include accessing of resources in other domains using a service credential delegation.

• *Credential Lifespan and Renewal*. In many scenarios, a job initiated by a user may take longer than the life span of the user's initially delegated credential. In those cases, the user needs the ability to be notified prior to expiration of the credentials, or the ability to refresh those credentials such that the job can be completed.

• *Authorization*. Allow for controlling access to OGSA services based on authorization policies (i.e., who can access a service, under what conditions) attached to each service. Also allow for service requestors to specify invocation policies (i.e. who does the client trust to provide the requested service). Authorization should accommodate various access control models and implementation.

• *Privacy*. Allow both a service requester and a service provider to define and enforce privacy policies, for instance taking into account things like personally identifiable information (PII), purpose of invocation, etc. (Privacy policies may be treated as an aspect of authorization policy addressing privacy semantics such as information usage rather than plain information access.)

• *Confidentiality*. Protect the confidentiality of the underlying communication (transport) mechanism, and the confidentiality of the messages or documents that flow over the transport mechanism in an OGSA compliant infrastructure. The confidentiality requirement includes point–to–point transport as well as store-and-forward mechanisms.

•*Message Integrity*. Ensure that unauthorized changes made to messages or documents may be detected by the recipient. The use of message or document level integrity checking is determined by policy, which is tied to the offered quality of the service (QoS).

•*Policy exchange*. Allow service requestors and providers to exchange dynamically security (among other) policy information to establish a negotiated security context between them. Such policy information can contain authentication requirements, supported functionality, constraints, privacy rules etc.

•*Secure logging*. Provide all services, including security services themselves, with facilities for time-stamping and securely logging any kind of operational information or event in the course of time - securely meaning here reliably and accurately, i.e. so that such collection is neither interruptible nor alterable by adverse agents. Secure logging is the foundation for addressing requirements for notarization, non-repudiation, and auditing.

•*Assurance*. Provide means to qualify the security assurance level that can be expected of a hosting environment. This can be used to express the protection characteristics of the environment such as virus protection, firewall usage for Internet access, internal VPN usage, etc.

•*Manageability*. Provide Manageability of security functions such as identity management, policy management, security key management and other critical aspects.

•*Firewall traversal*. A major barrier to dynamic, cross-domain Grid computing today is the existence of firewalls. As noted above, firewalls provide limited value within a dynamic Grid environment. However, it is also the case that firewalls are unlikely to disappear anytime soon. Thus, the OGSA security model must take them into account and provide mechanisms for cleanly traversing them—without compromising local control of firewall policy.

•*Securing the OGSA infrastructure*. Securing the OGSA infrastructure secures the OGSO itself. The model must include securing components like Grid HandleMap, discovery service etc.

2.5 E-COMMERCE SECURITY VS. GRID SECURITY

The major differences between e-commerce security and Grid security are [13]:

Collaboration and sharing: In e-commerce, there is no concept of collaboration and resource sharing. The notion of resources in e-commerce is restricted to files and databases.

Remote program execution: Users in e-commerce cannot run programs because of the security consequences on the e-commerce company. In Grid, users can run programs on remote sites.

Trust: There is no trust relationship between the e-commerce Company and customers. This allows them to install firewall and define trusted zones: Trusted in the company and un-trusted zone that is the Internet

Authentication: Authentication in e-commerce is not a top priority as in Grid. As long as the customer can provide a valid credit card details, he can get access to services and resources on the company's site [13].

2.6 GRID SECURITY MODEL PRINCIPLES

From a security point of view, the virtualization of a service definition encompasses the security requirements for accessing that service [18]. The need arises in the virtualization of security semantics to use standardized ways of segmenting security components (e.g., authentication, access control, etc.) and to provide standardized ways of enabling the federation of multiple security mechanisms. The benefits of having a loosely-coupled, language-neutral, platform-independent way of linking and securing applications within organizations, across enterprises, and across the Internet is fundamental to the problem set addressed by the OGSA architecture. Therefore, abstracting security components as a single security model enables organizations to use their existing investments in security technologies while communicating with organizations using different technologies. As evident from the Grid security requirements, securing Grid services is a fundamental requirement behind the security model reviewed here. While providing the required security infrastructure, the environment may use the security functions and components, which may be exposed as

Grid services. Therefore, the principles underlying the Grid security model can be categorized as:

a) A security model to secure Grid services in general and
b) Security services built to provide the necessary functionality.

2.6.1 Secure Invocation of Grid Services

The Grid security architecture must ensure that OGSA services when invoked by a service requestor adhere to policy constraints as levied by the hosting environment. Such policy may include a specific type of credential, integrity and confidentiality requirements and so forth for successful invocation of the service. This architecture must also enable service requestors to dynamically select services which meet policy constraints levied by the service requestor, as a service requestor may select a service provider which best meets the requestor's policies [18].

A Grid service must be able to define or publish the Quality of Protection (QoP) it requires and the security attributes of the service Aspects of the QoP include security bindings supported by the service, the type of credential expected from the service requestor, integrity and confidentiality requirements, etc. The security attributes of the service can include information such as service identity. This enables service requestors to discover a service based on the requestor's security characteristics.

Additionally, service requestors will be able to evaluate their invocation policies based on the security attributes of the service. Note that that there may be policy restrictions on the visibility of the service's security attributes. From the service provider's point of view requests to invoke Grid services by service requestors are subject to policy checks defined by the service's access policies. For example, some policies may require that the service provider will only allow the invocation of a service after the service requestor has authenticated itself first, and provides an appropriate credential when invoking the service.

These requirements highlight the need for establishing standard mechanisms for conveying and enforcing the quality of protection, security attributes and access policies associated with services and requesters.

2.6.2 Grid Security Services

The suite of security services and the primitives, which are required as building blocks, provide a rich set of services to application logic hosted in an OGSA environment [18]. This does not imply that application components need to be aware of security semantics as such. Rather, an OGSA implementation, which uses a hosting environment, may govern via authorization policy whether a given service can be instantiated. Depending on the application or hosting environment, a generic set of security primitives provide a robust foundation for applications and so forth.

Secure invocation of Grid services brings out the need for a security model that reflects the security components that need to be identified and defined based on the Grid security requirements. Security model is based on the requirements and identifies a set of security components that need to be defined and formalized as specifications. Some of the security components can be realized as Grid security.

2.7 GRID SECURITY MODEL

Industry efforts have rallied around Web services (WS) as an emerging architecture which has the ability to deliver integrated, interoperable solutions. Ensuring the integrity, confidentiality and security of Web services through the application of a comprehensive security model is critical, both for organizations and their customers – which is the fundamental starting point for constructing virtual organizations. The secure interoperability between virtual organizations demands interoperable solutions using heterogeneous systems [18]. For instance, the secure messaging model proposed by the Web Services Security roadmap document supports both public key infrastructure (PKI) and Kerberos mechanisms as particular embodiments of a more-general facility and can be extended to support additional security mechanisms.

The security of a Grid environment must take into account the security of various aspects involved in a Grid service invocation. This is depicted in the Fig 2.6. A web service can be accessed over a variety of protocols and message formats it supports, as defined by its bindings. Given that bindings deal with protocol and message formats, they

should provide support for quality of service, including such security functions as confidentiality, integrity, and authentication. Each participating end point can express the policy it wishes to see applied when engaging in a secure conversation with another end point. Policies can specify supported authentication mechanisms, required integrity and confidentiality; trust policies, privacy policies, and other security constraints. Given the dynamic nature of Grid service invocations, end points will often discover the policies of a target service and establish trust relationships with it dynamically.

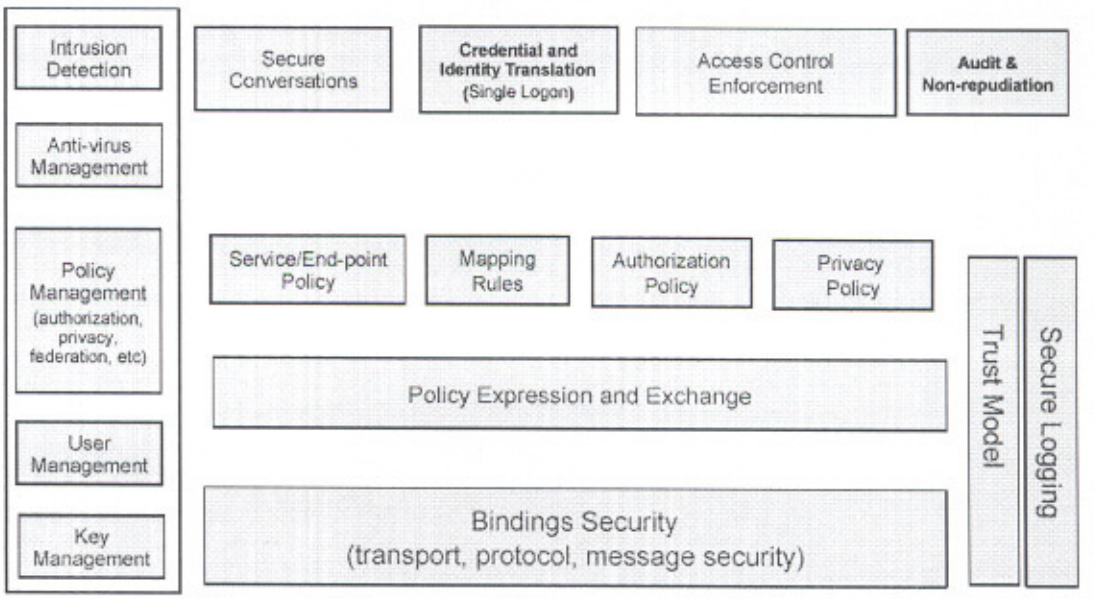


Fig 2.6: Components of grid security model [18]

Once a service requestor and a service provider have determined each other's policies, they can establish a secure channel over which subsequent operations can be invoked. Such a channel should enforce various qualities of service including identification, confidentiality, and integrity. The security model must provide a mechanism by which authentication credentials from the service requestor's domain can be translated into the service provider's domain and vice versa. This translation is required in order for both ends to evaluate their mutual access policies based on the established credentials and the quality of the established channel.

2.7.1 Scenario Involving Intermediaries

When an end user requests a service from the server, then request is traversed through intermediaries. Assume that a user, wishes to invoke a Grid service, e.g. at an intermediary node which will eventually result in accessing some resource at some N-step remote service provider (Fig 2.6). The user can obtain a credential by authenticating to an authentication server, local to his domain, and present that credential as part of the service request [16]. When the request gets routed through a gateway, the gateway may consult an attribute server to obtain the user's privilege attributes and rights and send the assertions with the request. Such a request may be routed through some intermediary, which can convert the assertions into a form understood by the target domain (e.g. based on WSFederation).

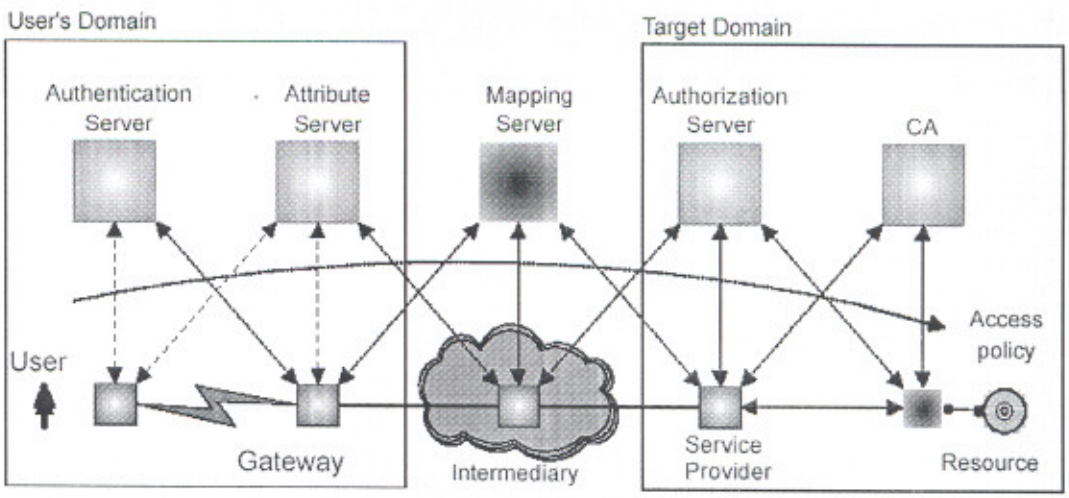


Fig 2.7: Service requests that traverse through intermediaries [18]

For example, the intermediary may convert the authentication credential (e.g., Kerberos ticket) into a credential form that target domain can work with (e.g., X509 certificate). Additionally, intermediary can honor a set of policies when forwarding the request, including the mapping rules and delegation policies.

When the request is received, the target can validate the certificate. Upon successful validation, it can derive an identity based on the certificate and make authorization decisions using the locally defined authorization policies [18]. This

example illustrates the value of looking at the problem of starting from an authentication credential in one domain and going through delegation and mapping to a target server that makes authorization decision based on credential mapping performed by an intermediary. This roadmap suggests an approach for a set of standardized OGSA compliant services which addresses the needs of credential exchange and propagation with WS-Security, policy services which may be used by the work hosting environment based on WS-Policy, the needs of validating and mapping identities with WSFederation, and lastly secure session and context establishment with WSSecureConversation [18]. These scenarios did not address specific bindings. This roadmap suggests a binding and implementation agnostic architecture that fosters the interoperability between heterogeneous systems, addressing the emerging needs of the Grid environment reflected in a business scenario.

Chapter 3

Secure Grid Service using GT3

Security is very important in collaborative environments like grid. It is vital that the security mechanisms are correctly implemented during application development and accurately configured during deployment. This section explains the security terminology and mechanisms available with the Globus Toolkit 3.0.

