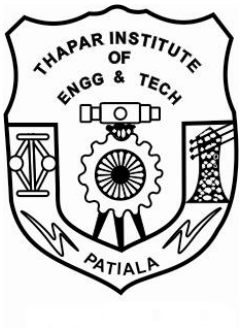
Development of TIETGrid Portal Using Web Services

A Thesis

submitted in partial fulfillment of the
requirements for the award of degree of

Master of Engineering
in
Software Engineering



By:
Gurleen Kaur Doad
(8043107)

Under the supervision of:
Dr. (Mrs.) Seema Bawa
Professor and Head
Department of Computer Science & Engineering

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY
(DEEMED UNIVERSITY)
PATIALA – 147004

MAY 2006

CERTIFICATE

I hereby certify that the work which is being presented in the thesis entitled, **"Development of TIETGrid Portal Using Web Services"**, in partial fulfillment of the requirements for the award of degree of Master of Engineering in Software Engineering submitted in Computer Science and Engineering Department of Thapar Institute of Engineering and Technology (Deemed University), Patiala, is an authentic record of my own work carried out under the supervision of *Dr. Seema Bawa.*

The matter presented in this thesis has not been submitted for the award of any other degree of this or any other university.

Gurleen Kaur Doad

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

Dr. (Mrs.) Seema Bawa
Supervisor, Professor and Head
Computer Science and Engineering Department
Thapar Institute of Engineering and Technology
Patiala-147004

Countersigned by

(Dr. Seema Bawa)
Head,Computer Sc.& Engg. Deptt.
Thapar Institute of Engg. & Tech.
Patiala-147004

(Dr. T.P. Singh)
Dean of Academic Affairs
Thapar Institute of Engg. & Tech.
Patiala-147004

ACKNOWLEDGEMENT

First and foremost, I would like to express my sincere gratitude to my advisor Seema Bawa, Professor& Head, Computer Science & Engineering Department for her immense help, guidance, stimulating suggestions and encouragement all the time. She always provided a motivating and enthusiastic atmosphere to work with; it was a great pleasure to do this thesis under her supervision.

I am equally grateful to Maninder Singh, Assistant Professor, Computer Science & Engineering Department, a nice person, an excellent teacher and a well-credited researcher, who always encouraged me to keep going with work and always advised me with his invaluable suggestions.

I am most indebted to the P.G coordinator Mr. Rajesh Bhatia for his help, assistance and suggestions during my work.

I would also like to express my appreciation to the TIETGrid Group. The group held weekly meetings on Thursdays to address various issues and share experiences. Thus, it invoked better understanding of the work and acted as a rostrum for information sharing and problem solving.

Finally, my special thanks go to my family for their never-ending love and support. Nothing would have ever been possible without my parent's unconditional love and encouragement.

The most valuable thing I acquired from the two years study in TIET is to protect, survive and endure myself in this world with increased confidence and additional faith in my abilities to achieve.

Last but not the least, I would like to thank the lord for his blessings and for driving me with faith, hope and courage in the thinnest of the times.

Gurleen Kaur Doad
(8043107)

ABSTRACT

Grid technology is designed to allow users seamless access to applications and services running on remote resources. Grids are becoming platforms for high-performance and distributed computing. Grids benefit users by permitting them to access heterogeneous resources, such as machines, data, people and devices that are distributed geographically and organizationally.

Globus Toolkit4 has been widely adopted as a Grid technology solution for scientific and technical computing. It implements web services mechanisms for building distributed systems. Web Services are the software components that have emerged as a popular standards-based framework for accessing network applications. Since majority of the grid applications are research-oriented, so there is need to provide a user-interface that does not involve much about grid programming.

This thesis describes an approach to building grid services based on the premise that users who wish to access and run these services prefer to do so without becoming experts on grid technology. These services are visible to the users and to resource providers through a family of Grid portal components that can be used to configure, launch, and monitor complex applications in the scientific language of the end user.

The work done as a part of the thesis involves developing a web service and deploying it in the GT4 container. This is followed by development of a TIET grid portlet that provides a user-friendly interface to this service.

TABLE OF CONTENTS

LIST OF FIGURES

ORGANISATION OF THESIS

The first chapter briefly introduces grid computing, it's need, architecture, applications and also covers the history of grid computing. The second chapter captures the problem statement, which deals with deploying a GT4 web service in the container and then accessing the web service via a portlet. The third chapter explores the background information needed for the thesis work covering Globus Toolkit4, which has been widely adopted as a Grid technology solution for scientific and technical computing, and Web services, which have emerged as a popular standards-based framework for accessing network applications. The fourth chapter provides a brief overview of portals, portlets, and grid portal design and finally, it explains an open-source portal framework- Gridsphere. The fifth chapter is all about the practical demonstration containing writing and deploying a WSRF web service and then accessing this web service via a portlet in the T.I.E.T Grid Portal. Finally the last chapter discusses the future scope of the work.

CHAPTER 1
INTRODUCTION

When most people think of a grid, the idea of an interconnected system for the distribution of electricity or electromagnetic signals over a wide area, especially a network of high-tension cables and power stations, comes to mind. Around 1995, this same concept was applied to computing.

What is grid and grid computing? According to IBM [1]: "A grid is a collection of distributed computing resources available over a local or wide area network that appear to an end user or application as one large virtual computing system. The vision is to create virtual dynamic organizations through secure, coordinated resource sharing among individuals, institutions, and resources. Grid computing is an approach to distributed computing that spans not only locations but also organizations, machine architectures and software boundaries to provide unlimited power, collaboration and information access to everyone connected to grid.

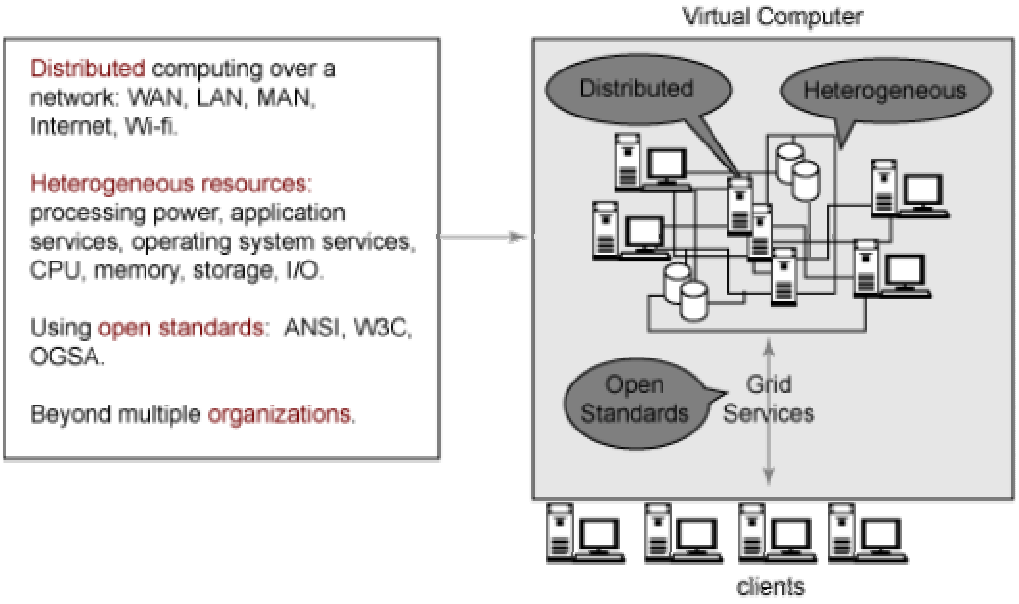


Figure 1.1: Grid computing at a glance [3]

Grid computing is about getting computers to work together. Almost every organization has enormous unused computing capacity. Various servers are actually

serving something less than 10 percent of the time. Most PCs do nothing for 95 percent of a typical day. Imagine an airline with 90 percent of its fleet on the ground, an automaker with 40 percent of its assembly plants idle, a hotel chain with 95 percent of its rooms unoccupied.

1.1 Revisiting Distributed Computing

Distributed computing [1-5] is often used to describe a type of computing in which different computing nodes can simultaneously run an application located on different computers that are connected by a communication network. The main motivation for constructing distributed systems is to obtain large-scale resource sharing at affordable cost. Advances in networking technology and computational infrastructure make it possible to construct large-scale high performance distributed computing environments that provide dependable, consistent and pervasive access to high-end computational and heterogeneous resources despite geographical distribution of both resources and users. Grid computing has been widely seen as a step beyond the conventional distributed computing, incorporating pervasive high-bandwidth, high-speed computing, intelligent sensors and large scale database into a seamless pool of managed and brokered resources. These kinds of large-scale high performance distributed services have recently received substantial interest from both research as well as industrial point of view.

Hence, grid computing is a form of distributed computing that involves coordinating and sharing computing, application, data, storage, or network resources across dynamic and geographically dispersed organizations. Grid technologies promise to change the way organizations tackle complex computational problems.

How does Grid Computing surpass Distributed Computing?

Distributed applications already exist, but they tend to be specialized systems intended for a single purpose or user group. Grids go further and take into account:

- Different kinds of resources: Not always the same hardware, data and applications
- Different kinds of interactions: User groups or applications want to interact with grids in different ways.
- Dynamic nature: Resources and users added/removed/changed frequently.

1.2 History of Grid

Many of the basic ideas behind the Grid have been around in one form or other throughout the history of computing. For example, one of the novel ideas of the Grid is sharing computing power. Nowadays, where most people have more than enough computing power on their own PC, sharing is unnecessary for most purposes. But back in the sixties and seventies, sharing computer power was essential. At that time, computing was dominated by huge mainframe computers, which had to be shared by whole organizations.

In 1965 the developers of an operating system called Multics [5], an ancestor of Unix, which in turn is an ancestor of Linux - a popular operating system today presented a vision of computing as a utility - in many ways uncannily like the Grid vision today. Access to the computing resources was envisioned to be exactly like water, gas and electricity - something that the client connects to and pays for according to the amount of use.

So there is a certain amount of "reinventing the wheel" going on in developing the Grid. However, each time the wheel is reinvented, it is reinvented in a much more powerful form, because computer processors, memories and networks improve at an exponential rates which are associated with Moore's law [8].

Because of the huge improvements of the underlying hardware typically more than a factor of 100x every decade, it is fair to say that reinvented wheels are qualitatively different solutions, not just small improvements on their predecessor.

1.3 Need for Grid Computing

In this part, we will discuss how, when and in what circumstances grid emerged as the new infrastructure of the 21st century.

Driven by increasingly complex problems and propelled by increasingly powerful technology, today's science is as much based on computation, data analysis, and collaboration as on the efforts of individual experimentalists and theorists. But even as computer power, data storage, and communication continue to improve

exponentially, computational resources are failing to keep up with what scientists demand of them.

According to a source, a personal computer in 2001 is as fast as a supercomputer of 1990. But 10 years ago, biologists were happy to compute a single molecular structure. But today they want to calculate the structures of complex assemblies of macromolecules and screen thousands of drug candidates. Personal computers now ship with up to 100 gigabytes of storage--as much as an entire 1990 supercomputer center. Several physics projects, CERN's Large Hadron Collider (LHC) [8-9] among them, produce multiple petabytes (10^{15} byte) of data per year. Some wide area networks operate at 155 megabits per second (Mb/s), three orders of magnitude faster than the state-of-the-art 56 kilobits per second (Kb/s) that connected US supercomputer centers in 1985. But to work with colleagues across the world on petabyte data sets, scientists now demand tens of gigabits per second (Gb/s).

The Grid offers a potential means of surmounting these obstacles to progress. Built on the Internet and the World Wide Web, the Grid is a new class of infrastructure. By providing scalable, secure, high-performance mechanisms for discovering and negotiating access to remote resources, the Grid promises to make it possible for scientific collaborations to share resources on an unprecedented scale, and for geographically distributed groups to work together in ways that were previously impossible.

1.4 Characteristics of Grids

There are three main issues that characterize computational grids:

- Heterogeneity: A grid involves a multiplicity of resources that are heterogeneous in nature and might span numerous administrative domains across wide geographical distances.
    - Resources are heterogeneous
    - Resources are administratively disparate
    - Resources are geographically disparate
    - Users do not have to worry about system details (e.g., location, operating system, accounts).

- Resources are numerous.
- Resources have different resource management policies.
- Resources are owned and managed by different, potentially mutually distrustful organizations and individuals that likely have different security policies and practices.

- Scalability: A grid might grow from few resources to millions. This raises the problem of potential performance degradation as a Grids size increases. Consequently, applications that require a large number of geographically located resources must be designed to be extremely latency tolerant.
- Dynamicity or Adaptability: In a grid, a resource failure is the rule, not the exception. In fact, with so many resources in a Grid, the probability of some resource failing is naturally high. The resource managers or applications must tailor their behavior dynamically so as to extract the maximum performance from the available resources and services.