3.1 SECURITY IN THE GLOBUS TOOLKIT 3

Security in Globus Toolkit version 3 (GT3) is based on a Public Key Infrastructure (PKI). The PKI system is a trust hierarchy system where participating entities are identified and verified by using certificates they possess. Public keys usually conform to the X.509 standard for certificates. For a Globus-based secure grid, all participating entities (services and clients) have an identity associated with them in the form of an X.509 certificate. One may already possess an X.509 certificate used by organization or a Globus certificate used with earlier versions of the toolkit. Many commercial products are available to set up a Certificate Authority (CA) for grid. However, the easiest may be to use the Simple CA package distributed by the Globus team to set up a CA to issue certificates for use with the Globus grid. The function of authorization or access control is to control which users have access to which parts of the system. Authorization in GT3 is based on a simple access control list placed in a flat file called *gridmap*. The Globus Toolkit 2.0 used the gridmap file to map a user to a user ID on a remote resource during job submission. GT3 extends this idea by applying access control policies to the factories and services. Gridmap files are associated with services and factories, a technique that restricts who can access the functions provided. GT3 provides mechanisms for both transport layer security and message level security. The transport layer security relies on securing the transport mechanism itself [19].

To make grid services inherently secure, there is a transport dependency. Message level security relies on securing each message individually. This mechanism allows the flexibility that it can be sent over any transport layer. The Grid Security Infrastructure (GSI) is based on the public key infrastructure. Secure Socket Layer (SSL) serves as the foundation for the security services in the Globus Toolkit. In GT3, the transport layer security is based on a GSI-enabled HTTP protocol. Message level security is based on emerging technologies and standards such as WS-Security, XML Encryption, and XML Signature standards. Message level security is more appreciating, thus Globus Toolkit recommends that applications must use message level security since the GSI based transport layer security may be phased out in future releases.

3.2 GRID SECURITY INFRASTRUCTURE (GSI)

The GT3 allows us to overcome the security challenges posed by grid applications through the *Grid Security Infrastructure* (or GSI), which offers the following three features:

- Complete public-key system
- Mutual authentication through digital certificates
- Credential delegation and single sign-on

GSI is composed of a set of command-line tools to manage certificates, and a set of Java classes to easily integrate security into grid services. It is based on standard technologies, such as TLS and secure Web Services specifications (XML-Signature, XML-Encryption, etc.) [20].

3.2.1 Complete public-key system

The GSI is based on public-key cryptography, and therefore can be configured to guarantee privacy, integrity, and authentication (strong authentication is provided in conjunction with certificates). However, not all communications need to have those three features all at once. In general, a GSI secure conversation must at least be authenticated.

Integrity is usually desirable, but can be disabled. Encryption can also be activated to ensure privacy. These features are easy to use by simply adding few lines in the client

3.2.2 Mutual authentication through digital certificates

The GSI uses X.509 certificates to guarantee a strong authentication. *Mutual* authentication simply means that in GSI, both parts of a secure conversation must be authenticated. In other words, when A wants to communicate with B, A must trust B and B must trust A. Remember that 'trust' (in this context) means that A must have the certificate of the CA that signed B's certificate, and vice versa. Otherwise, A won't trust B (and vice versa). This is achieved by setting by SimpleCA and get digital certificate.

3.2.3 Credential delegation and single sign-on (Proxy Certificates)

Credential delegation and single sign-on are one of the most interesting features of GSI, and this is achieved by *proxy certificates*. Proxy Certificates solves the problem like [20].

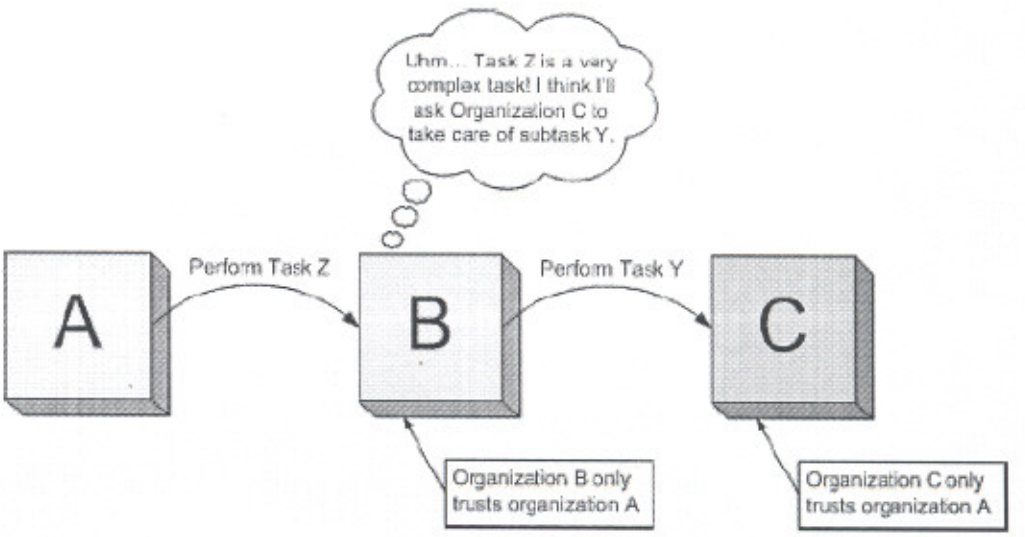


Fig 3.1 The Problem [20]

Organization A asks Organization B to perform a task. Since B trusts A, it accepts to perform the task. Third organization is organization C. In this case, B will ask C to perform subtask Y but C only trusts A. Organization C has two options:

• Turn down B's request. As, C doesn't trust B.

• Accept B's request. The 'original' requestor is A so, although C is answering a request from B, it will actually be carrying out a job for A.

In this situation, it seems logical that C should *accept* B's request. However, C has to know that B's request is performed on behalf of A:

This is not a very secure solution, since *anyone* could claim to be acting on A's behalf! One possible solution would be for C to contact A every time it receives a request on A's behalf. However, this could be a bit of a nuisance. Imagine that task Z is composed of 20 different subtasks, and that each subtask is dispatched to a different organization by B.

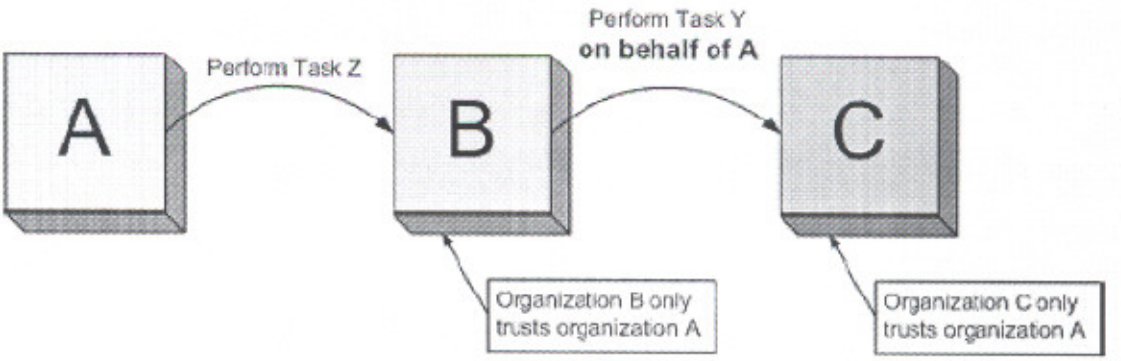


Fig 3.2: The Problem

Organization A would be flooded with messages saying "B just asked to perform a task on A's behalf... do confirm that this is correct?" In response, A would have to mutually authenticate itself with all those organizations and give a confirmation.

A more elegant solution would be to somehow make Organization C believe that Organization B *is* Organization A. In other words, it would interesting to find a legitimate way for B to demonstrate that it is, acting on A's behalf. One way of doing this would be for A to 'lend' it's public and private key pair to B. The private key has to remain *secret*, and sending it to another is a *big* breach in security.

The solution is **proxy certificates [20]**: Fig 3.3 is proxy certificate which assures that B is acting on A's behalf:

Webster's Dictionary defines 'proxy' as "The instrument by which a person is empowered to transact the affairs of another". In fact, it's very similar to the X.509 digital certificates, except that it's not signed by a Certificate Authority; it's signed by an end user. We can be sure that the certificate is authentic by checking its signature

A proxy certificate has a private-public key pair generated specifically for the proxy certificate. This private-public key pair is mutually agreed upon by both parties (in this case, A and B), and Organization A will only allow the holder of *that* private-public key pair to act on its behalf (in this case, B).

The Proxy certificate's lifetime is usually very limited (for example, to 24 hours). This means that, if the proxy certificate is compromised, the attacker won't be able to make much use of it. Furthermore, proxy certificates extend ordinary X.509 certificates with extra security features to limit their functionality even more (for example, by specifying that a proxy certificate can only be used for certain tasks). More correct representation of a proxy certificate would be the following:

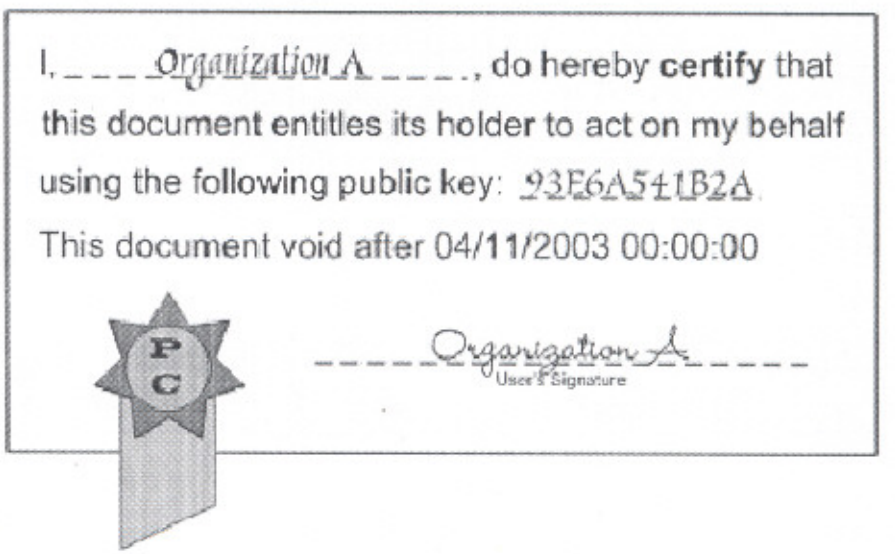


Fig 3.3: The Complete Proxy Certificate

Proxy Certificates achieves Delegation and single sign-on

A proxy certificate allows a user to act on another user's behalf. This is more properly called *credential delegation*, since proxy certificates allow a user to effectively

delegate a set of credentials (the user's identity) to another user. This solves the problem originally posed, since B could use a proxy certificate (signed by A) to prove that it is acting on A's behalf. Organization C would then accept B's request [20].

By using proxy certificates, desirable feature named *single sign-on* is also met. Without proxy certificates, Organization A would have to mutually authenticate itself with all the organizations that receive requests 'on behalf of A'. This mean that the user in Organization A with permission to read the private key would have to access the key each time a mutual authentication is needed. Since private keys are usually protected by a password, this means that the user would have to *sign on* (provide the password) to access the key and perform mutual authentication. Using proxy certificates, the user only has to sign in *once* to create the proxy certificate. The proxy certificate is then used for all subsequent authentications.

3.3 SIMPLE GRID SERVICE

To build GT3 grid service, there is requirement of specific software and grid environment. On Grid environment, the following are the five steps to build simple grid service [20].

1. **Define the service's interface**. This is done with *GWSDL*
2. **Implement the service**. This is done with *Java*
3. **Define the deployment parameters.** This is done with *WSDD*
4. **Compile everything and generate GAR file.** This is done with *Ant*
5. **Deploy service.** This is also done with *Ant*

Step 1: Defining the interface in GWSDL

The first step in writing a Grid Service is to define the *service interface*. There is a need to specify what a service is going to provide to the outer world. At this point, no one is concerned with the inner workings of that service (what algorithms it uses, what databases it will access, etc.) Only need to know what *operations* will be available to the users. The service interface is usually called the *port type* (written as *portType*). XML

language is used to specify what operations a web service offers: the Web Service Description Language (WSDL).

In this step, simply write a description of Grid Service using WSDL. In GT3 we have two options:

- Writing the WSDL directly.
- Generating WSDL from a Java interface.

Actually, grid service is written not in WSDL, but Grid WSDL (or GWSDL), which is an extension of WSDL used in the OGSI specification. This 'extended' WSDL supports all the features of grid service.

Step 2: Implementing the service in Java

After defining the service interface ("what the service does"), the next step is implementing that interface. The implementation is "how the service does what it says it does". The implementation of a Grid Service is simply a Java class, which besides implementing the operations described in the GWSDL file, has to meet certain requirements. This class can furthermore include additional private methods which won't be available through the Grid Service interface, but which our service can use internally. The name of these java files should have suffix Impl as it makes easier for layman to understand.

Step 3: Configuring the deployment in WSDD

Up to this point, the two most important parts has been done for a Grid Service: the service interface (GWSDL) and the service implementation (Java). Now, need to put all these pieces together, and make them available through a Grid Services-enabled web server! This step is called the *deployment* of the Grid Service.

One of the key components of the deployment phase is a file called the *deployment descriptor*. It's the file that tells the web server how it should publish our Grid Service (for example, telling it what the service's Grid Service Handle (GSH) will be). The deployment descriptor is written in WSDD format (Web Service Deployment Descriptor).

Step 4: Create a GAR file with Ant

At this point we have (1) a service interface in GWSDL, (2) a service implementation in Java, and (3) a deployment descriptor telling the grid services container how to present (1) and (2) to the outer world. However, all this is a bunch of loose files. Now, we are supposed to place this in a grid services container.

All these files are combined to generate a *Grid Archive*, or *GAR file* and then deploy it in the grid service container, to make it available to be used by grid users.

However, creating a GAR file is a pretty complex task which involves the following:

- Converting the GWSDL into WSDL
- Creating the stub classes from the WSDL
- Compiling the stubs classes
- Compiling the service implementation
- Organize all the files into a very specific directory structure

These steps are not followed as this all can be done with the tool named ANT.

Ant

Ant, an Apache Software Foundation project, is a Java *build tool*. In concept, it is very similar to the classic UNIX makes command. It allows programmers to forget about the individual steps involved in obtaining an executable from the source files, which will be taken care of by Ant. Each project is different, so the individual steps are described in a *build file*. This build file directs Ant on what it should compile, how it should compile it, and in what order.

This simplifies the whole process considerably. With Ant, only worry is about to write the service interface, the service implementation, and the deployment descriptor. Ant takes care of the rest. Ant generates the GAR directly from the three source files. Internally, it is carrying out all the steps listed earlier, sparing us the cumbersome task of doing them ourselves. In a GT3 project, Ant uses two sets of build files: a couple of build files which are a part of GT3, and a build file we'll have to write on our own. The GT3 build files cover all the important steps (generating the WSDL code, generating the stubs). Our build file essentially has all the unique parameters of our Grid Service, and a bunch of calls to the GT3 build files.

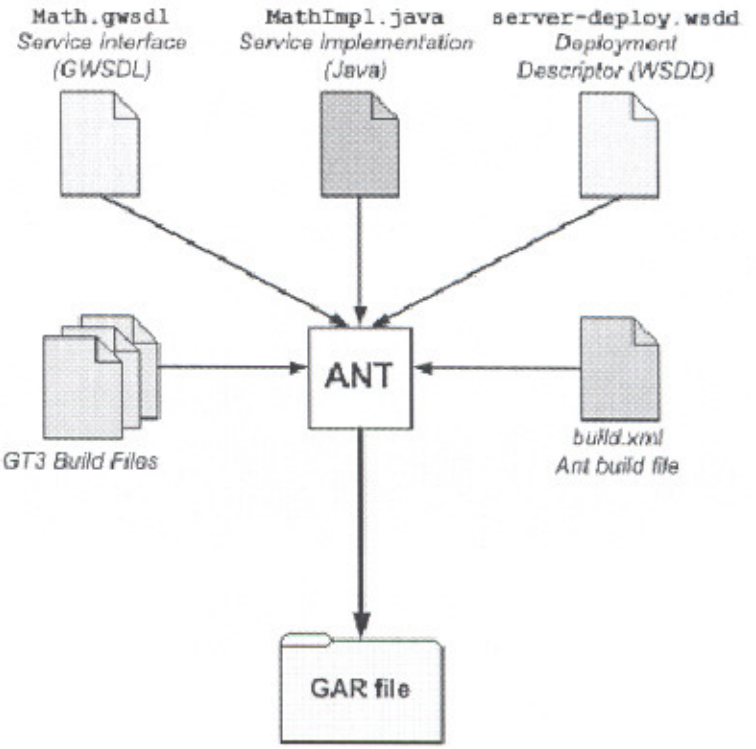


Fig 3.4: Creating Gar file using Ant [20]

A GAR file is created using the following command, run it from the folder where service is lying.

```
ant -Djava.interface=true -Dpackage=<path of package> -
Dinterface.name=<interface name> -Dpackage.dir=<package directory path>
-Dservices.namespace=<service's path>
```

Step 5: Deploy the service into a grid service container

The GAR file contains all the files and information the web server needs to deploy the Grid Service. Deployment is also done with the Ant tool, which unpacks the GAR file and copies the files within (WSDL, compiled stubs, compiled implementation, WSDD) into key locations in the GT3 directory tree. It also reads our deployment descriptor and configures the web server to take our new Grid Service into account.

This deployment command must be run from the root of GT3 installation. Deploy the file from %GLOBUS_LOCATION%. Furthermore, need to run it with a user that has write permission in that directory.

```
ant deploy -Dgar.name=<full path of GAR file>
```

SIMPLE CLIENT

To test the deployed service from container a simple client application can be developed. A client is developed in java, which will call the service deployed (available in the globus container) and invoke any of the method defined in the service. Each client will have one fix argument i.e. Grid Service Handle (GSH) and more it can have, according to the type of service. Simply compile and run the java client.

3.4 SETTING UP GSI FOR BUILDING SECURE GRID SERVICE

3.4.1 Building SimpleCA

SimpleCA is a simple utility to manage web x.509 certificates, both server certificates (IIS, apache, ...) and client certificates (IE, NS, ...). SimpleCA is available under GPL. SimpleCA implements a simple X.509 CA with a nice GUI. With this, ssl or personal certificates can be managed. No configuration option is available: all is based on sensible defaults (1024bit keys, etc...). The basic features are present (generate certificate, revoke certificate, export pkcs12, revocation lists).

SimpleCA allows to do following.

- Generate a server private key and certificate request, and generate the corresponding certificate.
- Generate a client private key and certificate request, generate the corresponding certificate, and export this certificate and private key as PKCS12 format (.p12) for import into a browser.
- Revoke a certificate.
- Publish a CRL of revoked certicates.
- Untested, but one should be able to sign a certificate request generated via other means.

The SimpleCA menu has following options

- **Server Certificates**
  - **New Server Certificate Request**

    Generates a new server certificate request. Here, a dialog asks for essential information about the server which wants a certificate. All fields marked with an asterisk (mandatory fields) need to be filled out. After that, a filename is asked under which to save the certificate. The default is the server common name in subfolder certificates, with .csr extension (for example certificates/www.mydomain.com.csr). When saved, two files will be generated: a certificate signing request (file with extension .csr) and a private key file (with extension .key). The private key is not protected by a password.

  - **Sign Server Certificate Request**

    This will generate a server certificate from an existing certificate signing request. You are asked to select a file (a certificate request) you want to have a certificate for. Then some certificate information (mainly the subject, and requested use) is shown for verification. After that, a certificate is generated having same file name but .csr extension replace by .crt. Also the certificate is registered as a valid certificate in the database. By default, the certificate is valid for an year.

- **Client Certificates**
  - **New Client Certificate Request**

    Generates a new client certificate request. The procedure is analog to a server certificate request; the main difference is that only a common name and an email address are required.

  - **Sign Client Certificate Request**

This will generate a client certificate from an existing certificate signing request. The procedure is analog to a server certificate request.

- **Export PKCS12 format**

  This will transform a client certificate into PKCS12 export format. You are requested to select a certificate file. Then you will have to provide friendly names for the certificate and for the CA, and a password to protect the PKCS12 file. This password is required when importing a PKCS12 (.p12) file into a browser.

3.4.2 Setting up SimpleCA

Now that SimpleCA has been built, we need to run a post-install script to set it up. This is done with the command:

```
gpt-postinstall
```

After installation, it will ask subject name. Use the following subject name:

```
cn=Globus SimpleCA, ou=Tietgrid, o=Globus
```

Then just run the simple CA, using the following command

```
grid-default-ca
```

3.4.3 Requesting and Signing certificate with SimpleCA

From the user account, run the following command:

```
grid-cert-request
```

Signing the certificate with SimpleCA includes the following procedures:

1. User A creates a certificate request.
2. User A sends the certificate request to a CA via email (e.g. ca-admin@foobar.com)
3. The CA's administrator receives the request, reviews it, and decides if it will be approved. If the request is approved, the CA administrator signs the request using the CA's private key, and sends the certificate to User A via email (e.g. user-a@somecompany.com)
4. User A receives the certificate and installs it in the $HOME/.globus directory.

3.4.4 Creating proxy certificates

An operation important during the security is creating proxy certificates. First set up a couple of environment variables. Now, run the following command:

```
grid-proxy-init
```

You should see the following output:

```
Your identity: /O=Globus/OU=tietgrid/CN=Raj
Enter GRID pass phrase for this identity:
```

Enter the password which protects user's private key. Then, following is seen:

```
Creating proxy ..................................................... Done
Your proxy is valid until: Sun Feb 15 22:55:45 2004
```

The date should be 12 hours more than the current time. If curious about disposition, take a look at some of the proxy certificate's contents by running the following command:

```
grid-proxy-info
```

This command will show, among other things, the path where the proxy certificate has been created. One can take an even closer look at the contents of the proxy certificate by running the following:

grid-cert-info -file <PATH_TO_PROXY_CERT>

3.5 SECURE GRID SERVICE

The GT3 toolkit provides mechanisms to configure services to use authentication and service level authorization. The API available with the toolkit helps programmers integrate other security mechanisms or finer-grained access control [20]. To develop the secure grid service, it follows the same steps as developing the simple service.

Step 1: The service interface

The interface for secure grid service is just like ordinary interface. There is no need to create a new GWSDL file since adding security to a service doesn't affect the interface description (i.e. the GWSDL file).

Step 2: The service implementation

To implement security in a service, OperationProvider must be used. OperationProvider is to separate or distribute the functionality of a service into several

classes rather than one class. For example, a service with basic functionality can add one more private method logSecurityInfo which will log certain security information.

The implementation of the public methods(methods performing basic functionality) is very simple. The logSecurityInfo method is called in other functionality's methods like add:

```
public void add(int a) throws RemoteException
{
logSecurityInfo("add");
value = value + a;
}
```

Finally, the logSecurityInfo includes some security information to the container's log. LogSecurityInfo looks like:

```
private void logSecurityInfo(String methodName)
{
Subject subject;
logger.info("SECURITY INFO FOR METHOD '" + methodName + "'");
// Print out the caller
String identity = SecurityManager.getManager().getCaller();
logger.info("The caller is:" + identity);
// Print out the caller's subject
subject = JaasSubject.getCurrentSubject();
logger.info("INVOCATION SUBJECT");
logger.info(subject==null?"NULL":subject.toString());
// Print out service subject
logger.info("SERVICE SUBJECT");
subject = SecurityManager.getManager().getServiceSubject(base);
logger.info(subject==null?"NULL":subject.toString());
// Print out system subject
logger.info("SYSTEM SUBJECT");
try{
subject = SecurityManager.getManager().getSystemSubject();
logger.info(subject==null?"NULL":subject.toString());
}catch(Exception e)
{
logger.warn("Unable to obtain service subject");
}
}
```

Enabling security in a grid service doesn't affect the server-side code at all.

Step 3: Deployment descriptor parameters

To tell the container that our service is going to be a secure service, we need to add two parameters to the deployment descriptor:

• The security configuration file (securityConfig parameter)

• Authorization method to use (authorization parameter)

The securityConfig parameter

This parameter tells the container where *security configuration file* is located. This is an XML file that includes configuration details related to security, such as what level of security is required when invoking certain methods. Here we are using a default security configuration file included with GT3. This default file specifies that all the methods must be accessed using a secure conversation. To tell the container to use the default configuration file for our service, we'll need to add the following lines to the deployment descriptor:

```
<parameter name="securityConfig"
value="org/globus/ogsa/impl/security/descriptor/gsi-security-
config.xml"/>
```

The authorization parameter

These parameters allow us to specify the type of server-side GSI authorization to use in this service. Different type of authorization exists like: no authorization or authority is checked from grid file.

To use no authorization, we'll need to add the following lines to the deployment descriptor:

```
<parameter name="authorization" value="none"/>
```

Another parameter is **gridmap**, Gridmaps are one of the forms of server-side GSI authorization. They allow us to control access to our grid services, and also play important roles in higher level services. A *gridmap* is basically an ACL (Access Control List) that allows us to specify what users have access to a service. The following line must also be included

```
<parameter name="gridmap" value="/home/tietgrid/gridmap"/>
```

So, the deployment descriptor contains following more lines to secure the service

```
<parameter name="securityConfig"
value="org/globus/ogsa/impl/security/descriptor/gsi-security-
config.xml"/>
<parameter name="authorization" value="gridmap"/>
<parameter name="gridmap" value="/home/tietgrid/gridmap"/>
```

The securityConfig specifies the deployment descriptor for the security properties. This parameter helps to provide finer-grained control over the security properties of a service. Set the securityConfig parameter correctly for authentication to take place. Here we will use the generic gsi-security-config.xml supplied with GT3 to provide a GSI secure conversation authentication mechanism.

The authorization parameter specifies the authorization mechanism to use. The options are none, selfcode, or gridmap. When gridmap is used for authorization, need to set the parameter gridmap to point to the gridmap file. If not initialize the authorization when authentication is performed, then by default self authorization is performed.

Step 4: Create Gar file using Ant like simple grid service.

Step 5: Deploy the service in container.

A SECURE CLIENT

The client used to invoke the secure service will be almost identical to clients seen in the previous part. In fact, to configure our client for a basic secure invocation we only need to add two lines!

```
((Stub)math)._setProperty(Constants.GSI_SEC_CONV,Constants.ENCRYPTION);
((Stub)math)._setProperty(Constants.AUTHORIZATION,NoAuthorization.getInstance()
);
```

These two lines configure the stub class (a GridServicePortType object) to use security. More specifically:

• We're telling the stub to use full-blown encryption. GSI allows us to choose between different levels of security, and encryption is just one of them. For example, we could have chosen to guarantee message integrity using a digital signature (without encrypting the whole message).

• We're telling the stub to use no client-side authorization. Remember that there is a difference in GSI between client-side and server-side authorization

As mentioned before, besides those two lines, the rest of the client is identical to the previous ones, with the basic functionality. The only difference is that we'll be putting the

functionality like add, subtract calls inside try...catch blocks to observe how certain exceptions are raised in certain circumstances.

So secure client must include the basic functionality and the following lines of code:

```
...
OGSIServiceGridLocator factoryService = new
OGSIServiceGridLocator();
Factory factory = factoryService.getFactoryPort(new
HandleType(handle));
...
// securing the factory stub
((Stub) factory)._setProperty(Constants.GSI_SEC_CONV,
Constants.ENCRYPTION);
((Stub) factory)._setProperty(Constants.AUTHORIZATION,
NoAuthorization.getInstance());
GridServiceFactory gridFactory = new GridServiceFactory(factory);
LocatorType locator = gridFactory.createService(null, id);
...
```

The secure grid service is deployed in the grid container; secure service will be used by the secure client.