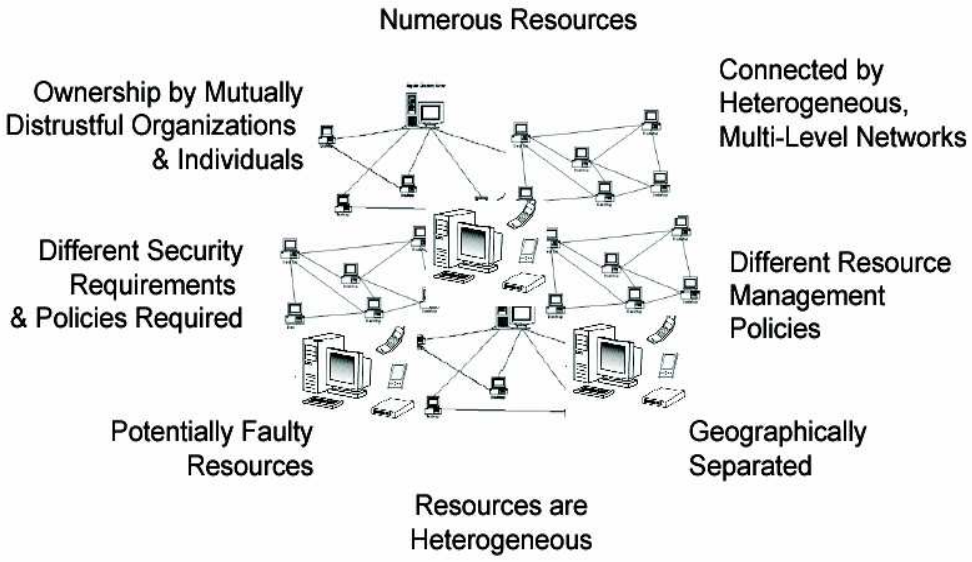


Figure 1.2: Characteristics of Grids [7]

1.5 Grid Architecture

Architecture identifies the fundamental system components, specifies purpose and function of these components, and indicates how these components interact with each

other. Grid architecture is protocol architecture, with protocols defining the basic mechanisms by which VO [3-5] users and resources negotiate, establish, manage and exploit sharing relationships. Grid architecture is also a services standards-based open architecture that facilitates extensibility, interoperability, portability and code sharing

The components that are necessary to form a grid are shown in Figure 2 and they are briefly discussed below:

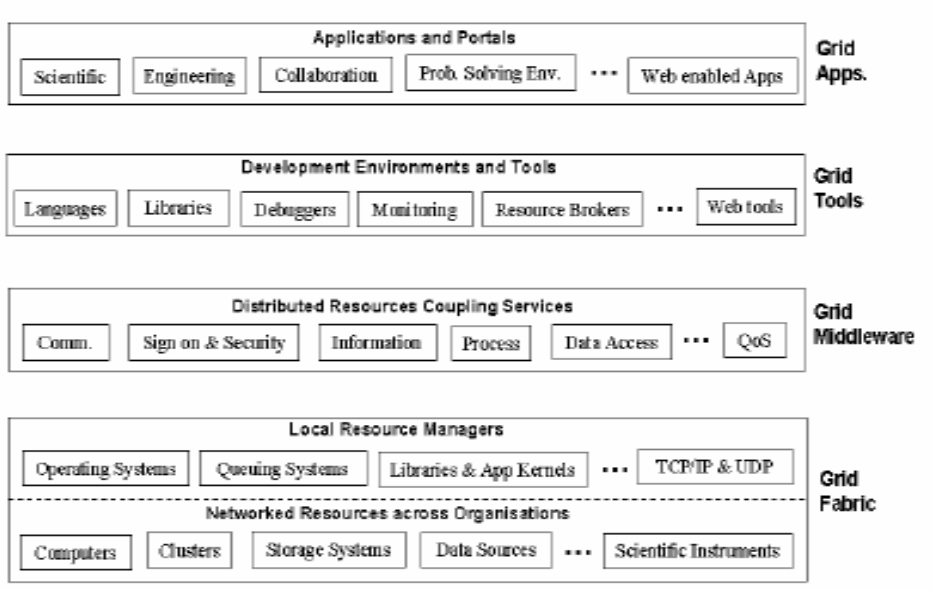


Figure 1.3: Grid Components [5]

- Grid Fabric: It comprises all the resources geographically distributed (across the globe) and accessible from anywhere on the Internet. They could be computers (such as PCs or Workstations running operating systems such as UNIX or NT), clusters (running cluster operating systems or resource management systems such as LSF, Condor or PBS), storage devices, databases, and special scientific instruments such as a radio telescope.
- Grid Middleware: It offers core services such as remote process management, co- allocation of resources, storage access, information (registry), security, authentication, and Quality of Service (QoS) such as resource reservation and trading.

- Grid Development Environments and Tools: These offer high-level services that allows programmers to develop applications and brokers that act as user agents that can manage or schedule computations across global resources.
- Grid Applications and Portals: They are developed using grid-enabled languages such as HPC++, and message-passing systems such as MPI. Applications, such as parameter simulations and grand-challenge problems often require considerable computational power, require access to remote data sets, and may need to interact with scientific instruments. Grid portals offer web-enabled application services — i.e., users can submit and collect results for their jobs on remote resources through a web interface.

1.6 Nature of Grid Applications

An important step in deploying Grid computing is to identify the applications that can benefit from these distributed computing paradigms. For grids, the candidates will be those applications that can be executed in parallel on multiple processors. Parallel applications may be categorized in the following way:

- *Embarrassingly Parallel Computations (EPC)*[22] can be immediately divided into independent parts, which may be allocated to multiple processors for simultaneous execution. Once the work has been allocated, no communication is required between the processors, so a speedup of n is possible, given n processors. Often the problem can be divided into many more parts than there are processors available, so the maximum speedup possible via parallel implementation is only constrained by the number of processors. Examples of EPC applications include testing large integers to determine whether they are prime numbers, and various Monte Carlo simulations.

- *Parametric and Data Parallel Computations* are similar to EPCs, with each processor working on an independent subset of the data or a separate set of parameters. These are also referred to as Nearly Embarrassingly Parallel Computations (NEPC). NEPC [22-23] requires results to be initially distributed among independent computers and then later gathered by a single process, with some

post processing generally required. Internet search engines and low-level digital image rendering provide examples of such applications.

- *Loosely Coupled Synchronous Parallel Computations* require interprocess communication between a small subset of the processors before the computation can be completed. Tightly Coupled Synchronous Parallel Computations require communication between all of the processors (global synchronization) before the computation can be completed. Synchronous applications can be further characterized in the following terms:

  - Degree to which they require simultaneous computation and communications to be performed by the individual processors
  - The distribution of the message sizes that are transferred between nodes
  - Sensitivity of the application to message latency
  - Synchronous applications include clustered databases and a large class of complex simulations of discrete event systems, particle systems, and lumped/continuous variable systems. Among these latter applications are those that were developed to run on parallel processing supercomputers, such as Massively Parallel Processors (MPPs), or Symmetrical Multi-Processing (SMP) computers. Synchronous applications that have already been paralleli zed for MPP or SMP can be readily transitioned to cluster or Grid computing.

1.7 Benefits of Grid Computing

- **High Performance Computing (HPC):** For applications that lend themselves to parallel computing, the Grid offers the possibility of executing computationally intensive applications on networked computer arrays consisting of numerous commodity or specialized systems. Computational Grids [19] can offer significant advantages in both price/performance and in maximum performance over more conventional types of supercomputers. As a result, the Grid makes HPC accessible to more enterprises, accelerates the availability of results of computationally intense

analysis needed for product research and development, and allows scientists to solve grand challenge problems that were too large for conventional supercomputers.

In its current stage of evolution, most applications of the Grid fall into the HPC classification. This is due to the fact that Grid computing arose out of the need for more cost-effective HPC solutions to address critical problems in science and engineering. The initial adoption of the Grid by commercial enterprises has continued to focus on HPC because of the high return on investment and competitive advantage realized by solving compute intensive problems that were previously insolvable in a reasonable period of time or cost. The HPC Grid has successfully addressed a wide range of computational problems in the following fields:

- Climate/weather/ocean modeling and simulation
- Computational chemistry and materials science
- Environmental quality modeling and simulation
- Military forces modeling and simulation
- Internet search engines
- Pharmaceutical research
- Simulation and verification of electronic and mechanical design
- Seismic processing and interpretation
- Signal/image processing
- Modeling of financial portfolios and markets

Data Federation and Collaboration: The Grid also allows a federated approach to data integration where data from different sources (relational databases, files, or application data) can be consolidated in a single data service that hides the complexities of data location, local ownership, and infrastructure from the consuming application. With data federation, the data remains in place at its source, with no disruption of local users, applications, or data management policies. Integration of data from multiple sources and locations facilitates a wide range of integrated applications, including corporate performance dashboards, marketing analysis tools, customer service applications, and data mining applications. Because the data

resource is accessed as a service on the network, minimal modifications of existing data publishing or data-consuming applications is required.

- **Resource Allocation and Optimization:** Most desktop systems and servers are idle more than 80% of the time because computers have traditionally been rigidly dedicated to certain sets of users and applications. Grid computing provides a virtualization framework [23-24] that allows flexible sharing of computing and storage to improve resource utilization. In the simplest example, a batch job can be transparently allocated to an idle server in the pool of resources. Alternatively, both the application and the job could be transferred to an idle server. In addition, if virtual machine software is installed, a single physical server can even be configured on the fly to run different operating systems and applications. In a similar way, idle storage capacity of Network Attached Storage, or SANs, could be utilized for data storage. The primary benefit of resource optimization through virtualization is that it reclaims much of the stranded capacity of the computing infrastructure and therefore reduces the level of capital investment required to support a given level of IT functionality. Another aspect of resource optimization is that it generally does not require modification of existing applications.

While all three categories of Grid application rely on good network performance for interconnecting the distributed system components, HPC makes the greatest demands on the network by far. This is due to the fact that a Grid version of an HPC application may involve intense interprocess communications that are highly sensitive to the bandwidth and latency characteristics of the network. For these reasons, the remainder of this document will focus primarily on the characteristics of computational Grids for HPC.

1.8 Defining Grid Portal, Globus Toolkit & Web Services

The main terms that we will encounter in this thesis work are Grid Portal, Globus Toolkit& Web Services. A basic definition of these terms is included as a part of this sub-section to promote better understanding of the problem statement.

- The *Globus Toolkit* [7] is a community-based, open-architecture, open-source set of services and software libraries that support Grids and Grid applications. The toolkit addresses issues of security, information discovery, resource management, data management, communication, fault detection, and portability.
- A *Grid Portal* [16] is a web based application that commonly provides personalization, single sign on, content aggregation, and seamless access to Grid heterogeneous resources and services that support the end user in one or more tasks through a Web-based user interface.
- *Web service* [14] is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine processable format specifically WSDL. Web services standardize the messages that entities in a distributed system must exchange in order to perform various operations.

CHAPTER 2
PROBLEM STATEMENT

Grid Computing has evolved as a new infrastructure of the 21st century and is all set to revolutionize the technology trends. Grid computing allows uniting pools of servers, storage systems, and networks into a single large system that can deliver the power of multiple-systems resources (computing/processing power, data storage/networked file systems, communications and bandwidth, application software) to a single user point for a specific purpose.

A Grid Portal is a web-based application that provides personalization, single sign on, content aggregation from different sources and hosts the presentation layer of information systems. Thus a portal liberates the end-user from details of grid middleware programming and allows seamless access to applications and services running on remote resources.

The main thrust is on the development of a framework that allows applications to be installed and used as Web services accessed through a Grid portal.

We focus on two specific issues:

- How can a Grid application be captured as a Web service?
- Once we have a way to encapsulate and execute Grid applications as Grid services, how does one provide a way for an application developer to generate an application specific interface that can be provided to the end user through the portal?

CHAPTER 3
WEB SERVICES WITH GLOBUS TOOLKIT

Grid Computing is defined as the "set of middleware, libraries and tools that allow the cooperative use of geographically distributed resources unified to act as a single powerful platform for the execution of a range of parallel and distributed applications". So grid is a flexible, secure, coordinated resource sharing among dynamic collection of individuals, institutions and resources. We review two technologies: *Globus Toolkit4*, which has been widely adopted as a Grid technology solution for scientific and technical computing, and *Web services,* which have emerged as a popular standards-based framework for accessing network applications.

3.1 Globus Toolkit

In a grid, the middleware is used to hide the heterogeneous nature and provide users and applications with a homogeneous and seamless environment by providing a set of standardized interfaces to a variety of services. Globus Toolkit [21], a middleware technology, provides a software infrastructure that enables applications to handle distributed heterogeneous computing resources as a single virtual machine. It defines basic services and capabilities required to construct a computational grid. The Globus Toolkit is a community-based, open-architecture, open-source set of services and software libraries that support Grids and Grid applications. The toolkit addresses issues of security, information discovery, resource management, data management, communication, fault detection, and portability. We will work with the latest version of the toolkit i.e. GT4.

3.1.1 Exploring GT4

The Globus Toolkit4 is a software toolkit, developed by The Globus Alliance, which can be used to program grid-based applications. The toolkit includes a few high-level services that we can use to build Grid applications. Globus Toolkit includes a resource monitoring and discovery service, a job submission infrastructure, a security infrastructure, and data management services. It is a realization of the OGSA [7]

requirements and a de facto standard for the Grid community. Most of these services are implemented on top of WSRF [21]. The toolkit also includes some services that are not implemented on top of WSRF and are called the non-WS components. The Globus Toolkit includes a complete implementation of the WSRF specification.