Let's compile the client:

```
javac -classpath ./build/classes/:$CLASSPATH full path/filename .java
```

Before running any of the client applications, need to create a proxy certificate for user account. The default behavior in the client-side is to use a proxy certificate for authentication, so we need to create one first. Now, run the client:

```
Java -classpath ./build/classes/:$CLASSPATH \
<Class filename with proper path> <full service link from container>
<command line parameter, if any>
```

If all goes well functionality of application is seen on the client side. And the following on the server side:

```
INFO: SECURITY INFO FOR METHOD 'add'
INFO: The caller is:/O=Globus/OU=GT3 /CN=raj
INFO: INVOCATION SUBJECT
INFO: Subject:
Principal: /O=Globus/OU=GT3 /CN=GLOBUS 3 Administrator
Private credential: org.globus.gsi.gssapi.GlobusGSSCredentialImpl@ae1393
INFO: SERVICE SUBJECT
INFO: NULL
INFO: SYSTEM SUBJECT
INFO: Subject:
```

```
Principal: /O=Globus/OU=GT3 /CN= GLOBUS 3 Administrator
Private credential: org.globus.gsi.gssapi.GlobusGSSCredentialImpl@ae1393
```

Here, the caller's subject is the one in my account's certificate while the invocation and system subject is the subject of the certificate belonging to the globus account.

If authorization is been used with the grid map file and the respective client is not been authorized to use to requested service. After compiling and running the secure client, it may give an error if the user running the client is not in the gridmap file; it has been denied access to the server. The error will be

```
org.globus.ogsa.impl.security.authorization.AuthorizationException:
Gridmap authorization failed:
peer "/O=Globus/OU=GT3 /CN=raj" not in gridmap file.
```

Chapter - 4

Implementation of Grid Service

4.1 GT3 INSTALLATION (Setup on Windows 2000)

To setup GT3 on windows 2000, numbers of software tool are required. Following are the steps to setup grid environment [21].

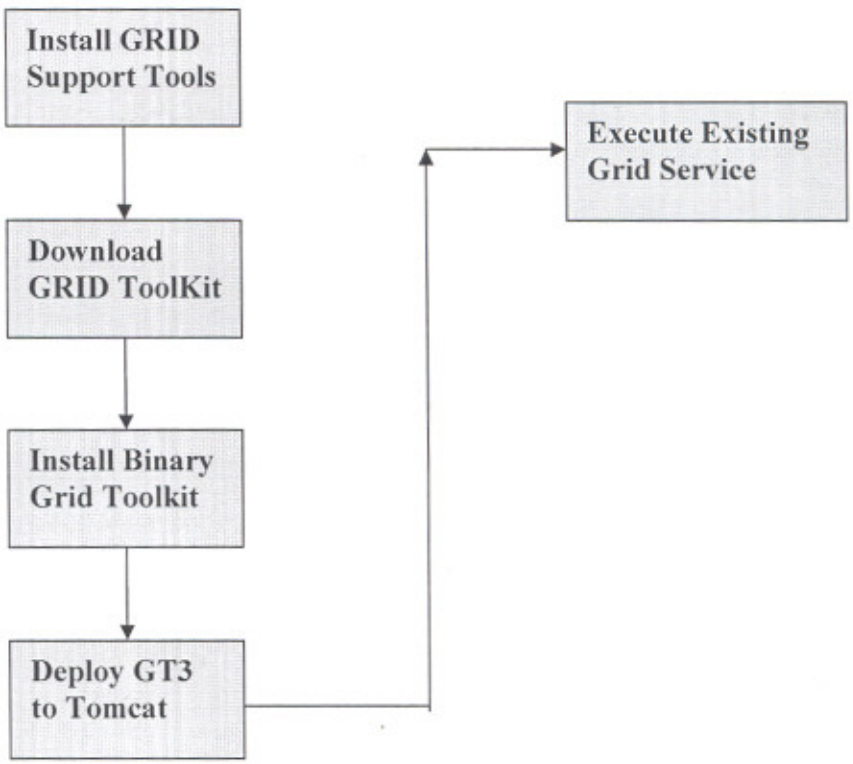


Fig 4.1: GT3 Installation

Before installing any tool for GT3 grid installation, first step is to setup GLOBUS Directory Structure and Environment Variables .

Tasks:

1. Create Environment Variables
2. Update PATH environment variable

3. Create GLOBUS Sandbox Directory Structure

1. Set Environment Variables
    a. GLOBUS_ROOT = c:\grid
    b. ANT_HOME = %GLOBUS_ROOT%\apache-ant-1.5.4
    c. JAVA_HOME=c:\j2sdk1.4.2_04
    d. GLOBUS_LOCATION=%GLOBUS_ROOT%\ogsa-3.0.2
    e. CATALINA_HOME=%GLOBUS_ROOT%\Tomcat4.1
2. Add to PATH Environment Variable
    a. %JAVA_HOME%\bin
    b. %ANT_HOME%\bin
    c. %GLOBUS_LOCATION%\bin
    d. %GLOBUS_ROOT\bin
3. Create GLOBUS Sandbox Directory Structure
    a. Create GLOBUS_ROOT directory
    b. Create GLOBUS_ROOT\download directory
    c. Create GLOBUS_ROOT\bin directory

Installing Grid Support tools

The tools required for grid are listed below and installed in the following sequence.

1) Java JSDK 1.4.2:

Tasks:

- java1.4.2_04 is downloaded and installed

2) Apache Ant 1.5.4

Tasks:

- ant1.5.4 is downloaded and installed into %ANT_HOME%

3) Junit 3.8 Test Driver

Tasks:

- Junit is installed into %GLOBUS_LOCATION%
- junit.jar is copied to %ANT_HOME%\lib

4) **Apache Tomcat**

Tasks:

- Tomcat is downloaded and installed to %CATALINA_HOME%
- GLOBUS mime types is added to tomcat
- WebDAV is activated

5) **Python**

Tasks:

- Python Self Extracting Installer is downloaded
- Python is installed to %GLOBUS_ROOT%\Python

6) **Python Packages / Modules**

Tasks:

- The Following Packages are downloaded to %GLOBUS_HOME%\download
    - Windows Installer package ZSI-1.4.1
    - Windows Installer 4Suite-1.0a3
    - Windows Installer Twisted Network Library
    - Windows Installer wx GUI package
    - Windows Installer BOA constructor
    - fpconst.zip Floating Point Package
    - Soapy Simple SOAP package
- Python Packages are then installed in %GLOBUS_HOME%\download

7) **HSQL Database**

Tasks:

- HSQL is downloaded
- Tomcat is installed to %GLOBUS_ROOT%

Globus Grid GT3 Downloaded

Tasks:

- Globus Grid Toolkit 3 is downloaded from www.globus.org

- Globus Grid Toolkit 3 is then installed.

Globus GT3 Installation

Tasks:

- Untar binary Globus GT3 Toolkit
- Grid Toolkit is setup

Deploy to Tomcat

Tasks:

- Ant is used to Deploy Globus GT3 to Tomcat

```
cd %GLOBUS_LOCATION%
ant -Dtomcat.dir=%CATALINA_HOME% deployTomcat
```

- Stop Tomcat
- Start Tomcat

Grid Service Execution

To have GUI to run grid service, there are two ways:

1)

- open Command Window (start->programs->accessories->Command Prompt)
- cd %GLOBUS_ROOT%\ogsa-3.0.2
- ant GUI

2)

- Open 2 command window
- In one window run " globus-start-container" from grid directory
- In other window run "globus-service-browser" from ogsa-3.0.2/bin directory.

After setting the grid environment, I have developed the simple grid service and then secure grid service.

4.2 SIMPLE BasicCalculatorService

To secure typical and all grid services, we must be able to secure the simple or basic grid service. Firstly, I have tried to secure simple grid service. For that a simple

Grid Service is been created and after that security issues are being applied to the service to make it more secure.

This service is made available to the grid users. The functionality of this service is to add, subtract, multiply and divide the number to the given stored digit in grid (initially it is zero), means on command line, a number is to be passed to add/subtract/multiply/divide to the internal number.

For developing this service, the same five steps are followed as defined in last chapter. A proper directory structure is being used to develop the grid service. Following structure is used for my grid service.

c:\ → grid → ogsa-3.0.2 → tietgrid → counter → core → factory → client
                                                              ↘ impl

Step 1: Defining the interface in GWSDL

The gwsdl file created for the GridService is CounterService.gwsdl and contains code

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
targetNamespace="http://factory.core.counter/Counter"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:gridservicesoapbinding="http://www.gridforum.org/namespaces/200
3/03/OGSI/bindings"  xmlns:impl="http://factory.core.counter/Counter"
xmlns:intf="http://factory.core.counter/Counter"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"><wsdl:import
location="../../ogsi/ogsi_bindings.wsdl"
namespace="http://www.gridforum.org/namespaces/2003/03/OGSI/bindings"
/>
 <wsdl:types>
  <schema       targetNamespace="http://factory.core.counter/Counter"
xmlns="http://www.w3.org/2001/XMLSchema">
   <element name="getValue">
    <complexType/>
   </element>
   <element name="getValueResponse">
    <complexType>
     <sequence>
      <element name="getValueReturn" type="xsd:int"/>
     </sequence>
    </complexType>
```

```
    </element>
    <element name="add">
     <complexType>
      <sequence>
       <element name="in0" type="xsd:int"/>
      </sequence>
     </complexType>
    </element>
    <element name="addResponse">
     <complexType/>
    </element>
::::
::::
::::
::::
</wsdl:definitions>
```

Step 2: Implementing the service in Java

This java file contains all the functionality of the service and is contained in the impl directory. The name of the file is CounterImpl.java and its contents are

```
package counter.core.factory.impl;

import org.globus.ogsa.impl.ogsi.GridServiceImpl;
import counter.core.factory.Counter.CounterPortType;
import java.rmi.RemoteException;

public class CounterImpl extends GridServiceImpl implements
CounterPortType
{
      private int value = 0;

      public CounterImpl()
      {
            super("Simple Counter Factory Service");
     }
     public void add(int a) throws RemoteException
     {
            value = value + a;
     }
     public void subtract(int a) throws RemoteException
     {
            value = value - a;
     }
     public void multiply(int a) throws RemoteException
     {
            value = value * a;
     }
     public void divide(int a) throws RemoteException
     {
            value = value / a;
```

```
    }
    public int getValue() throws RemoteException
    {
            return value;
    }
}
```

Step 3: Configuring the deployment in WSDD

The deployment file named Counter.wsdd is contained in the "factory" directory and includes the following lines of code

```
<?xml version="1.0"?>
<deployment                                    name="defaultServerConfig"
xmlns="http://xml.apache.org/axis/wsdd/"
      xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">

      <service      name="counter/core/factory/CounterFactoryService"
provider="Handler" style="wrapped">
              <parameter name="name" value="CounterService Factory"/>
              <parameter  name="instance-name"  value="CounterService
Instance"/>
              <parameter                       name="instance-schemaPath"
value="schema/counter.core.factory/Counter/CounterService.wsdl"/>
              <parameter                  name="instance-baseClassName"
value="counter.core.factory.impl.CounterImpl"/>
              <!-- Start common parameters -->
             <parameter name="allowedMethods" value="*"/>
             <parameter name="persistent" value="true"/>
             <parameter                                  name="className"
value="org.gridforum.ogsi.Factory"/>
             <parameter                              name="baseClassName"
value="org.globus.ogsa.impl.ogsi.PersistentGridServiceImpl"/>
      :::::
      :::::
      :::::
     </service>
</deployment>
```

Step 4: Creating GAR file with Ant

Gar file is being created using the Ant tool and the following command is run on the command prompt to create the gar file.

```
C:\grid\ogsa-3.0.2\tietgrid\ant        -Djava.interface=true        -
Dpackage=counter.core.factory -Dinterface.name=Counter -Dpackage.dir=
counter/core/factory/ -Dservices.namespace=factory.core.counter
```

By running this command, different directories and files are automatically created including build, schema, lib etc. And gar file is also created in build/lib/ with the name counter.core.factory.Counter.gar

Step 5: Deploying the service into a grid service container

The service's gar file is deployed into the globus container by running the following command on command prompt.

```
C:\grid\ogsa-3.0.2\ant deploy -Dgar.name =tietgrid/build/lib/
counter.core.factory.Counter.gar
```

A client is created in java to make use of this service. CounterClient.java

```
package counter.core.factory.client;

import counter.core.factory.Counter.CounterServiceGridLocator;
import counter.core.factory.Counter.CounterPortType;
import java.net.URL;

public class CounterClient
{
public static void main(String[] args)
{
try
{
// Get command-line arguments
URL GSH = new java.net.URL(args[0]);
int a = Integer.parseInt(args[1]);

// Get a reference to the CounterService instance
CounterServiceGridLocator counterServiceLocator = new
CounterServiceGridLocator();
CounterPortType counter =
counterServiceLocator.getCounterService(GSH);

// Call remote method 'add'
counter.add(a);
System.out.println("Added " + a);

// Get current value through remote method 'getValue'
int value = counter.getValue();
System.out.println("Current value: " + value);
}catch(Exception e)
{
System.out.println("ERROR!");
```

```
e.printStackTrace();
}
}
}
```

Client is Compiled

```
C:\grid\ogsa-3.0.2\tietgrid\javac -classpath
./build/classes/:%CLASSPATH%
counter/core/factory/client/CounterClient.java
```

Start the Globus Container

c:\grid\ogsa-3.0.2\globus-start-container

Create instance for the grid service

```
C:\grid\ogsa-3.0.2\tietgrid\ogsi-create-service
http://172.31.5.107:8080/ogsa/services/counter/core/factory/CounterFa
ctoryService r1
```

Run the java file

```
C:\grid\ogsa-3.0.2\tietgrid\java -classpath
./build/classes/:%CLASSPATH%
counter.core.factory.client.CounterClient
http://172.31.5.107:8080/ogsa/services/counter/core/factory/CounterFa
ctoryService/r1 5
```

4.3 SECURE BasicCalculaterService

Step 1: Interface in GWSDL

Same file created in gwsdl will be used for the secure BasicCalculatorService. As interface is not affected by providing security issues in the grid service

Step 2: The service implementation

A private method logSecurityInfo is called from the service with basic functionality. The logSecurityInfo method is called in functionality's methods like add, subtract, multiply, divide:

```
public void add(int a) throws RemoteException
{
logSecurityInfo("add");
value = value + a;
}
public void subtract(int a) throws RemoteException
{
logSecurityInfo("subtract");
value = value - a;
}
public void multiply(int a) throws RemoteException
{
logSecurityInfo("multiply");
value = value * a;
}
public void divide(int a) throws RemoteException
{
logSecurityInfo("divide");
value = value / a;
}
```

The logSecurityInfo includes security information to the container's log. LogSecurityInfo looks like:

```
private void logSecurityInfo(String methodName)
{
Subject subject;
logger.info("SECURITY INFO FOR METHOD '" + methodName + "'");
// Print out the caller
String identity = SecurityManager.getManager().getCaller();
logger.info("The caller is:" + identity);
// Print out the caller's subject
subject = JaasSubject.getCurrentSubject();
logger.info("INVOCATION SUBJECT");
logger.info(subject==null?"NULL":subject.toString());
// Print out service subject
logger.info("SERVICE SUBJECT");
subject = SecurityManager.getManager().getServiceSubject(base);
logger.info(subject==null?"NULL":subject.toString());
// Print out system subject
logger.info("SYSTEM SUBJECT");
try{
subject = SecurityManager.getManager().getSystemSubject();
logger.info(subject==null?"NULL":subject.toString());
}catch(Exception e)
{
logger.warn("Unable to obtain service subject");
}
}
```

Enabling security in a grid service doesn't affect the server-side code at all.

Step 3: Deployment descriptor parameters

So, the deployment descriptor contains following more lines to secure the service.