GT4 primarily consists of five main components as shown in Figure 2.1 :-

- **Common Runtime:** The Common Runtime components provide a set of fundamental libraries and tools which are needed to build both WS and non-WS services.
- **Security:** Using the Security components, based on the Grid Security Infrastructure (GSI), we can make sure that our communications are secure.
- **Data management:** These components will allow us to manage large sets of data in our virtual organization.
- **Information services:** The Information Services, commonly referred to as the Monitoring and Discovery Services (MDS) [22], includes a set of components to discover and monitor resources in a virtual organization. GT4 also includes a non-WS version of MDS (MDS2) for legacy purposes.
- **Execution management:** Execution Management components deal with the initiation, monitoring, management, scheduling and coordination of executable programs, usually called jobs, in a Grid.

So, GT4 is a set of software components for building distributed systems: systems in which diverse and discrete software agents interact via message exchanges over a network to perform some tasks. Distributed systems face particular challenges relating to sometimes high and unpredictable network latencies, the possibility of partial failure, and issues of concurrency. In addition, system components may be located within distinct administrative domains, thus introducing issues of decentralized control and negotiation. Thanks to GT4 as it renders solution to above problems.

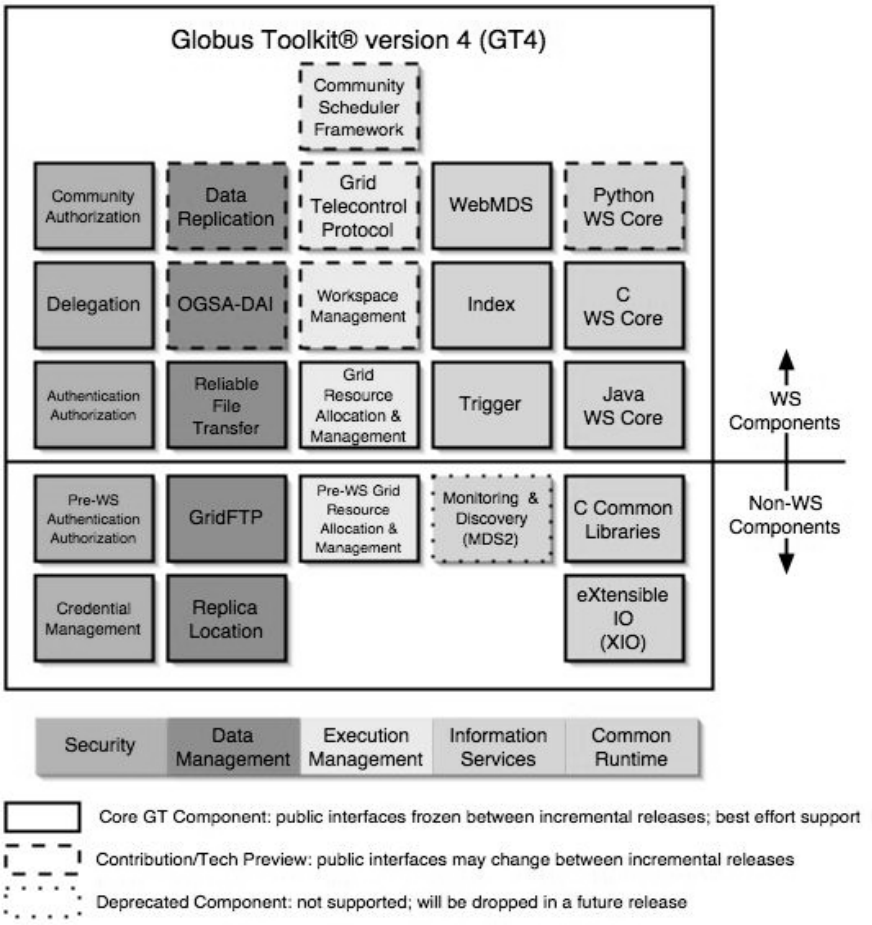


Figure 3.1: GT4 Components [21]

3.1.2 Predefined GT4 Services

GT4 provides a set of predefined services. Nine GT4 services implement Web services interfaces: Job management (GRAM); Reliable file transfer (RFT); Delegation; MDS-Index, MDS-Trigger, and MDSArchive [21-23] (collectively termed the Monitoring and Discovery System, or MDS); Community authorization (CAS); DAI data access and integration; and GTCP Grid TeleControl Protocol for online control of instrumentation.

For two of these services, GRAM and MDS-Index, pre-WS legacy implementations are also provided. For three additional GT4 services, WS interfaces are not yet provided. They are GridFTP data transport, replica location service, and My Proxy online credential repository. Other libraries provide powerful authentication and

authorization mechanisms, while the eXtensible I/O (XIO) library provides convenient access to a variety of underlying transport protocols. SimpleCA is a lightweight certification authority.

3.1.3 GT4 Architecture Overview

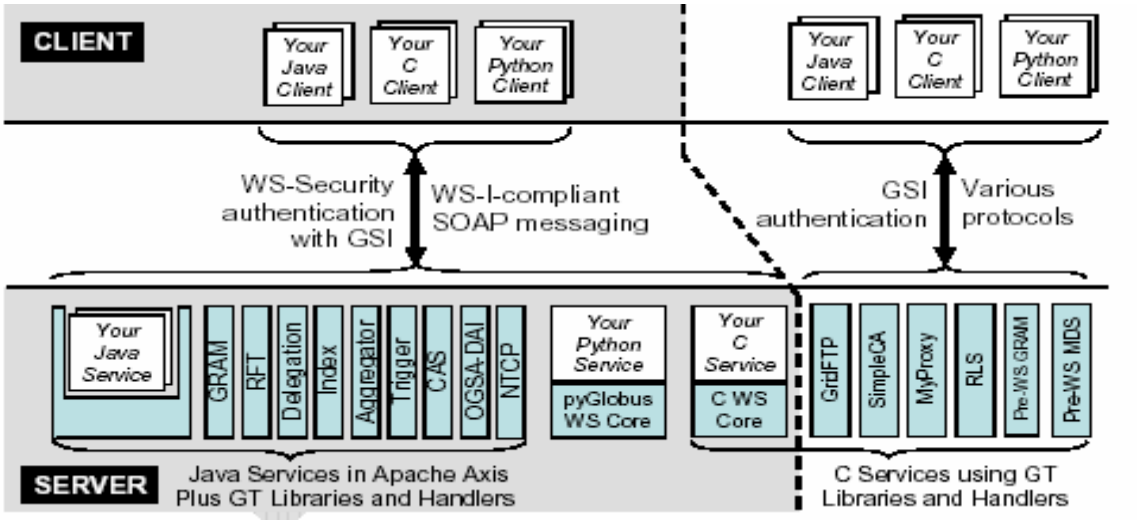


Figure 3.2: Schematic View of GT4 Components [21]

As shown in Figure 4, GT4 comprises both a set of service implementations i.e. server code and associated client libraries. GT4 provides both Web services components and non-WS components. All boxes in the client domain denote custom applications and/or third-party tools that access GT4 services or GT4-enabled services. All GT4 WS components use WS-Interoperability-compliant transport and security mechanisms, and can thus interoperate with each other and with other WS components. In addition, all GT4 components, both WS and non-WS, support X.509 end entity certificates and proxy certificates. Thus a client can use the same credentials to authenticate with any GT4 WS [15] or non-WS component.

3.1.4 GT4 Container

The GT4 source code includes implementations of a set of Web services specifications important for Grid applications. Some of these specifications are

standards such as WSRF, WS-Notification while others are currently unique to Globus such as GRAM, RFT. These implementations can be combined with other components like Web servers, SOAP engines, implementations of other Web services specifications, etc. to produce a variety of different GT4 containers.

A GT4 container implements SOAP [14] over HTTP as a message transport protocol, and both transport-level and WS-Security message-level security for all communications; implements WS-Addressing, WSRF, and WS-Notification functionality; supports logging via Log4j which implements the Jakarta Commons Logging API; defines WSRF WS-Resources with properties providing access to information about services deployed in the container and container properties such as version and start time.

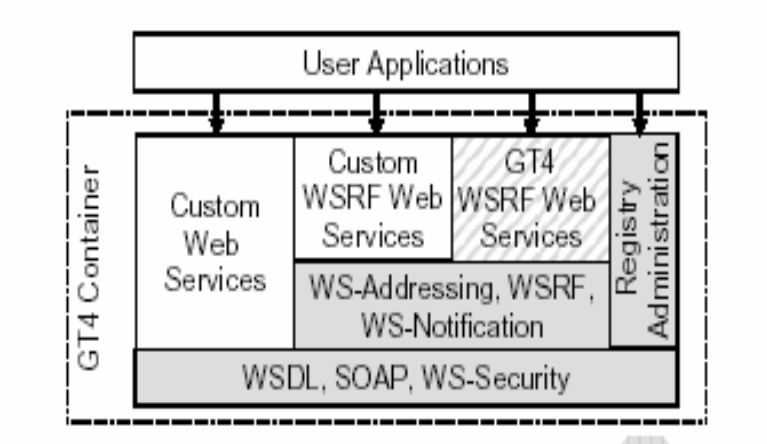


Figure 3.3: GT4 Container [23]

Thus, any GT4 container can:

- Host services whose client interfaces are defined in terms of only basic WS specifications and are called custom Web services and
- Host services whose client interfaces make use of WSRF and related mechanisms and are called custom WSRF Web services.

If appropriate software is installed, then a GT4 Java container can also host advanced services provided by GT4, such as GRAM, MDS, and RFT referred to as GT4 WSRF Web services. Regardless of what services are deployed, application clients can use GT4 container registry interfaces to determine which services are hosted in a particular GT4 container, and GT4 container administration interfaces to perform

basic administration functions. All GT4 container interfaces are compliant with the WS-I Basic Profile, and can be configured to be compliant with the WS-I Basic Security Profile.

The GT4 distribution includes software and instructions to construct GT4 containers as in Figure 6 for three different Web service implementation languages (Java, C, and Python). They are discussed below:

- **GT4 Java Containers:** GT4 support for Web services implemented in Java comprises the GT4 Java WS Core code, which implements WSRF and WS-Notification as well as supporting code for security and management. This code is designed to use with Apache Axis as a SOAP engine plus other relevant Apache components such as the WS-Addressing and WS-Security implementations. To produce a complete GT4 Java container, we can host the GT4 Java WS Core + Axis [14] combination either in a "simple Java container" provided by GT4 or alternatively in a more featureful but complex servlet container such as Tomcat. The former approach provides for easier installation and administration and is recommended unless a site or resource already runs Tomcat for other purposes. Such a GT4 Java container can also host various GT4 services implemented in Java, such as GRAM, RFT, MDS-Index, MDS-Trigger, and MDS-Archive.

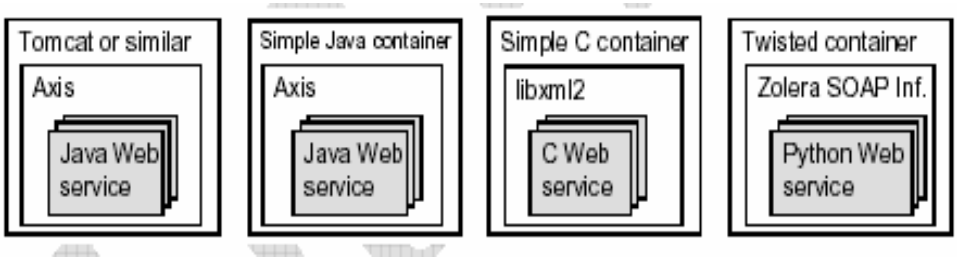


Figure 3.4: Different GT4 container configurations [22]

- **GT4 C Container:** GT4 support for Web services written in C comprises the GT4 C WS Core, which implements WSRF and WS-Notification as well as supporting code for security and management.
- **Python Web services:** GT4 support for Web services written in Python comprises the GT4 Python WS Core, which implements WSRF and WS-Notification as well as

supporting code for security and management. To produce a complete GT4 Java container, we combine this code with the Zolera SOAP Infrastructure and Twisted container.

3.2 Web Services

3.2.1 Defining Web services

GT4 is a set of software components that implement Web services [11-14] mechanisms for building distributed systems. Web services provide a standard means of interoperating between different software applications running on a variety of platforms and/or frameworks.

According to W3C [11], a Web service is: "A software system designed to support interoperable machine-to-machine interaction over a network". It has an interface described in a machine processable format specifically WSDL. Web services standardize the messages that entities in a distributed system must exchange in order to perform various operations. At the lowest level, this standardization concerns the protocol used to transport messages typically HTTP, message encoding using SOAP, and interface description with WSDL. A client interacts with a Web service by sending it SOAP message; the client may subsequently receive response message(s) in reply. At higher levels, other specifications define conventions for securing message exchanges such as WS-Security, for management such as WSDM and for higher-level functions such as discovery and choreography. Figure 7 presents a view of these different component technologies.