```
<?xml version="1.0"?>
<deployment name="defaultServerConfig"
xmlns="http://xml.apache.org/axis/wsdd/"
xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
<service name="progtutorial/security/first/MathService"
provider="Handler" style="<parameter name="name" value="Secure
MathService"/>
<parameter name="schemaPath"
value="schema/progtutorial/MathService/Math_service.<parameter
name="className"
value="org.globus.progtutorial.stubs.MathService.MathPortType"/>
<parameter name="operationProviders"
value="org.globus.progtutorial.services.security.first.impl.MathProvider
"/>
<parameter name="baseClassName"
value="org.globus.ogsa.impl.ogsi.GridServiceImpl"/>
<parameter name="securityConfig"
value="org/globus/ogsa/impl/security/descriptor/gsi-security-
config.xml"/>
<parameter name="authorization" value="none"/>
<!-- Start common parameters -->
<parameter name="allowedMethods" value="*"/>
<parameter name="persistent" value="true"/>
<parameter name="handlerClass"
value="org.globus.ogsa.handlers.RPCURIProvider"/>
</service>
</deployment>
```

Step 4: Gar file is created using Ant like simple grid service

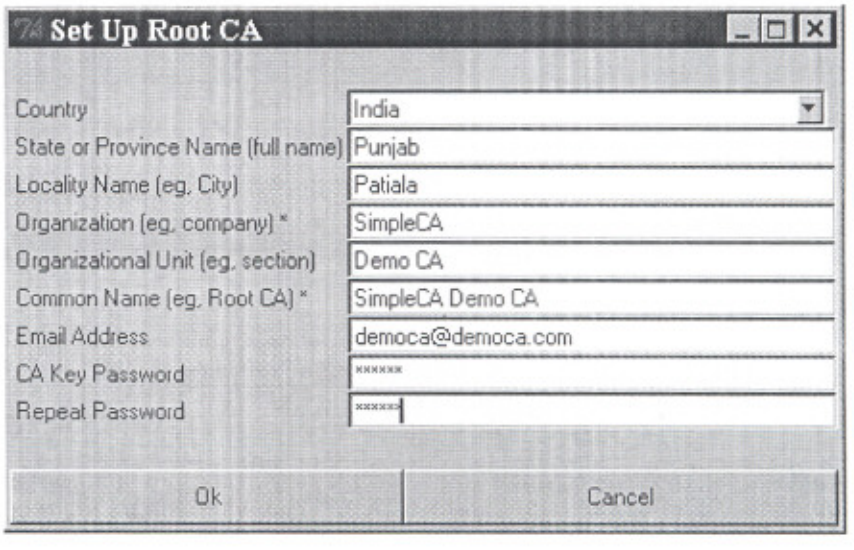
Step 5: The service is deployed in container.

A SECURE CLIENT

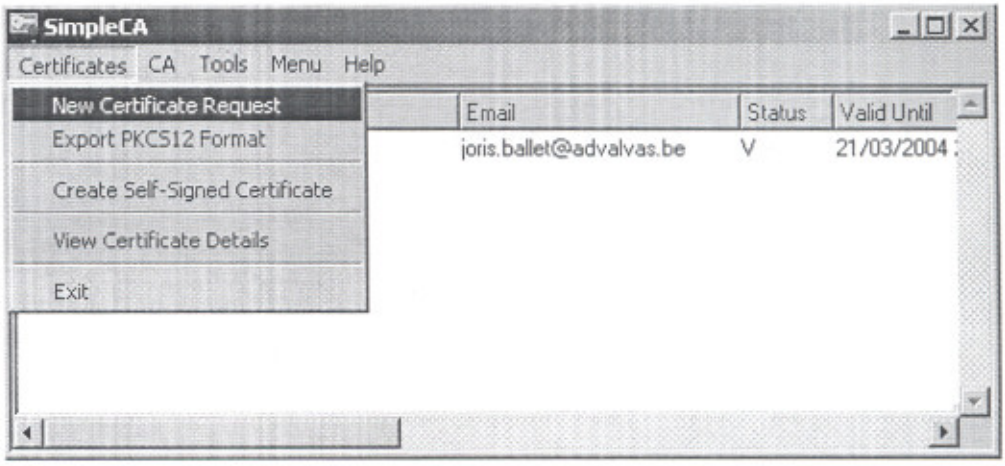
```
import
org.globus.progtutorial.stubs.MathService.service.MathServiceGridLocator
;
import org.globus.progtutorial.stubs.MathService.MathPortType;
import org.globus.ogsa.impl.security.Constants;
import org.globus.ogsa.impl.security.authorization.NoAuthorization;
import javax.xml.rpc.Stub;
import java.net.URL;
public class ClientGSIConvEncrypt
{
public static void main(String[] args)
{
try
{
// Get command-line arguments
```

```
URL GSH = new java.net.URL(args[0]);
int a = Integer.parseInt(args[1]);
// Get a reference to the MathService instance
MathServiceGridLocator mathLocator = new MathServiceGridLocator();
MathPortType math = mathLocator.getMathServicePort(GSH);
// Setup security options
((Stub)math)._setProperty(Constants.GSI_SEC_CONV,Constants.ENCRYPTION);
((Stub)math)._setProperty(Constants.AUTHORIZATION,NoAuthorization.getIns
tance());
// Call remote method 'add'
try{
math.add(a);
System.out.println("Added " + a);
} catch(Exception e)
{
:::::::::

:::::::::

:::::::::