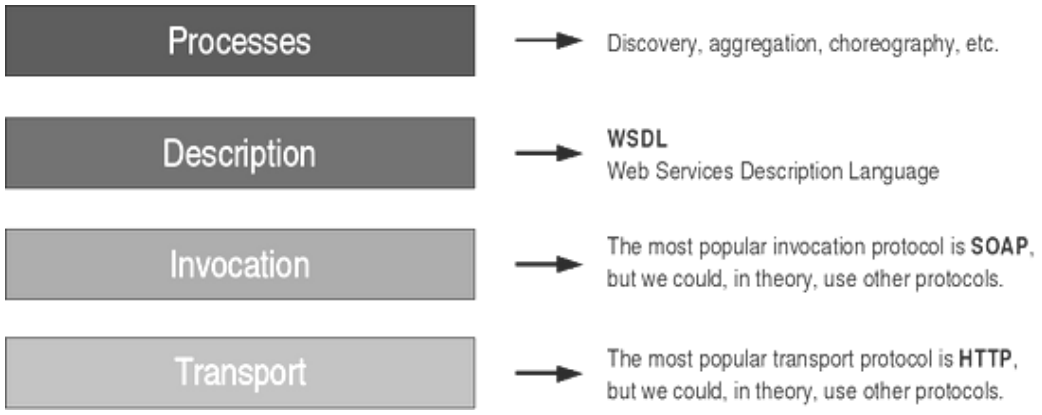


Figure 3.5: Web Services architecture [23]

- **Service Processes**: This part of the architecture generally involves more than one Web service. For example, discovery belongs in this part of the architecture, since it allows us to locate one particular service from among a collection of Web services.

- **Service Description**: One of the most interesting features of Web Services is that they are self-describing. This means that, once you've located a Web Service, you can ask it to 'describe itself' and tell you what operations it supports and how to invoke it. This is handled by the Web Services Description Language (WSDL).

- **Service Invocation**: Invoking a Web Service involves passing messages between the client and the server. SOAP (Simple Object Access Protocol) specifies how we should format requests to the server, and how the server should format its responses. In theory, we could use other service invocation languages such as XML-RPC, or even some XML language. However, SOAP is the most popular choice for Web Services.

- **Transport**: Finally, all these messages must be transmitted somehow between the server and the client. The protocol of choice for this part of the architecture is HTTP (HyperText Transfer Protocol), the same protocol used to access conventional web pages on the Internet.

3.2.2 Web Services Implementation

From the client perspective, a Web service is simply a network-accessible entity that processes SOAP messages. Things are somewhat more complex under the covers. To simplify service implementation, it is common for a Web services implementation to distinguish between:

- The hosting environment or container, contains the domain-independent logic used to receive a SOAP message and identify and invoke the appropriate code to handle the message, and potentially also to provide related administration functions, and:

- The Web service implementation, describes the domain-specific code that handles the message. This separation of concerns means that the developer need only provide the domain-specific message handling code to implement a new service.

We can further partition the hosting environment logic into that concerned with transporting the SOAP message typically via HTTP, thus an HTTP engine or Web server— sometimes termed an application server and that concerned with processing SOAP messages the SOAP [11-14] engine of SOAP processor. Figure 3 illustrates these various components.

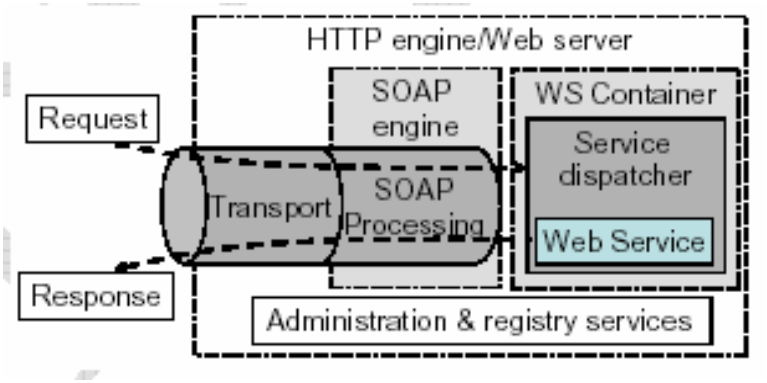


Figure 3.6: Web Services Implementation [11]

3.2.3 Web Services Specifications in GT4

We provide pointers to the Web services specifications that underlie GT4. These comprise the core specifications that define the Web services architecture (XML, SOAP, WSDL), WS-Security, WS-Addressing, WSRF, and WS-Notification.

3.2.3.1 XML, SOAP, WSDL

XML is used extensively within Web services as a standard, flexible, and extensible data format. SOAP 1.2 provides a standard, extensible, composable framework for packaging and exchanging XML messages between a service provider and a service requestor. SOAP is independent of the underlying transport protocol, but is most commonly carried on HTTP. WSDL 1.1[14] is an XML document for describing Web services. Standardized binding conventions define how to use WSDL in conjunction with SOAP and other messaging substrates.

3.2.3.2 WS-Security and Friends

The WS-Security family of specifications addresses a range of issues relating to authentication, authorization, policy representation, and trust negotiation in a Web services context. GT4 uses a number of these specifications, notably Security Authorization Markup Language (SAML), to address message protection, authentication, delegation, and authorization, as follows:

- TLS (transport-level) or WS-Security and WS-Secure Conversation (message level) are used as message protection mechanisms in combination with SOAP.
- X.509 End Entity Certificates or Username and Password are used as authentication credentials.
- X.509 Proxy Certificates and WS-Trust [13] are used for delegation.
- SAML assertions are used for authorization.

3.2.3.3 WS-Addressing, WSRF, and WS-Notification

A number of specifications are used to represent and manipulate stateful entities such as physical resources of various kinds, logical components such as software licenses, and transient activities such as tasks and workflows. The WS-Addressing specification defines transport-neutral mechanisms to address Web services and messages. The WS Resource Framework (WSRF) specifications define a generic and open framework for modeling and accessing stateful resources using Web services. This framework comprises mechanisms to describe views on the state (WS-Resource Properties), to support management of the state through properties associated with the Web service (WSResourceLifetime), to describe how these mechanisms are extensible to groups of Web services (WS-Service Group) [18], and to deal with faults (WS-Base Faults). The WS-Notification family of specifications defines a pattern-based approach to allowing Web services to disseminate information to one another. This framework comprises mechanisms for basic notification (WS-Notification), topic-based notification (WS-Topics), and brokered notification (WS-Brokered Notification).

3.3.3.4 Other Relevant Specifications

GT4 includes an implementation of the GridFTP extensions to FTP recently finalized within GGF and the DAIS implementation of the GGF OGSA-DAI specification under development within GGF. The WS-Interoperability (WS-I) organization has produced a number of profiles that define ways in which existing Web services specifications can be used to promote interoperability among different implementations. The WS-I [13-14] Basic Profile speaks to messaging and service description: primarily XML, SOAP, and WSDL. The WS-I Basic Security Profile speaks to basic security mechanisms. Web services distributed management (WSDM) specifications under development within OASIS are likely to play a role in future GT implementations as a means of managing GT components. WS-CIM specifications under development within DMTF are likely to play a role in future GT implementations as a means of representing physical and virtual resources. The Global Grid Forum's Open Grid Services Architecture (OGSA) working group has completed a document that provides a high-level description of the functionality required for future service-oriented infrastructure and applications, and a framework that suggests how this functionality can be factored into distinct specifications.

3.3 Web Services – Why or Why Not?

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 you 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 your 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.). Fortunately, there are a lot of emerging Web services specifications (including WSRF) that are helping to make Web services more and more versatile.

3.4 WSRF –It's All About State

Web Services are the technology for Internet-based applications with loosely coupled clients and servers. That makes them the natural choice for building grid-based applications. However, Web Services do have certain limitations. Plain Web services are usually stateless. This means that the Web service cannot remember information, or keep state, from one invocation to another. However, Grid applications do generally require statefulness. So we introduce WSRF [12-13] (Web Services Resource Framework), which improves several aspects of web services to make them more adequate for grid applications.

3.4.1 The resource approach to Statefulness

Giving Web services the ability to keep state information while still keeping them stateless is a complex problem. Fortunately, it's a problem with a very simple solution: simply keep the Web service and the state information completely separate. Instead of putting the state in the Web service we will keep it in a separate entity called a resource, which will store all the state information. Each resource will have a unique key, so whenever we want a stateful interaction with a Web service we simply have to instruct the Web service to use a particular resource. A pairing of a Web

service with a resource is called a WS-Resource [12]. The address of a particular WS-Resource is called an endpoint reference.

3.4.2 The WSRF Specification

The Web Services Resources Framework is a collection of five different specifications. They all relate in some way or another to the management of WS-Resources.

3.4.2.1 WS-ResourceProperties

A resource is composed of zero or more resource properties. For example, in the figure shown above each resource has three resource properties: Filename, Size, and Descriptors. WS-ResourceProperties [17] specifies how resource properties are defined and accessed.

3.4.2.2 WS-ResourceLifetime

Resources have non-trivial lifecycles. In other words, they are not a static entity that is created when our server starts and destroyed when our server stops. Resources can be created and destroyed at any time. The WS-ResourceLifetime supplies some basic mechanisms to manage the lifecycle of our resources.

3.4.2.3 WS-ServiceGroup

We will often be interested in managing groups of Web Services [12-14] or groups of WS-Resources, and performing operations such as add new service to group, remove this service from group, and more importantly find a service in the group that meets a condition. The WS-ServiceGroup specifies how exactly we should go about grouping services or WS-Resources together. Although the functionality provided by this specification is very basic, it is nonetheless the base of more powerful discovery services such as GT4's IndexService which allow us to group different services together and access them through a single point of entry i.e. the service group.

3.4.2.4 WS-BaseFaults

Finally, this specification aims to provide a standard way of reporting faults when something goes wrong during a WS-Service invocation.

3.4.2.5 WS-Notification

WS-Notification is another collection of specifications that, although not a part of WSRF, is closely related to it. This specification allows a Web service to be configured as a notification producer, and certain clients to be notification consumers or subscribers.This means that if a change occurs in the Web service or in one of the WS-Resources, that change is notified to all the subscribers .

3.4.2.6 WS-Addressing

The WS-Addressing specification provides us a mechanism to address Web services which is much more versatile than plain URIs. In particular, we can use WS-Addressing to address a Web service + resource pair (a WS-Resource).

3.4.3 Unifying GT4 and Web Services

We have already discussed the terms GT4, Web services abd WSRF. But another important acronym-OGSA [23-24] is yet to be covered. It is the heart of the GT4. A grid application will usually consist of several different components. For example, a typical grid application could have:

- **VO Management Service**: To manage what nodes and users are part of each Virtual organization.
- **Resource Discovery and Management Service:** So applications on the grid can discover resources that suit their needs, and then manage them.
- **Job Management Service**: So users can submit tasks in the form of jobs to the grid.
- And a whole other bunch of services like security, data management, etc.

Furthermore, all these services are interacting constantly. For example, the Job Management Service might consult the Resource Discovery Service to find computational resources that match the job's requirements. With so many services, and so many interactions between them, there exists the potential for chaos. What if every vendor out there decided to implement a Job Management Service in a

completely different way, exposing not only different functionality but also different interfaces? It would be very difficult to get all the different software pieces to work together.

The solution is standardization*:* define a common interface for each type of service. For example, take a look at the World Wide Web. One of the reasons why the Web is such a popular Internet application is because it is based on standards (HTML, HTTP, etc.) agreed upon by all the different major players Microsoft, Netscape, etc. On the other hand, that you could only use a Microsoft browser to access websites implemented with Microsoft technology. It would be definitely uncool. Thanks to standards, I can use my favorite browser provided it follows standards, which most modern browsers do to access most of the websites out there regardless of what technology is used to implement the website. Why? Because a set of common languages was agreed upon for all the browsers and websites out there. Standardization is definitely a good thing.

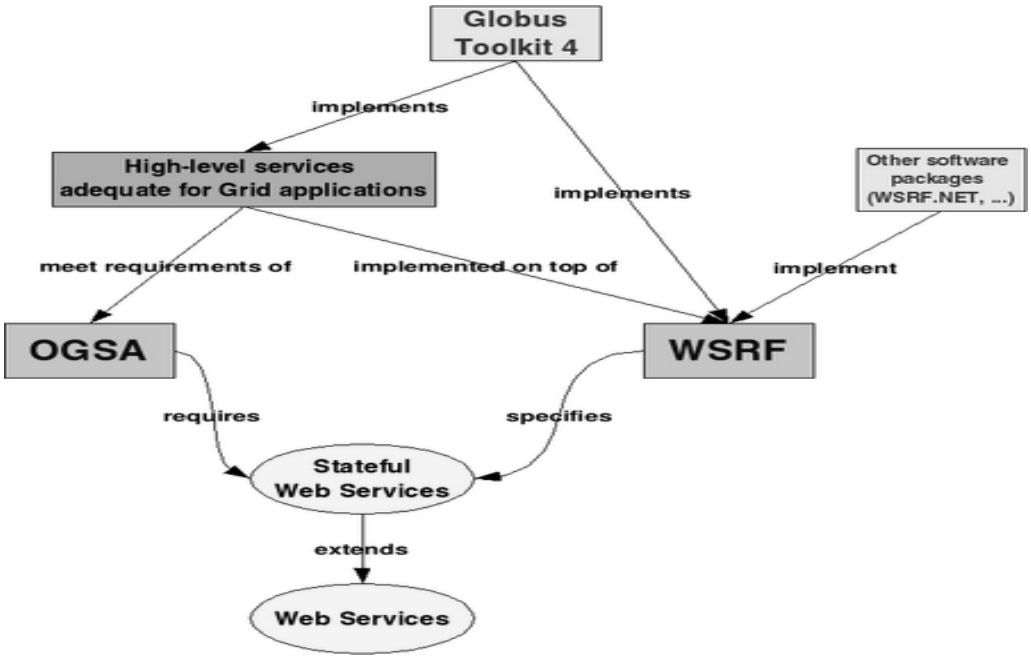


Figure 3.7: Relationship between GT4, WSRF, and Web Services [19]

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 commonly finds in a grid application (job management services, resource management services, security services, etc.) by specifying a set of standard interfaces for these services.

However, when the task of creating this new architecture was undertaken, the developers realized they needed to choose some sort of distributed middleware on which the architecture can be based. If OGSA, for example, defines that the JobSubmissionInterface 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. However, although the Web Services Architecture was the best option, it still didn't meet one of OGSA's most important requirements: the underlying middleware had to be stateful. It was then WSRF came to picture to provide Stateful services to OGSA.

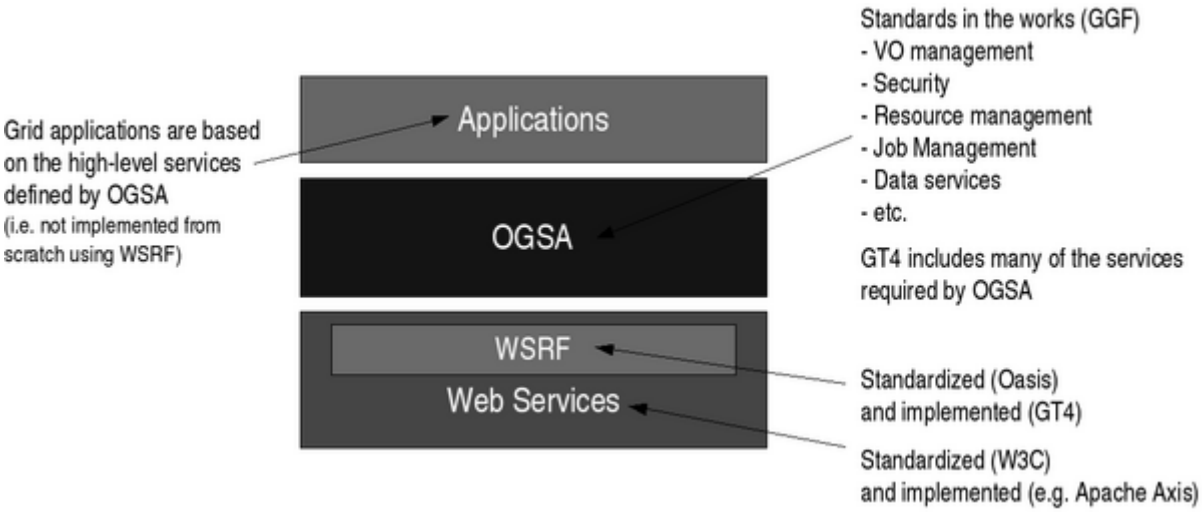


Figure 3.8: Layered diagram of OGSA, GT4, WSRF, and Web Services [22]

WSRF, a specification developed by OASIS (Organization for the Advancement of Structured Information Standards), specifies how we can make our Web Services

stateful. So what exactly is the relation between OGSA and WSRF? WSRF provides the stateful services that OGSA needs. In the diagram below WSRF specifies stateful services as opposed to those services simply being required by OGSA. Another way of expressing this relation is that, while OGSA is the architecture, WSRF is the infrastructure on which that architecture is built on.

CHAPTER 4

GRID PORTALS

4.1 Defining a Grid Portal

Portals are used to provide people with access to information and applications in a condensed form. A portal is a base that acts as a starting point to help the navigation through huge amounts of information on the World Wide Web such as Yahoo. A web portal is different from a web search engine, such as Google. Search engines tell us where the information is, it does not aggregate it. Instead, a web portal aggregates information and provide index services.

A Grid portal [15-16] is a user's point of access to a Grid system. It provides an environment where the user can access Grid resources and services, execute and monitor Grid applications, and collaborate with other users. Therefore we can define a grid portal as:

"A Grid portal is a web based application that commonly provides personalization, single sign on, content aggregation, and seamless access to Grid heterogeneous resources and services that support the end user in one or more tasks through a Web-based user interface."

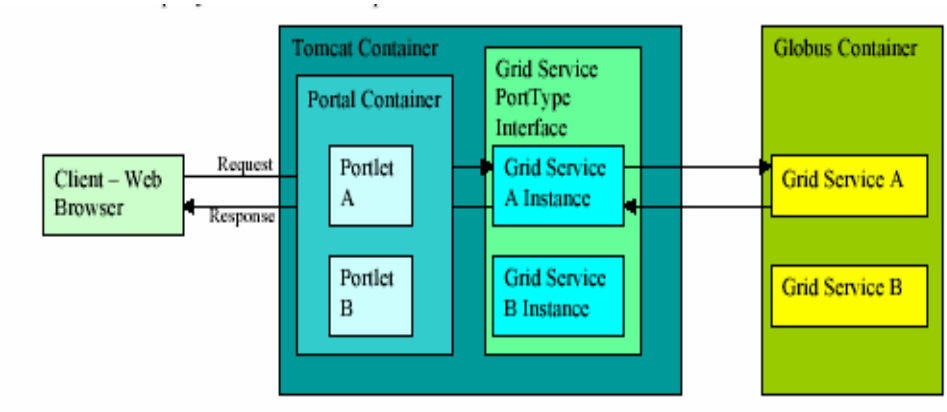


Figure 4.1: Grid Portal [23]

Grid Portals are necessary for two reasons.

- The Grid environment is very complex including scheduling, co-allocation, security, quality of service, data transfer, network protocol, Grid programming model, etc.
- Computational science environment is also complex. It's cumbersome for users to access a variety of grid resources in single application and follow the changes of interface, operation system and Grid tools. As a result, users may get overwhelmed in one single scientific application.

Thus, a Grid Portal is defined to be a web based application server enhanced with the necessary software to communicate with Grid services and resources. Grid Portals build upon the familiar Web portal model to deliver the benefits of Grid computing to virtual communities of users, providing a single access point to Grid services and resources. The major difference is that a Grid Portal is for Grid resources rather than classified web pages/sites.

4.1.1 Portlet

Putting forth a portlet simply, we can say that a portlet is a mini-application that resides in the portal. In other words, a portlet provides a mini-window within a portal page. Multiple portlets can be composed in a portal page. The portal containers contain Portlets. A portlet [23-24] container runs portlets and provides them with the required runtime environment It manages the lifecycle of the portlet. It provides persistent storage for portlet preference. Hence, we can say that:

"Portlets are the pluggable user-interface components, to provide a presentation layer to information systems."

4.2 Grid Portal Services

Generic services that are included in a portal are at a higher level than atomic web or grid services. Examples might include:

- Session Management
- Logon/ Authentication
- Policy-Based Authorization
- Shopping Cart
- Query And Result

- Job List And Status
- Query-Based Resource Discovery
- Query-Based Application Selection
- Problem Specification
- Job Composition
- Service Registry Lookup Service
- Service Deployment
- Lifecycle Management
- Workflow Selection And Enactment
- Resource Scheduling
- Job Submission, Execution, Monitoring, Profiling, Interaction
- Event Notification Services

4.3 Benefits of Grid Portals

- Grid Portal hides the complexity from users.
- Grid portal provides end users with a customized view of software and hardware resources specific to their particular problem domain.
- They also provide a single point of access to Grid-based resources and allow single source through which authorization can be checked/enforced.
- They allow scientists or engineers to focus on their problem area by making the grid a transparent extension of their desktop computing environment.
- Grid Portals facilitate the establishment of VOs, comprised of distributed researchers allowing them to collaborate and access heterogeneous resources and services more efficiently and seamlessly.

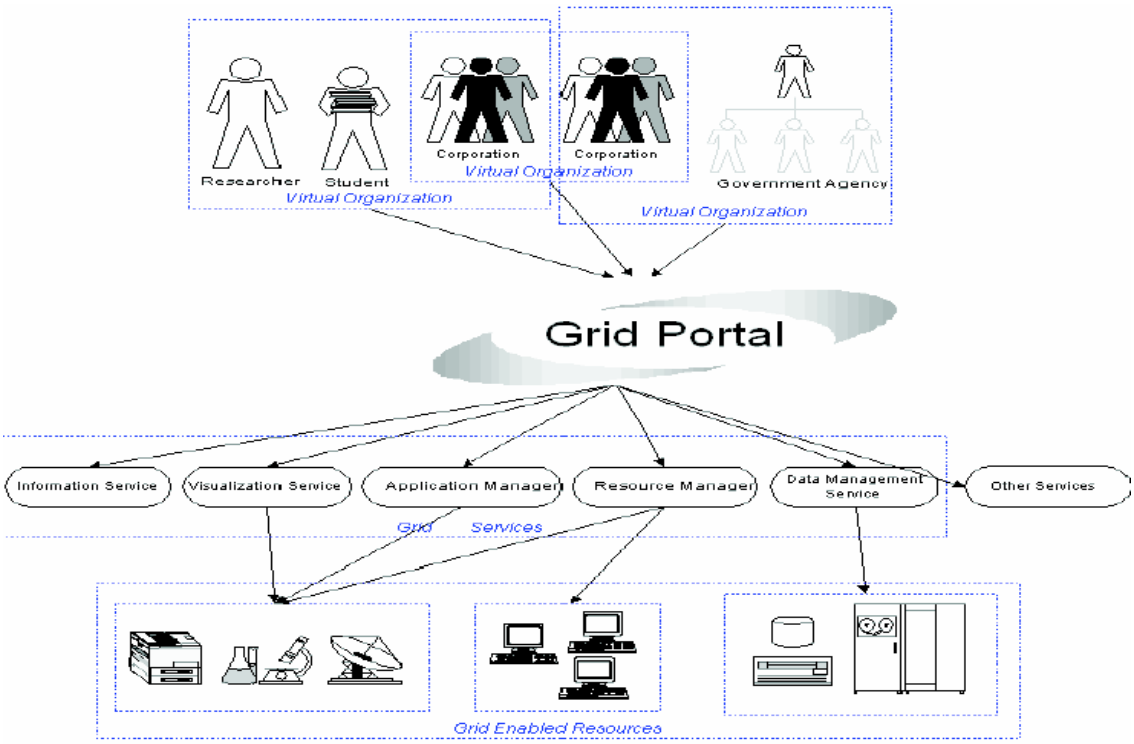


Figure 4.2: Grid Portal merging gap between users & services [24]

4.4 Grid Portal Design

Grid technology is designed to allow users seamless access to applications and services running on remote resources. There are several properties that distinguish these applications as Grid applications:

- It is very common for Grid applications to consist of a heterogeneous composition of several (and sometimes many) remote services, each of which is responsible for one part of the overall computation. Sometimes this service composition can be seen as sequences of operations that are scheduled over time as a workflow and other times they involve applications running at difference locations that directly interact with each other in a message-based dialog.
- These applications often involve a complex web of collaborations (Fig. 1). The end users are scientists and engineers that only want solutions to problems and they want to do this in language of their scientific domain. They interact with the application through web portal systems that record their choices for the application parameters. These application parameters are entered into Grid

execution workflow scripts that are authored by a second group of scientists who understand how to compose Grid services into distributed applications. Domain experts are the authors and maintainers of the basic service and tool components such as the simulation codes and data filters that compose the basic elements of the computation.

- The end user, who initiates or interacts with the application, sits at a remote location and will expect to authenticate his or her identity only once. From this initial authentication, many resources from many different administrative domains may be accessed. The Grid security services pass the users identity to the local security mechanisms for individual services. Authorization to use the remote resources and services is provided through authorization capabilities granted by the remote resource providers and managed by an authorization service.

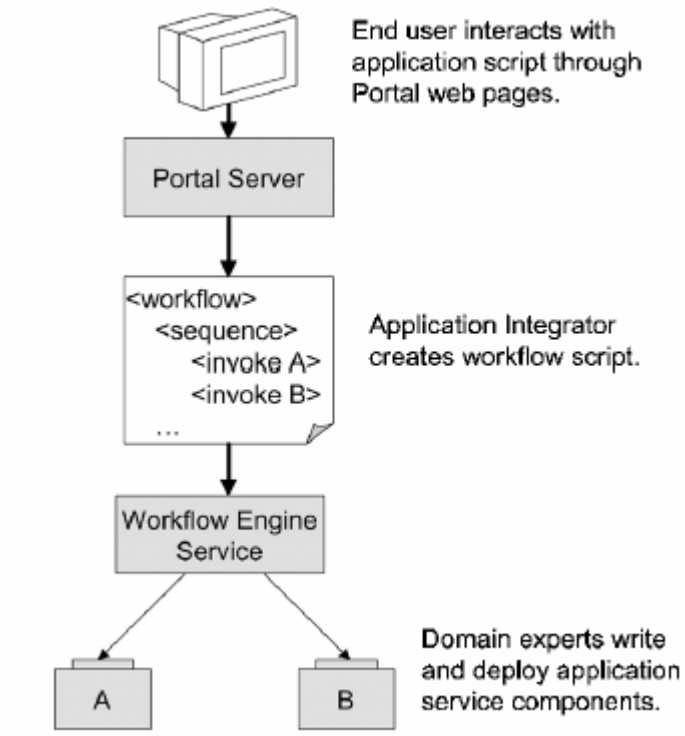


Figure 4.3: Three levels of Grid Portal Design [24]

4.5 Selecting a grid portal framework

In order to develop a grid portal we can either choose amongst on of the existing portal frameworks or develop a new one. A number of portal frameworks are available and are mainly open-source. So it is better to develop a portal using the existing portals and customize them according to our needs. We chose Gridsphere, which is of the most popular, open-source, and resilient frameworks known to exist.