}
```

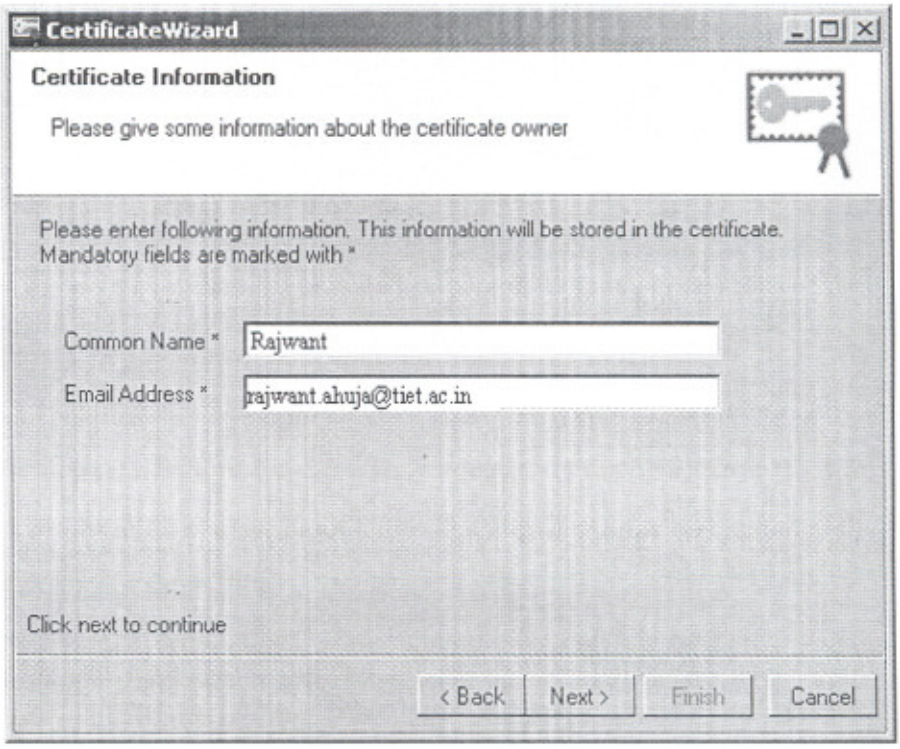
Before running any of the client applications, need to create a proxy certificate for user account. The default behavior in the client-side is to use a proxy certificate for authentication, so we need to create one first. The Proxy certificates are created using the SimpleCA package. We first need to setup the RootCA



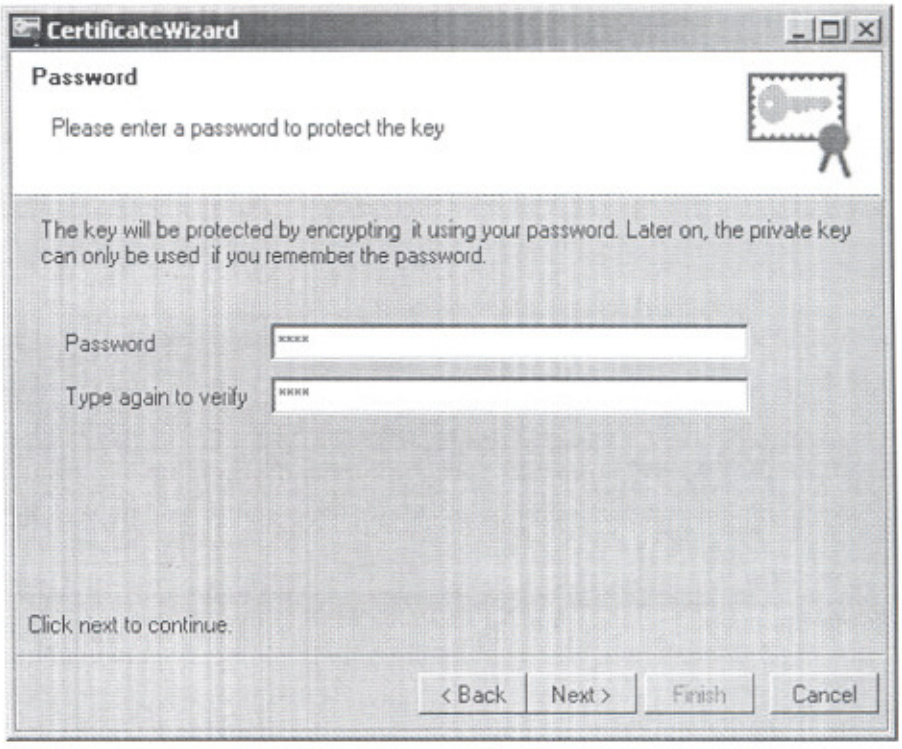
Once the Root CA setup is completed, SimpleCA can be used and Certificate is requested as following.



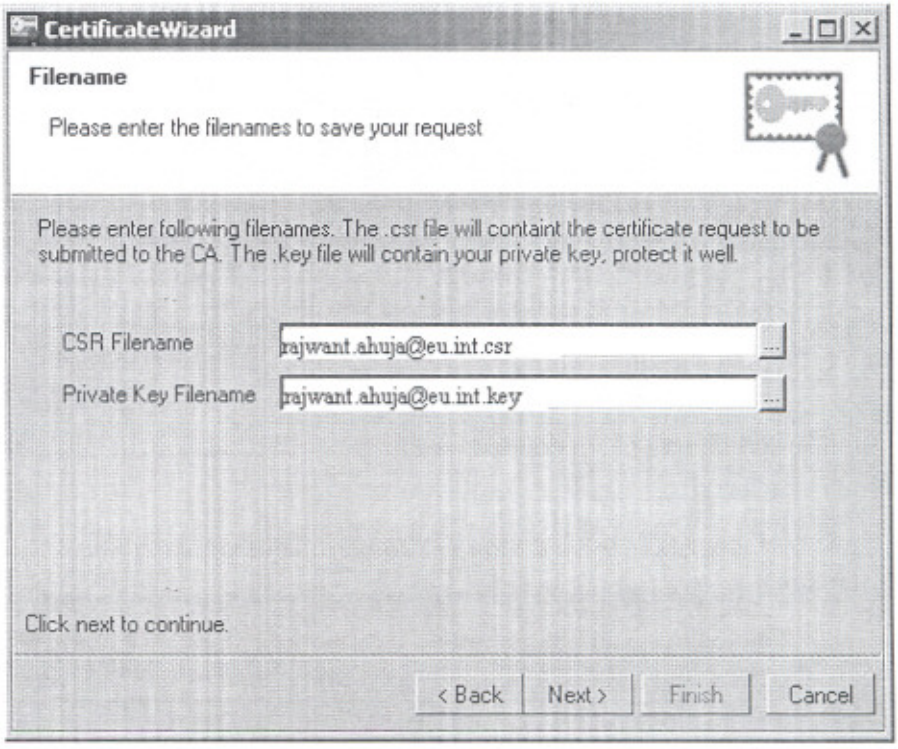
Certificate information is to be provided so that certificate can be created.



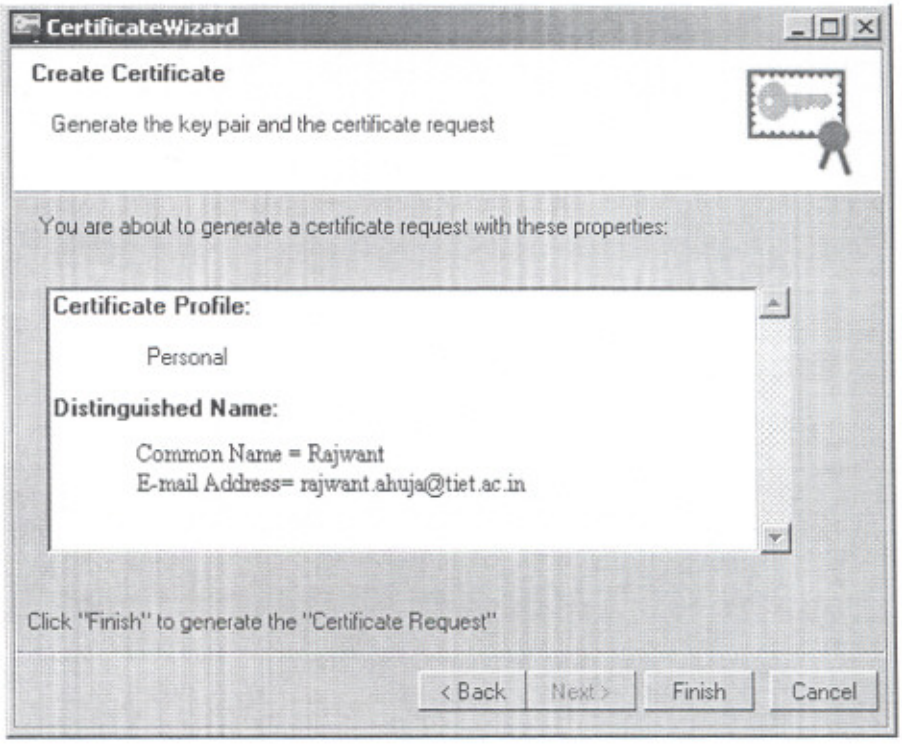
Password is to be provided to make it secure.



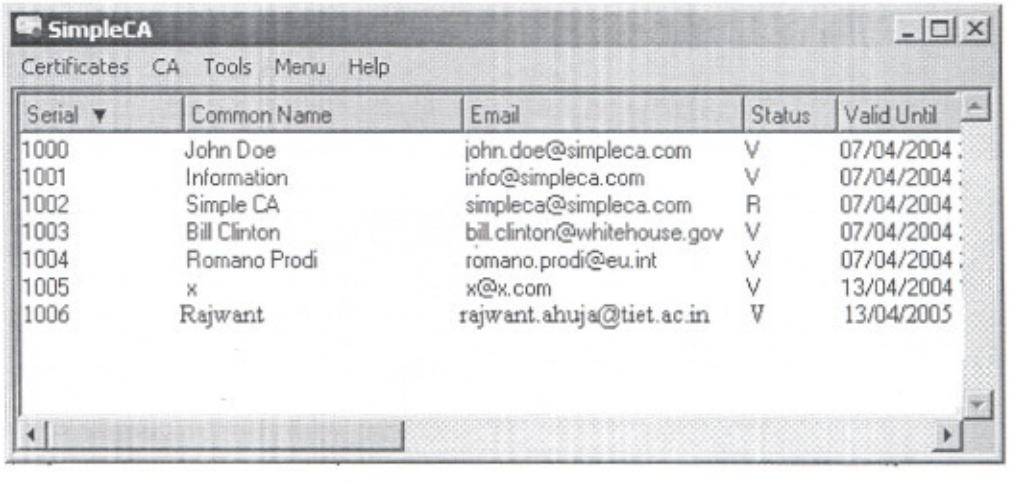
CSR file name and file where Private Key exists is to be provided.



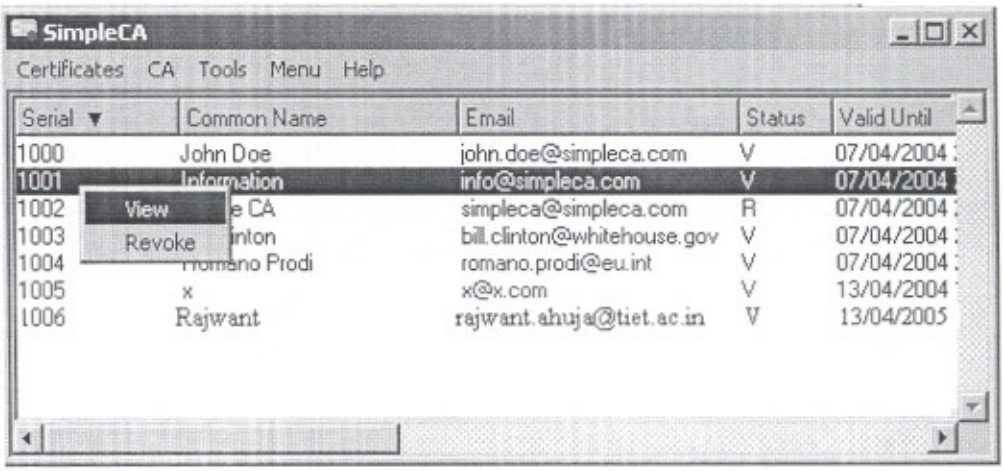
Certificate request are then made using the following properties.



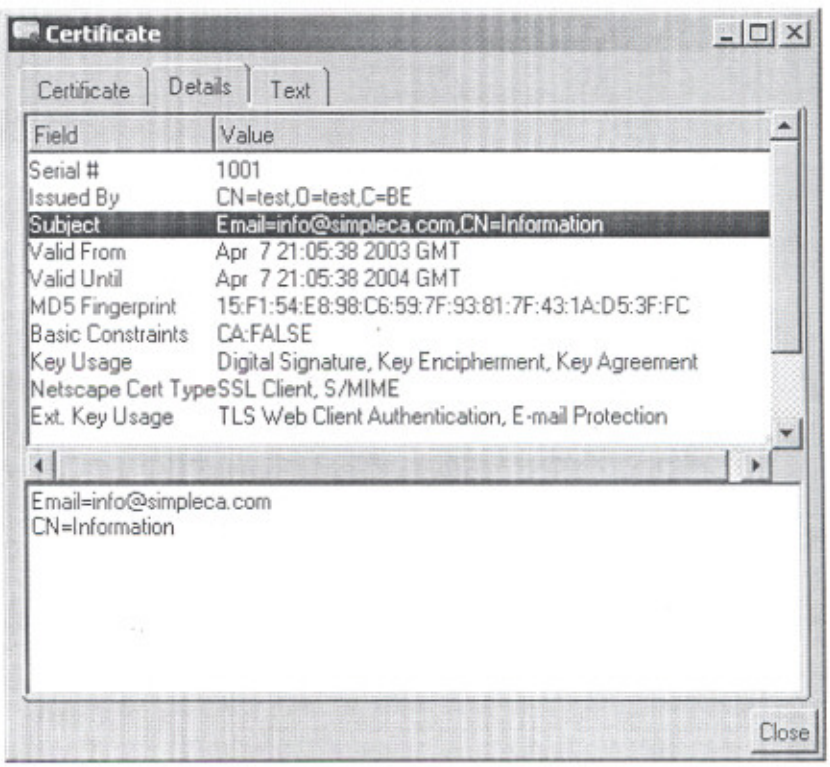
Now, Main Window will look like

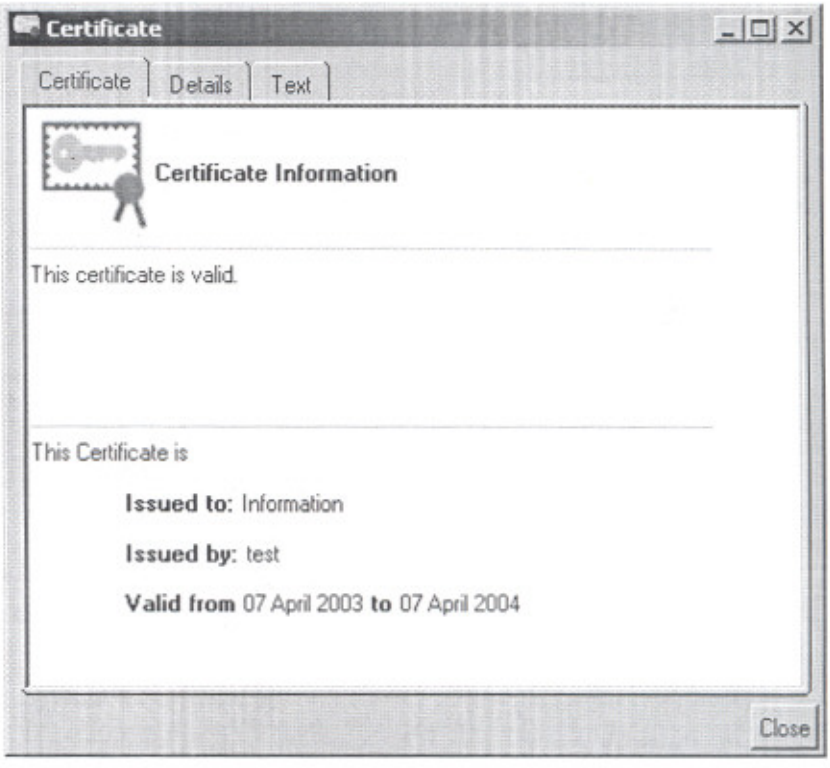


We can view any of the Certificates existing in the SimpleCA list

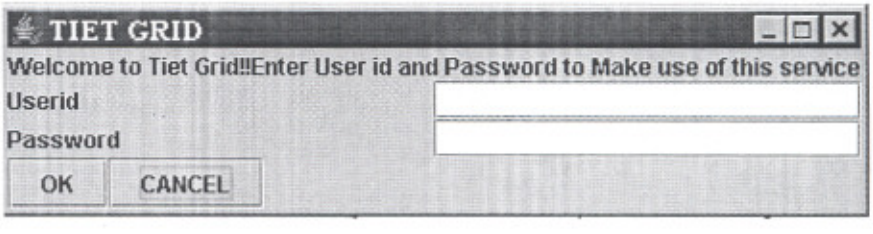


The Certificate of Information will look like

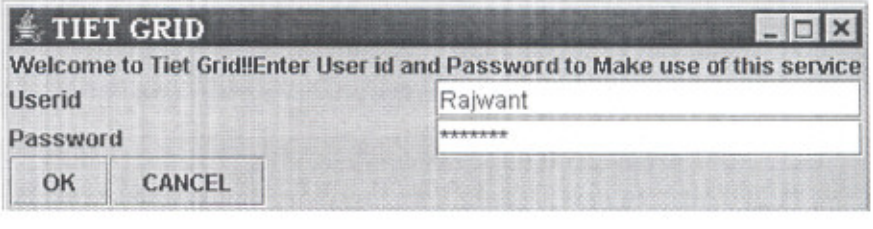


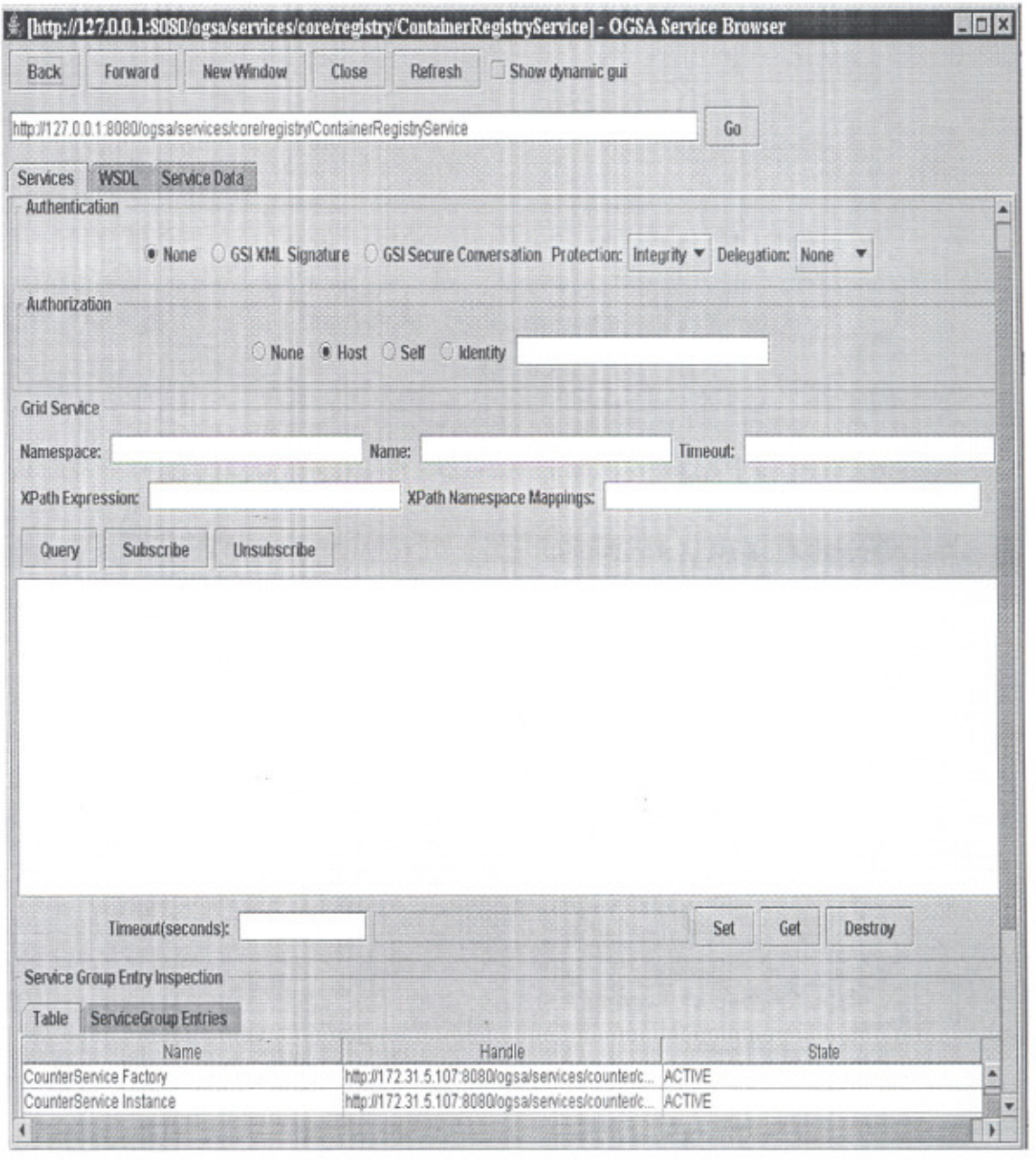


Login Window, Here the user needs to enter user id and password to make use of the service.

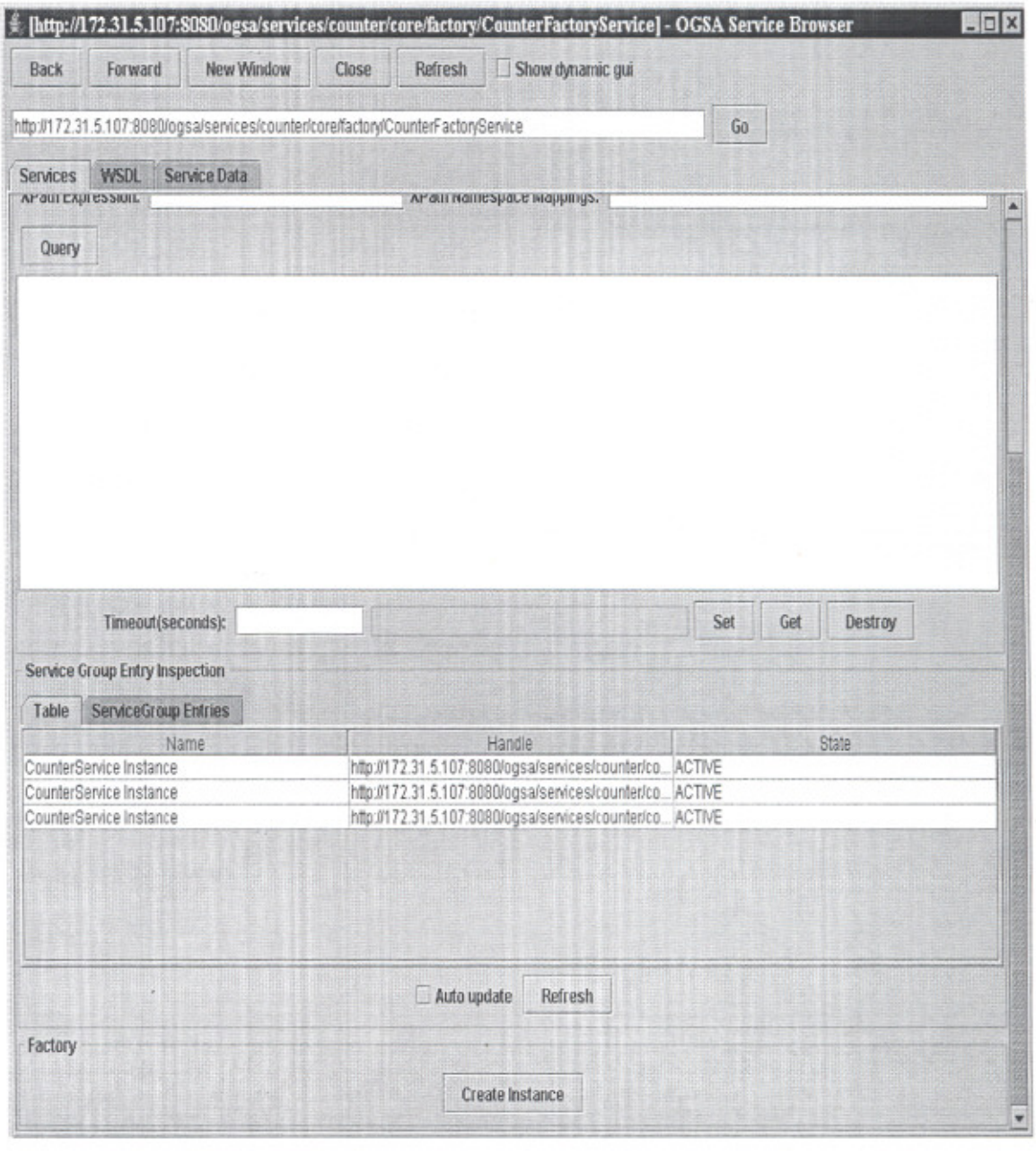


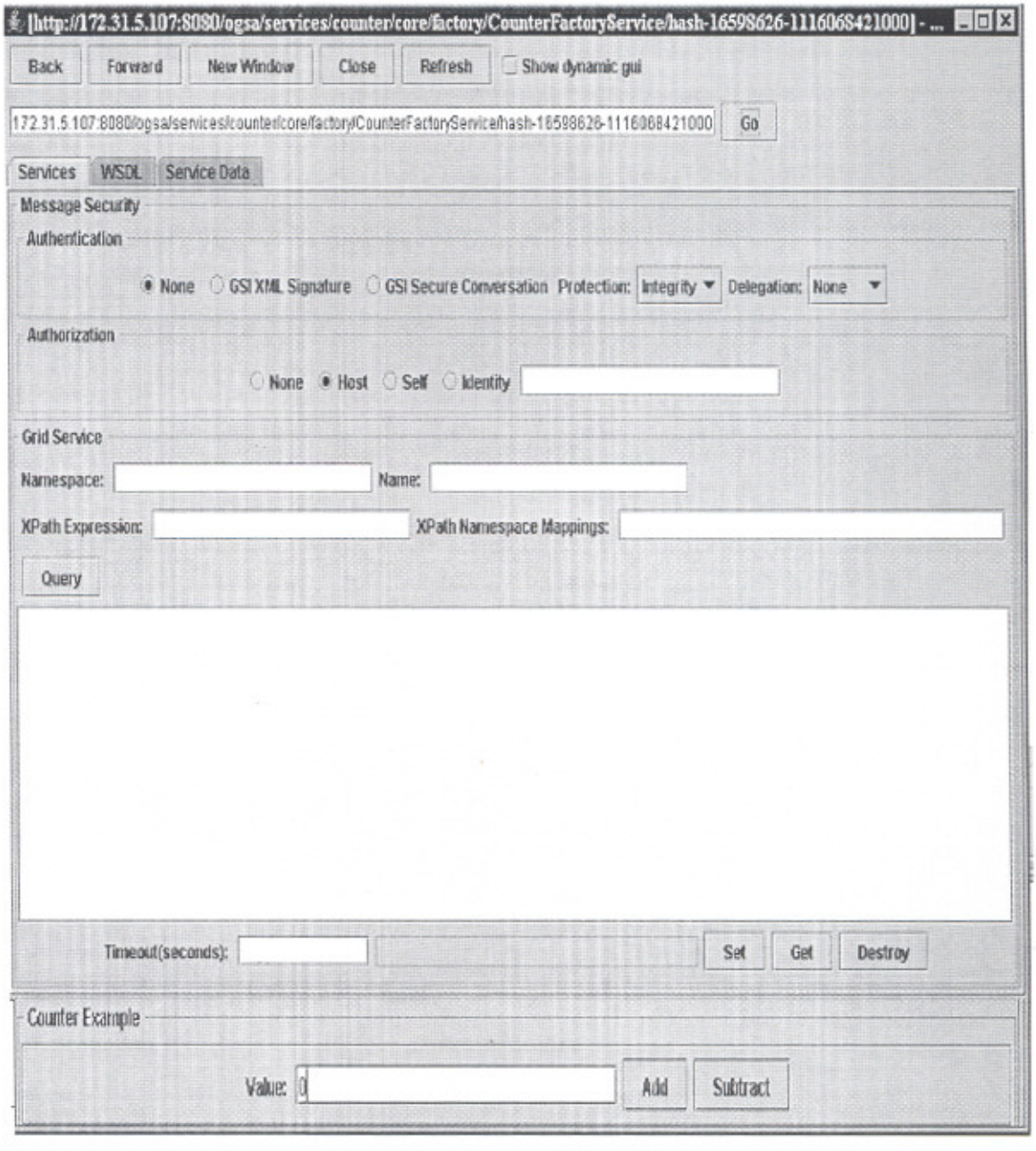
Entry is checked from the database if the entered user is authorized then only the service will be open to use.

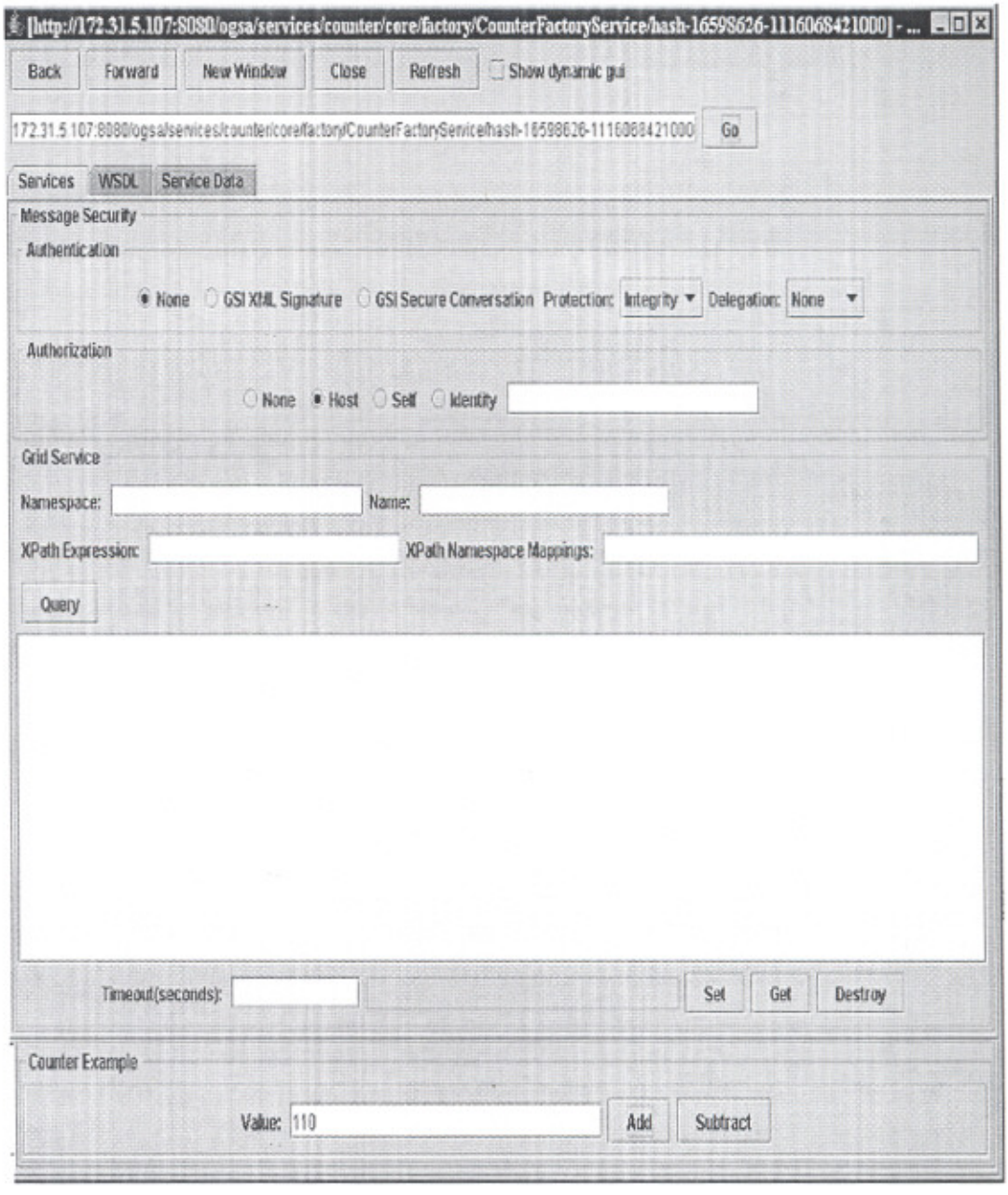


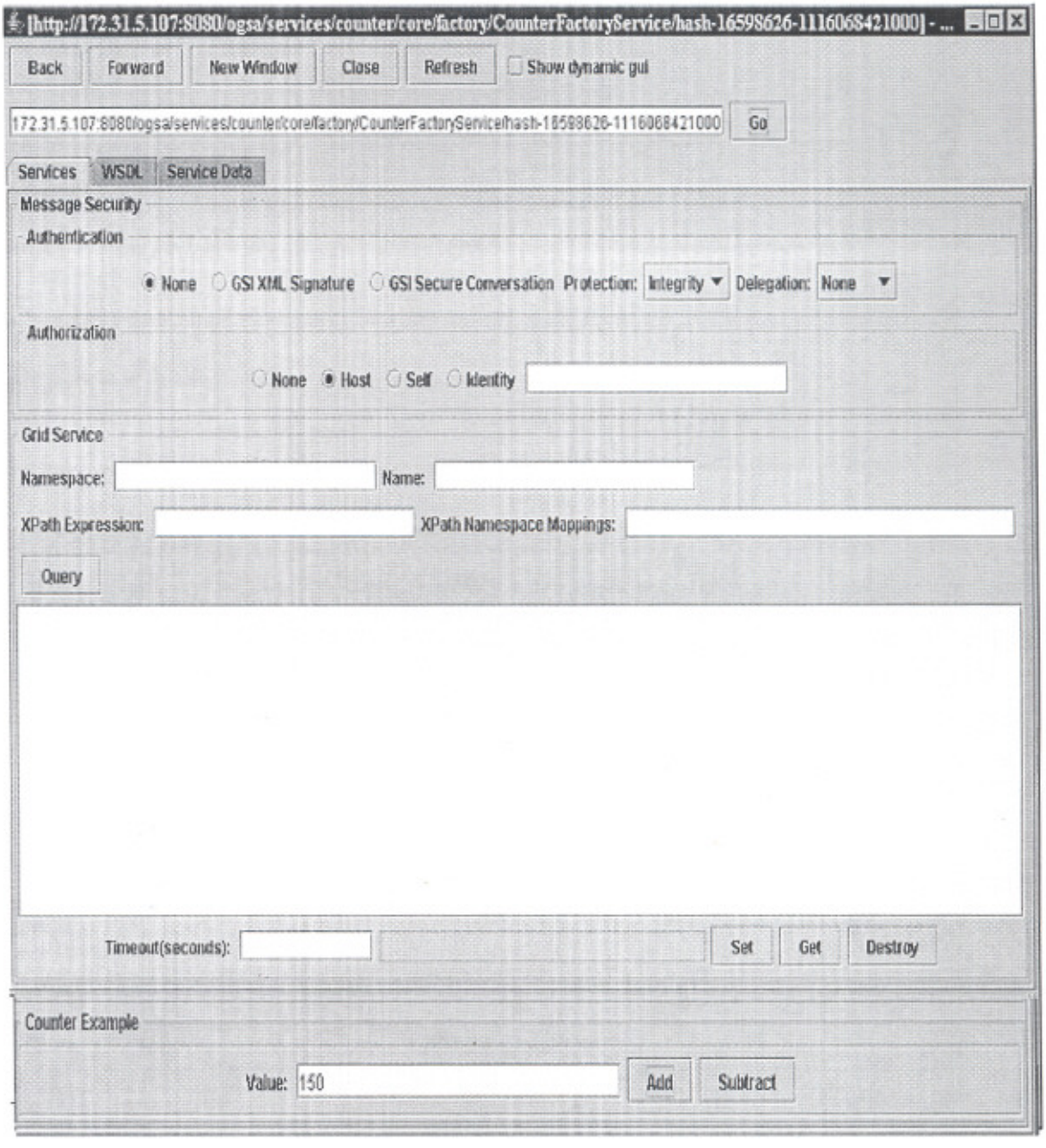

[http://127.0.0.1:8080/ogsa/services/core/registry/ContainerRegistryService] - OGSA Service Browser
Back
Forward
New Window
Close
Refresh
Show dynamic gui
http://127.0.0.1:8080/ogsa/services/core/registry/ContainerRegistryService
Go
Services
WSDL
Service Data
Authentication
None
GSI XML Signature
GSI Secure Conversation
Protection:
Integrity
Delegation:
None
Authorization
None
Host
Self
Identity
Grid Service
Namespace:
Name:
Timeout:
XPath Expression:
XPath Namespace Mappings:
Query
Subscribe
Unsubscribe
Timeout(seconds):
Set
Get
Destroy
Service Group Entry Inspection
Table
ServiceGroup Entries
Name
Handle
State
CounterService Factory
http://172.31.5.107:8080/ogsa/services/counter/c...
ACTIVE
CounterService Instance
http://172.31.5.107:8080/ogsa/services/counter/c...
ACTIVE


[http://127.0.0.1:8080/ogsa/services/core/registry/ContainerRegistryService] - OGSA Service Browser
Back
Forward
New Window
Close
Refresh
Show dynamic gui
http://127.0.0.1:8080/ogsa/services/core/registry/ContainerRegistryService
Go
Services
WSDL
Service Data
Namespace:
Name:
Timeout:
XPath Expression:
XPath Namespace Mappings:
Query
Subscribe
Unsubscribe
Timeout(seconds):
Set
Get
Destroy
Service Group Entry Inspection
Table
ServiceGroup Entries
Name
Handle
State
progtutorial/core/first/MathService
http://172.31.5.107:8080/ogsa/services/progtutori...
INACTIVE
progtutorial/core/first/MathFactoryService
http://172.31.5.107:8080/ogsa/services/progtutori...
INACTIVE
CounterService Factory
http://172.31.5.107:8080/ogsa/services/counter/c...
ACTIVE
SecureCounterService Factory
http://172.31.5.107:8080/ogsa/services/counter/c...
ACTIVE
CounterService Instance
http://172.31.5.107:8080/ogsa/services/counter/c...
ACTIVE
CounterService Instance
http://172.31.5.107:8080/ogsa/services/counter/c...
ACTIVE
core/admin/AdminService
http://172.31.5.107:8080/ogsa/services/core/admi...
INACTIVE
core/management/OgsiManagementService
http://172.31.5.107:8080/ogsa/services/core/man...
INACTIVE
Auto update
Refresh


[http://172.31.5.107:8080/ogsa/services/counter/core/factory/CounterFactoryService] - OGSA Service Browser
Back
Forward
New Window
Close
Refresh
Show dynamic gui
http://172.31.5.107:8080/ogsa/services/counter/core/factory/CounterFactoryService
Go
Services
WSDL
Service Data
Query
Timeout(seconds):
Set
Get
Destroy
Service Group Entry Inspection
Table
ServiceGroup Entries
Name
Handle
State
CounterService Instance
http://172.31.5.107:8080/ogsa/services/counter/co...
ACTIVE
CounterService Instance
http://172.31.5.107:8080/ogsa/services/counter/co...
ACTIVE
CounterService Instance
http://172.31.5.107:8080/ogsa/services/counter/co...
ACTIVE
Auto update
Refresh
Factory
Create Instance


[http://172.31.5.107:8080/ogsa/services/counter/core/factory/CounterFactoryService/hash-16598626-1116068421000] - ...
Back
Forward
New Window
Close
Refresh
Show dynamic gui
172.31.5.107:8080/ogsa/services/counter/core/factory/CounterFactoryService/hash-16598626-1116068421000