4.5.1. What makes GridSphere different?

Although a number of portals exist: Jetspeed2, uPortal, StringBeans, Exo, Liferay, Jboss but what makes gridsphere [23] the choice for our grid portal are listed below:

- A handy template build system-using Ant: *ant new-project.*
- It is lightweight with no EJB (Enterprise Java Beans), based on popular, robust libraries such as Hibernate for persistence.
- It exhibits the ability to add support for new authentication schemes with pluggable auth modules descriptor.
- Visual UI tags and beans make presentation development much easier.
- It offers support for the Grid: GridPortlets offered as add-on webapp.

4.5.2 Gridsphere

Gridsphere is 100% JSR 168 compliant [19-20] and an open-source portal framework. The Gridsphere portal framework provides an open-source portlet based web portal. GridSphere enables developers to quickly develop and package portlet web applications that can be run and administered within the GridSphere portlet container.

The biggest asset of Gridsphere portal framework is the significant volume of documentation that is provided with it. The support comes in the form of GridSphere PortletReference Guide, GridSphere Tutorials, GridSphere User's Guide, and GridSphereAdministrator'sGuide.

Major features of GridSphere include:

- Portlet API implementation nearly fully compatible with IBM's WebSphere 4.2.
- Support for the easy development and integration of third-party portlets.
- Higher-level model for building complex portlets using visual beans and the GridSphere User Interface (UI) tag library.
- Flexible XML based portal presentation description can be easily modified to create customized portal layouts.
- Built-in support for Role Based Access Control (RBAC). Enables managing of access for guests, users, admins and super users.
- Sophisticated portlet service model that allows for creation of user services where service methods can be limited according to user rights.
- Persistence of data provided using Castor JDO from ExoLab for RDMS database support, SQL and OQL.
- Integrated Junit and Cactus unit tests for complete server side testing of portlet services including the generation of test reports.
- Documentation uses DocBook for HTML and PDF output of guides and tutorials.
- GridSphere core portlets offer base functionality including login, logout, user and access control management.
- Localization support in the Portlet API implementation and GridSphere core portlets support English, German, Czech, Polish, Hungarian and Greek.
- Open-source and 100% free!

The core and basic services provided in GridSphere are:

- Portlet Manager Service – It provides lifecycle methods to allow portlets to be installed, removed, initialized and destroyed by authorized users.
- Login Service – It allows a user to be retrieved from a username and password.
- User Manager Service – It allows to add/remove user accounts and edit user profiles.
- Access Control Service – It allows to add/remove user groups and add/remove user roles.
- Credential Manager Service – It allows to add/remove allowed user credentials and configure use of Credential Retrieval Service.

- Job Manager Service – It is used for listing, starting, migrating, stopping jobs.
- Job Monitoring Service –It is used to specify what to monitor for any given job and archive related information.
- File Transfer Service - It is used for managing and scheduling file transfers.
- Data Manager Service – It helps to access data replica catalogues and describe data with meta-data.

CHAPTER 5
IMPLEMETATION DETAILS & EXPERIMENTAL RESULTS

5.1 Developing GT4 Web Service

Writing and deploying a WSRF Web Service is easy. We have to follow five simple steps.

- Define the service's interface: This is done with WSDL.
- Implement the service: This is done with Java.
- Define the deployment parameters: This is done with WSDD and JNDI.
- Compile everything and generate a GAR file: This is done with Ant.
- Deploy service: This is also done with a GT4 tool.

5.1.1 Defining the interface in WSDL

The first step in writing a web service including those that use WSRF to keep state is to define the service interface. We need to specify what our service is going to provide to the outer world. We are not concerned with the inner workings of the service such as what algorithms it uses, other systems it interacts with, etc. We just need to know what operations will be available to users. In Web Services terminology, the service interface is called the port type usually written portType. The WSDL for our HelloWorld service is named *HelloWorld.wsdl*. The WSDL file has three features that are specific to WSRF or more specifically the Globus implementation of WSRF we are using. The following is a brief overview of these three features:

- **Resource properties**: We use the wsrp:ResourceProperties attribute of the portType element to specify what our service's resource properties are. The resource properties must be declared in the <types> section of the WSDL file. The resource properties are where we will keep all state information.
- **The WSDL Preprocessor**: In the wsdlpp:extends attribute of the portType element, we can include existing WSRF portTypes in our own portType without having to

copy-and-paste from the official WSRF WSDL files. A WSDL Preprocessor will use the value of that attribute to generate correct WSDL which includes our own portType definitions plus any WSRF portType we might need in our service. This is a Globus-specific feature that is included to make life easier for programmers.

- **No bindings**: Bindings are an essential part of a normal WSDL file. However, we do not have to add them manually, since they are generated automatically by a GT4 tool that is called when we build the service.

An important feature of WSDL is that it is language-neutral. There is no mention of the language in which the service is going to be implemented, or of the language in which the client is going to be implemented. However, we will need to refer to the service interface from a specific language such as Java. For a tool to successfully generate the stub classes, we need to tell it where i.e. in what Java package to place the stub classes. We do this with a mappings file, which maps WSDL namespaces to Java packages. We name this file HelloWorldNamespaces.java.

5.1.2 Implementing the service in Java

After defining the service interface i.e what the service does, the next step is implementing that interface. The implementation is "how the service does what it says it does".

5.1.2.1 The QNames interface

The QNames Interface is a very simple Java interface that will make coding the service a bit easier. When we have to refer to anything related to a service, we will need to do it using its qualified name, or QName for short. This is a name which includes a namespace and a local name. For example, the QName of the Value RP is:

{http://www.globus.org/namespaces/examples/core/HelloWorldService_instance}Value. The file is named HelloWorldNamespaces.java.

5.1.2.2 Writing the service in Java

In this step, we write down the service code in java and name the file HelloWorldService.java.

5.1.3 Configuring the deployment in WSDD and JNDI

We have written the two most important parts of our stateful Web service: the service interface inWSDL and the service implementation in Java. In this step we will address another aspect, that is, how do we actually make our web service available to client connections? To realize this we will actually take the service and it's WSDL we have written and make them available through a Web services container. This step is called the deployment of the web service.

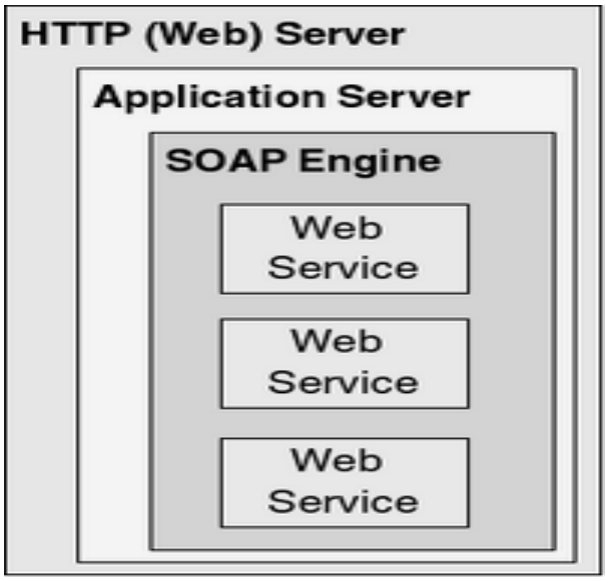


Figure 5.1: Deploying Web service

5.1.3.1 The WSDD deployment descriptor

One of the key components of the deployment phase is a file called the deployment descriptor. It is the file that tells the Web Services container how it should publish the web service for instance telling it what the our service's URI will be. The deployment descriptor is written in WSDD (Web Service Deployment Descriptor) format. The deployment descriptor file is named *deploy-server.wsdd*.

5.1.3.2 The JNDI deployment file

The Java Naming and Directory Interface (JNDI) is a standard extension to the Java platform, providing Java-enabled applications with a unified interface to multiple naming and directory services in the enterprise. The JNDI deployment file is called *deploy-jndi-config.xml*.

5.1.4 Creating GAR file with Ant

At the end of the three steps, we have a service interface in WSDL, a service implementation in Java, and a deployment descriptor in WSDD and JNDI telling the Web Services container how to present the service to the outer world. However, all this is a bunch of loose files. Next we have to place them into a Web services container. Using the three files we wrote in the previous three steps we will generate a Grid Archive, or GAR file. This GAR file is a single file which contains all the files and information the Web services container needs to deploy our service and make it available to the whole world. However, creating a GAR file is a complex task which involves the following:

- Processing the WSDL file to add missing pieces such as bindings.
- 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.

But we need not perform all the above steps as they will be done using a tool called Ant.

5.1.4.1 Apache Ant

Ant, an Apache Software Foundation project, is a Java build tool. 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 buildfile. This buildfile directs Ant on what it should compile, how it should compile it, and in what order. This simplifies the whole process considerably. In fact, it reduces the number of steps to one. With Ant, we simply have to write the service interface, the service implementation, and the deployment descriptor. Ant takes care of the rest.

As shown in the figure below, Ant generates the GAR directly from the three sets of source files. Internally, it is carrying out all the steps. In a GT4 project, Ant uses two sets of buildfiles: a couple of buildfiles, which are a part of GT4, and a buildfile, we

will have to write on our own. The GT4 buildfiles cover all the important steps generating the WSDL code, generating the stubs, etc. Our build file essentially has all the unique parameters of our web service, and a bunch of calls to the GT4 buildfiles.

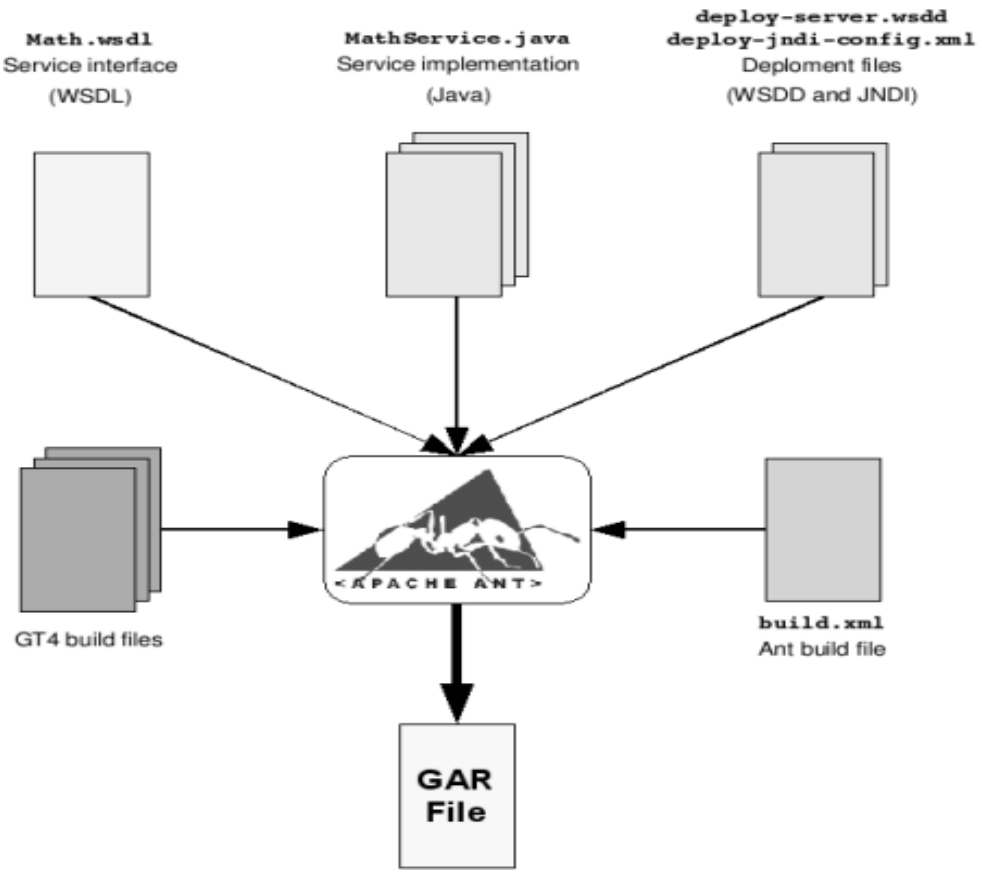


Figure 5.2: Generating GAR file with Ant

5.1.4.2 The globus-build-service script and buildfile

We will use the globus-build-service script and buildfile, one of the tools developed as part of the Globus Service Build Tools (GSBT) project. This tool will allow us to create a GAR file with minimal effort.

5.1.4.3 Creating the HelloWorldService GAR

Using the Ant buildfile, building a web service will proceed as following:

./globus-build-service.py *-d <service base directory> -s <service's WSDL file>*

The service base directory is the directory where we placed the deploy-server.wsdd file, and where the Java files can be found i.e inside an impl directory. To build the first example we simply need to do the following:

```
C:\Tutorial\globus-build-service.py
-d org\globus\examples\services\core\first\
-s schema\examples\HelloWorldService_instance\HelloWorld.wsdl
```

globus-build-service.py also allows us to use a shorthand notation which is much easier and faster to use. For example, to build our service and generate its GAR file, we simply need to do the following:

C:\Tutorial\globus-build-service.py first

Finally at the end of this step the GAR file will be placed in TUTORIAL. In our case, the GAR file generated for this example will be the following:

C:\TUTORIAL\org_globus_examples_services_core_first.gar

5.1. 5 Deploying the service into a Web Services container

The GAR file contains all the files and information the web server needs to deploy the web service. Deployment is done with a GT4 tool that, using Ant, unpacks the GAR file and copies the files within (WSDL, compiled stubs, compiled implementation, WSDD) into key locations in the GT4 directory tree.On the command prompt, enter the following:

globus-deploy-gar C:\TUTORIAL\org_globus_examples_services_core_first.gar

A practical demonstration to realize a GT4 web service is below:

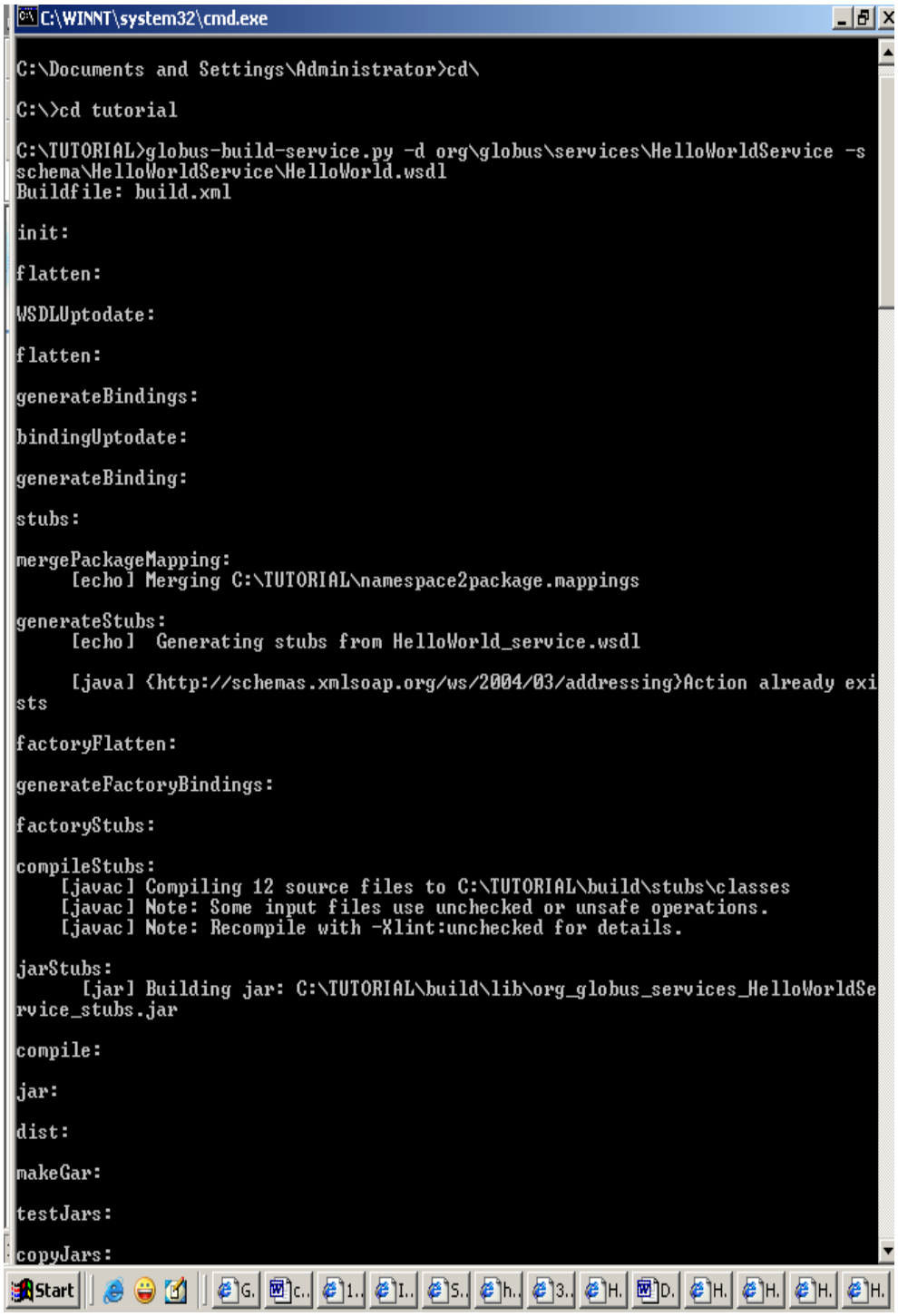


Figure 5.3: Creating GAR File

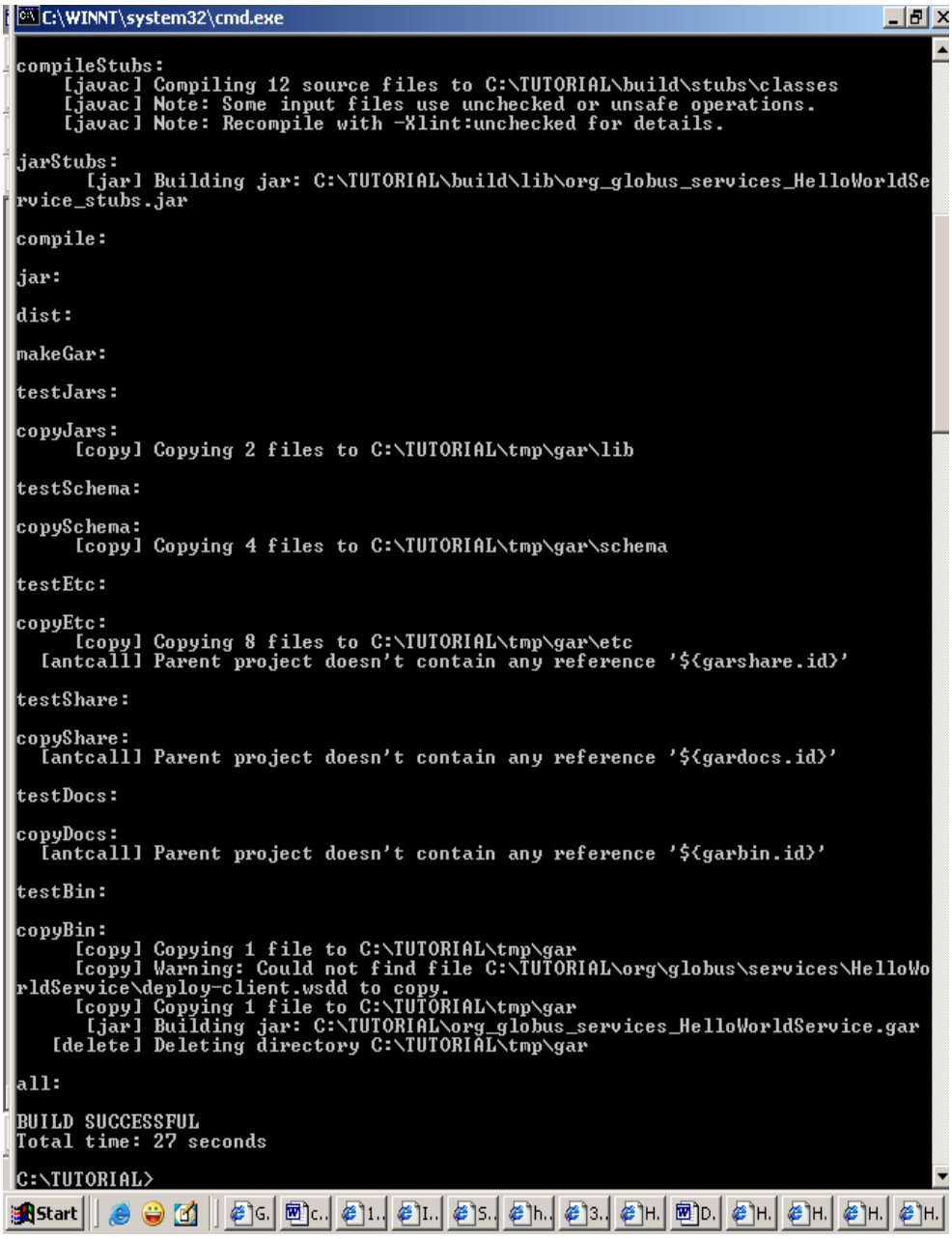


Figure 5.4: GAR File created successfully

Figure 5.5: Deployed service successfully

At the end of five steps, the service gets deployed into the GT4 Container. Running the command below starts the container:

globus-start-container –nosec

The service that we have deployed can be seen in the container below:

It is the second service with URI:

http://172.31.5.55:8080/wsrf/services/HelloWorldService

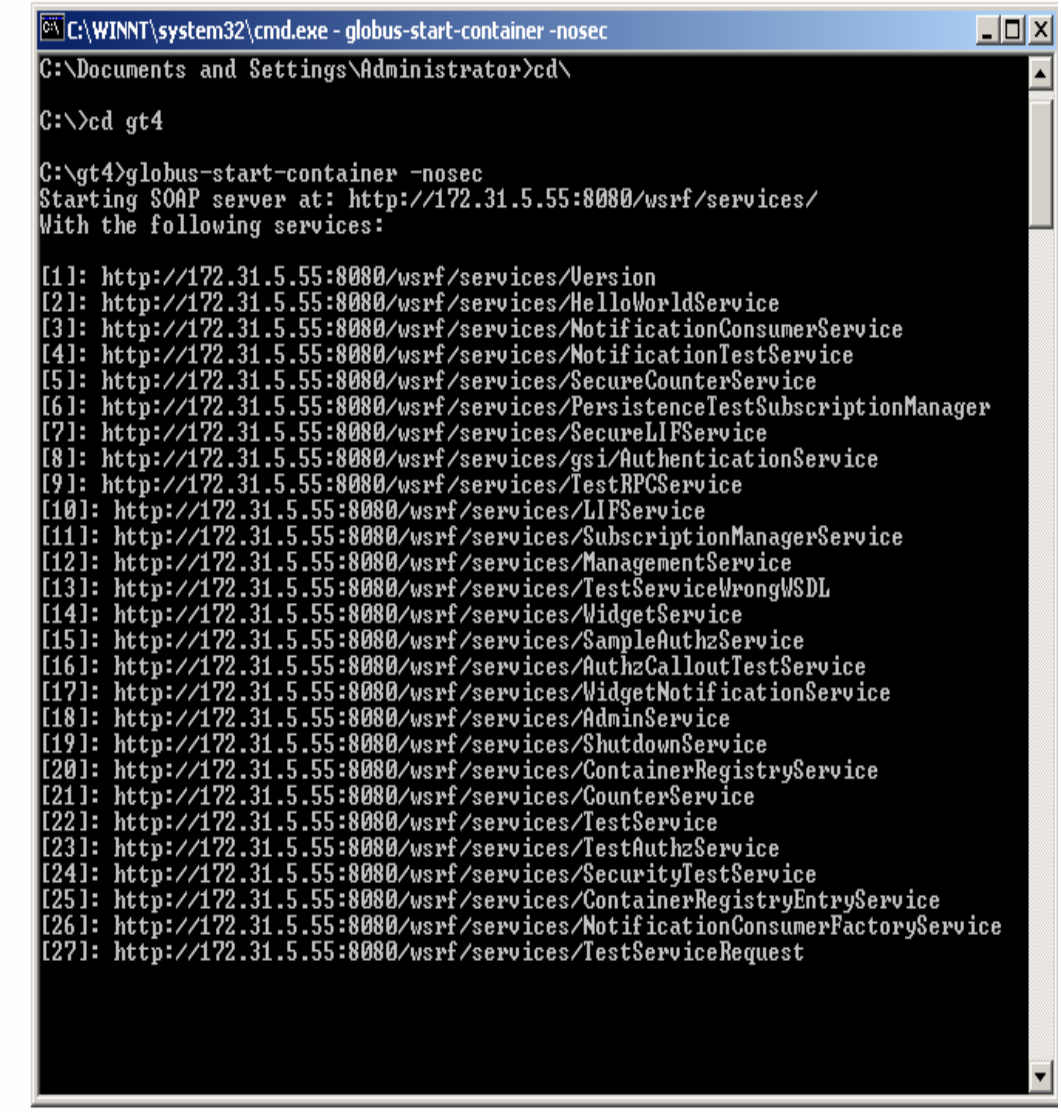


Figure 5.6: GT4 Container with deployed service

5.2 TIET Grid Portal

After deploying the *HelloWorldService* in the GT4 container, we move ahead to develop a grid portlet that provides an interface for that web service. The first step is making a choice. We can either develop a new portal framework or work with existing ones. Since a number of portal frameworks are available, it was better to work with an existing popular framework. We chose to work with Gridsphere which is 100% JSR 168 compliant and an open-source portal framework. The Gridsphere portal framework provides an open-source portlet based web portal. GridSphere enables developers to quickly develop and package portlet web applications that can be run and administered within the GridSphere portlet container**.**

5.2.1 Installing Gridsphere

To run in GridSphere, first download and install GridSphere. We started installing the gridsphere portal framework in C drive under the folder-name *gridsphere*. We should set your CATALINA_HOME and ANT_HOME environment variables. GridSphere normally runs in Tomcat 5.0.2x. As we are using Tomcat 5.5.x, we should a) install the java 1.4.2 compatibility package as instructed above, and b) copy ant.jar and ant-launcher.jar from Tomcat 5.0.2x into 5.5.x's common/lib.