Go
Services
WSDL
Service Data
Message Security
Authentication
None
GSI XML Signature
GSI Secure Conversation
Protection:
Integrity
Delegation:
None
Authorization
None
Host
Self
Identity
Grid Service
Namespace:
Name:
XPath Expression:
XPath Namespace Mappings:
Query
Timeout(seconds):
Set
Get
Destroy
Counter Example
Value:
0
Add
Subtract


[http://172.31.5.107:8080/ogsa/services/counter/core/factory/CounterFactoryService/hash-16598626-1116068421000] - ...
Back
Forward
New Window
Close
Refresh
Show dynamic gui
172.31.5.107:8080/ogsa/services/counter/core/factory/CounterFactoryService/hash-16598626-1116068421000
Go
Services
WSDL
Service Data
Message Security
Authentication
None
GSI XML Signature
GSI Secure Conversation
Protection:
Integrity
Delegation:
None
Authorization
None
Host
Self
Identity
Grid Service
Namespace:
Name:
XPath Expression:
XPath Namespace Mappings:
Query
Timeout(seconds):
Set
Get
Destroy
Counter Example
Value:
110
Add
Subtract


[http://172.31.5.107:8080/ogsa/services/counter/core/factory/CounterFactoryService/hash-16598626-1116068421000] - ...
Back
Forward
New Window
Close
Refresh
Show dynamic gui
172.31.5.107:8080/ogsa/services/counter/core/factory/CounterFactoryService/hash-16598626-1116068421000
Go
Services
WSDL
Service Data
Message Security
Authentication
None
GSI XML Signature
GSI Secure Conversation
Protection:
Integrity
Delegation:
None
Authorization
None
Host
Self
Identity
Grid Service
Namespace:
Name:
XPath Expression:
XPath Namespace Mappings:
Query
Timeout(seconds):
Set
Get
Destroy
Counter Example
Value:
150
Add
Subtract

Chapter 5

Conclusion and Future Scope

5.1 CONCLUSION

Computer security is machine access centric. Network security is network access centric. Grid security is application centric. Security requirements in grid are more than security requirements in distributed systems as grid includes virtual organization. To secure distributed system, various security technologies are discussed in this thesis. Grid computing is concerned with sharing and coordinated use of diverse resources in distributed "virtual organizations". The Globus toolkit includes different architecture and infrastructure which help to provide required security in Grid environment.

Grid computing presents a number of security challenges that are met by the Globus Toolkit's Grid Security Infrastructure (GSI). The grid security model must be able to leverage existing security standards for their QoS requirement on secure message exchange. The Open Grid Service Architecture (OGSA) asserts that the OGSA security model needs to be consistent with the Web service security model.

Version 3 of the Globus Toolkit (GT3) implements the emerging OGSA; its GSI implementation (GSI3) takes advantage of this evolution to improve on the security model used in earlier versions of the toolkit. Currently, GSI of GT3 provides maximum security to the Grid environment. Web Services are distributed computing technology. Web services have few limitation which make it unfit for its use in grid. So, for grid the extension of web service called grid services are used. Grid Service is like web service which includes the basic functionality and can be used by the user. Here, grid services are made using the Globus Toolkit version 3. Simple Grid Service and secure Grid service are implemented. Any user can make use of the simple service but if secure grid service exists, then only limited number of users can make use of the method existing in the grid

service. The grid map file is used which include the information about the list of authorized users.

One of the major challenges in writing grid service is to provide appropriate security and to get the security to work effectively. Security mechanisms available with GT3 have been discussed and demonstrate how to create grid service and secure Grid service.

5.2 FUTURE SCOPE

In this thesis, Simple and secure Grid service are implemented using the Globus toolkit version 3 (GT3). Grid service with limited functionality is developed and then security issues are applied to that service only, to implement the secure grid service. A typical Secure grid service with huge and complex functionality can be developed by following the same steps.

Globus toolkit Version 4 is also released on 2 May, 2005, by Globus Alliance. In future, rather than using version 3, grid service can be developed with GT4. Key additions to GT4 include:

-- GT4 complies with the latest Web Services Interoperability Organization (WS-I) web services standards, which provides maximum interoperability between different environments.

-- GT4 includes initial support for important authorization standards, including Security Markup Language (SAML) and Extensible Access Control Markup Language (XACML). These provide business with a foundation for building a secure web-services enabled Grid infrastructure.

-- GT4 implements the Web Services Resource Framework (WS-RF) and Web Services Notification (WS-N) specifications, which are emerging standards in OASIS backed by major vendors for web services enablement of Grid and resource management systems.

-- GT4 features sophisticated authorization and security capabilities- Globus has always been diligent in Grid security, and GT4 is also "enterprise ready" from a security perspective.

References

[1] **I. Foster and C. Kesselman**. "*Globus: A toolkit based grid architecture.*" In *[2]*, pages 259 – 278, 1999.

[2] **I. Foster, C. Kesselman, J. Nick and S. Tuecke, "***The Physiology of the Grid: An Open Grid Services architecture for Distributed Systems Integration*", Globus Project, 2002.

[3] **G. coulouris, J. Dollimore, T. Kindberg, "***Distributed system*", Edition 2001.

[4] **I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke,** *"Security architecture for computational grids"*, In ACM Conference on Computers and Security, pages (83-91) ACM Press, 1998.

[5] **Jennifer M. Schopf**, "*Grids: The Top Ten Questions*", Mathematics and Computer Science Division, Argonne National Lab, Department of Computer Science, Northwestern University, 2002.

[6] **J. Joseph, C. Fellenstein,** "Grid Computing", LPE, Pearson Education, Edition 2004.

[7] **B. Saxena**, "*Grid Computing*", Thesis submitted to Computer Science and Department, Thapar Institute of Engineering and technology, 2003.

[8] **R.Butler, D. Engerty, S. Tueckez, J. Volmery, V. Welch,** *"Design and Deployment of a National-Scale Authentication Infrastructure"*, 2000.

[9] **V. Welch, F. Siebenlist, K. Czajkowski, J. Gawo, S. Meder,** *"Security for Grid Services,* 2003.

[10] **Wm A.Wulf, C. Wang, D. Kienzle,** *"A new Model of Security for distributed Systems"*, Computer Science Technical report, Pages: 34 – 43, 1996.

[11] **V. Talwar, R. Kumar, J. Dwoskin, S. Basu**, *"Scoping Security Issues for Interactive Grids"*, HP lab, 2003.

[12] **Michael Brown, Srikrishnan Sundararajan** *"Globus Toolkit 3.0 Quick Start", IBM, Sept 2003.*

[13] **Ali Nasrat Haidar,** *"Critical Evaluation of Current Approaches to Grid Security"*, University of London, 2003

[14] **Vinay Bansal,** *"Policy Based Firewall for GRID Security"*, Dept. of Computer Science, Duke University, 2002

[15] **Von Welch1, Frank Siebenlist, Ian Foster** *"GSI3: Security for Grid Services"*, University of Chicago, Feb 2003

[16] **Jason Reid,** *"Configuring the Secure Shell Software"* Sun BluePrints™ OnLine—April 2003

[17] **Jean Carlo Binder** *"Introduction to PKI - Public Key Infrastructure"* Version1.1, 2002.

[18] **Nataraj Nagaratnam, Philippe Janson,** *"The Security Architecture for Open Grid Services"* 2002.

[19] **Lavanya Ramakrishnan** *"Writing secure grid services using Globus Toolkit3.0"*,IBM Works, 2003.

[20] **Borja Sotomayor,** *"The Globus Toolkit 3Programmer's Tutorial"*, 2004.

[21] Globus Grid GT3 Install / Setup on Windows 2000 from www.bigdogsoftware.com

[22] **Luis Ferreira, Arun Thakore, Michael Brown,** *"Grid Services Programming and Application Enablement"*, IBM, 2004.

[23] **Bart Jacob, Luis Ferreira, Norbert Bieberstein,** *"Enabling Applications for Grid Computing with Globus", IBM, 2003.*

[24] **Satoshi Shirasuna, Aleksander Slominski,** "P*erformance Comparison of Security Mechanisms for Grid Services"*, Indiana University.

[25] **Von Welch, Ian Foster,** *"X.509 Proxy Certificates for Dynamic Delegation",IEEE, 2004.*

[26] **H.W. Lim, M.J.B. Robshaw,** *"A Dynamic Key Infrastructure for Grid"* University of London.

List of Papers

[1] **Rajwant Kaur Ahuja, Seema Bawa** "*Identification and Elimination of Security Threats in Grid Computing Environment*" Paper - 07, in BVCON'2005 National Level Conference on "Cyber Crime: Past, Present and Future" organized by Bharati Vidyapeeth's, Sangli on March 11-12th 2005 **(Published)**.

[2] **Rajwant Kaur Ahuja, Seema Bawa** "*Creating Secure Grid Services using Globus toolkit*", in International Conference on "Computer & Information Science", ICCIS'05 to be held in Kolhapur, India on December 15-17th 2005 **(Communicated)**.

[3] **Rajwant Kaur Ahuja, Seema Bawa** "*Implementing Secure Grid Services using GT3*", in 6th IEEE/ACM International Workshop on "Grid Computing", GRID'2005 to be held in Washington, USA on November 13-14th 2005 **(Communicated)**.