The procedure to a successful Installation is given.

First set a gridsphere environment variable GRIDSPHERE_HOME=C:\gridsphere. GridSphere relies on Ant for its compilation and deployment. The build script, build.xml, provides various targets for compiling, deploying, building javadoc api documentation, creating new portlet projects. From inside the GridSphere source directory, we will run:

ant install

This will compile the framework code, create documentation if necessary and deploy GridSphere to a Tomcat container.

Before we run gridsphere, we need to modify the following files:

- To enable deploying portlets in a running GridSphere modify the file: C:\tomcat\conf\tomcat-users.xml. We have to make the following entry in the <tomcat-users> section of the file.
  <user username="gridsphere" password="gridsphere" roles="manager, admin"/>

- Edit the file C:\tomcat\webapps\gridsphere\WEB-INF\GridSphereServices.xml. In this file, edit the user name and password used in the Portlet Manager Service entry to be the name and password used in the tomcat-users.xml.

- To ensure that clients cannot access the manager webapp directly, add as a child of <host>C:\tomcat\conf\server.xml. We need to edit the following code:

  ```
  <Context path="/manager" debug="0" privileged="true"
  docBase="C;\tomcat\server\webapps\manager">
  <Valve className="org.apache.catalina.valves.RemoteAddrValve"
  allow="127.0.0.1"/>
  </Context>
  ```

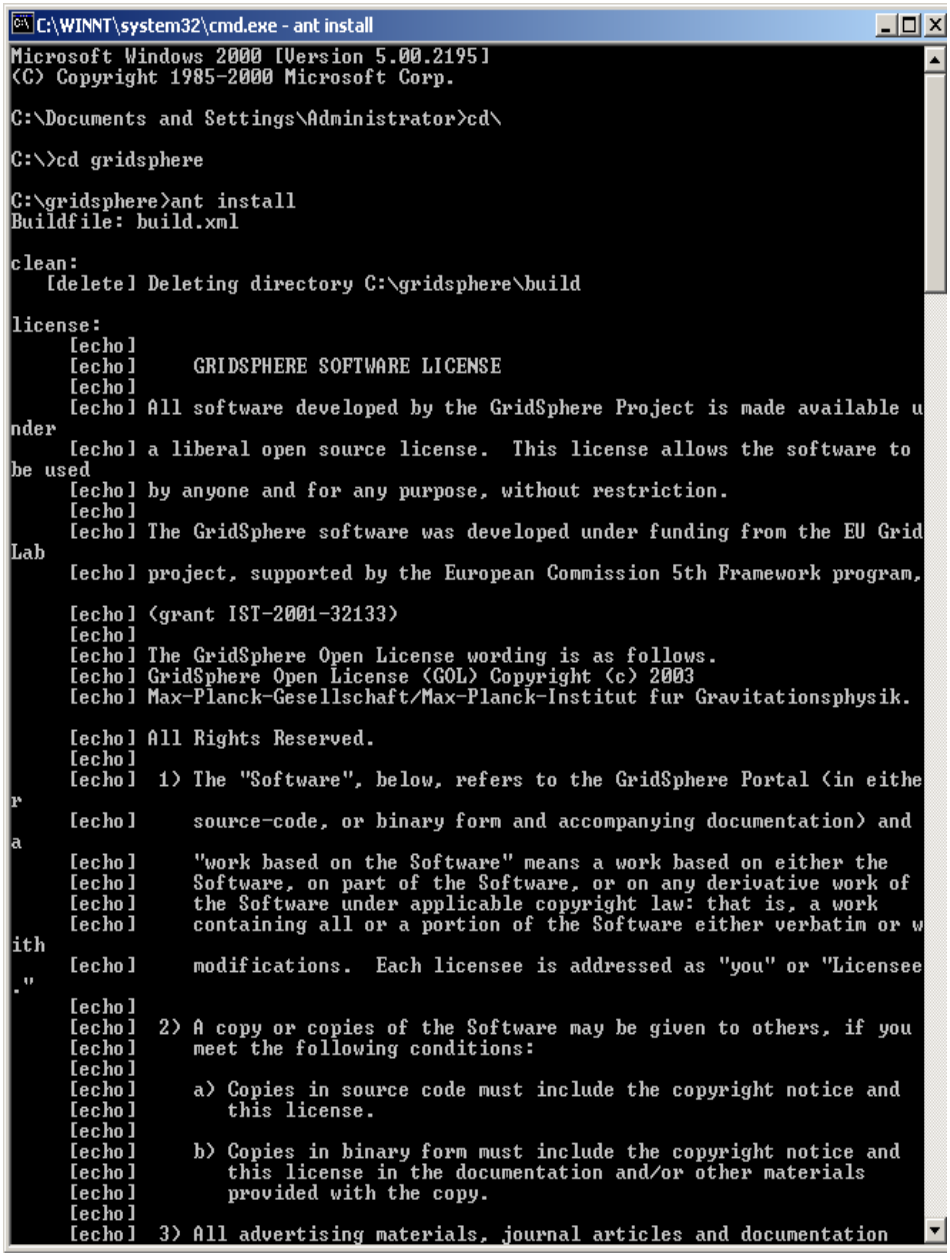


Figure 5.7: Installing Gridsphere

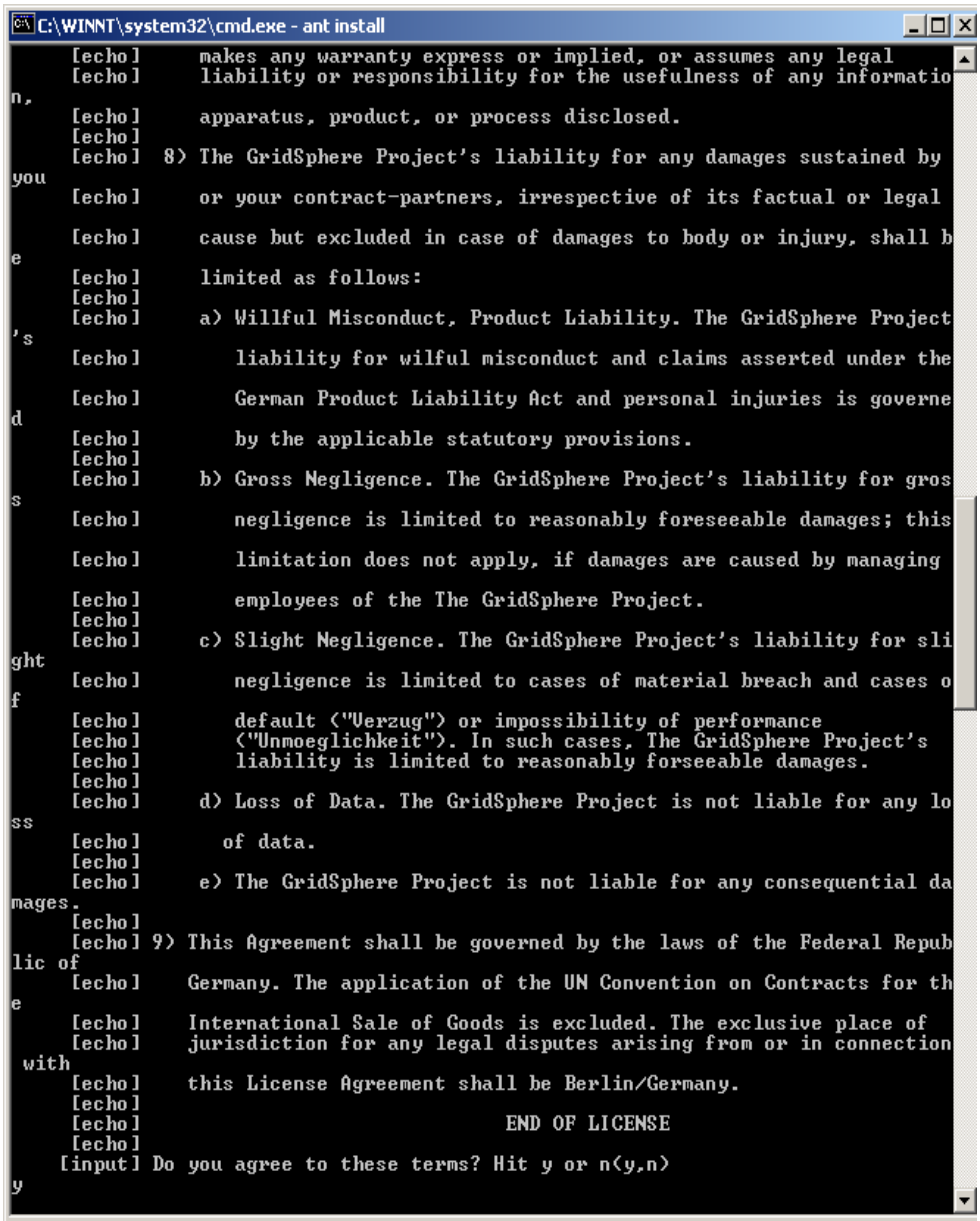


Figure 5.8: Installation (Contd.)

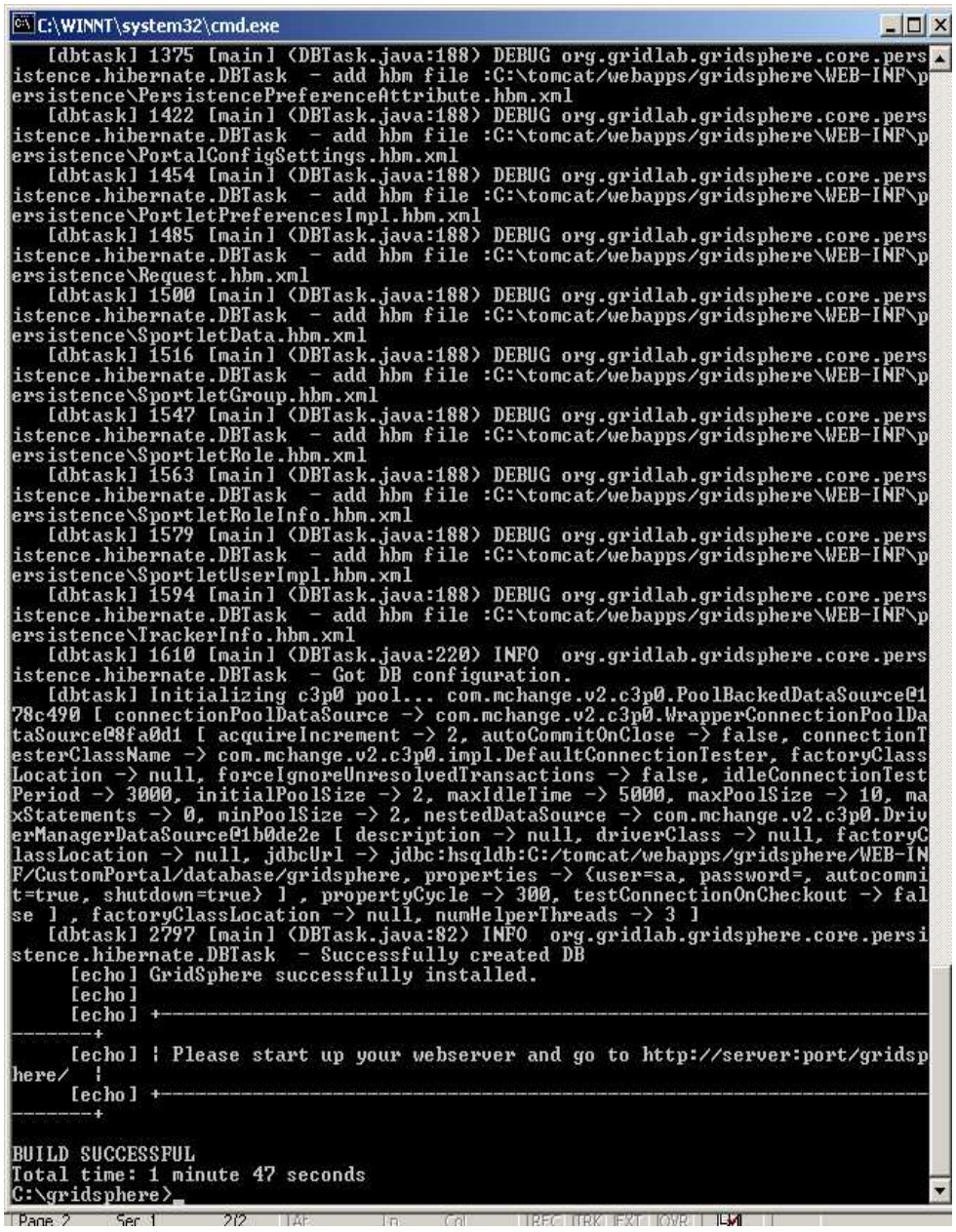


Figure 5.9: Installation Complete

After the installation completes and we see "BUILD SUCCESSFUL" (in screen shot 5.7). Next we start the Tomcat container .To start Tomcat we have to run the following command: *startup.bat.* The details can be seen in the screen shots below:

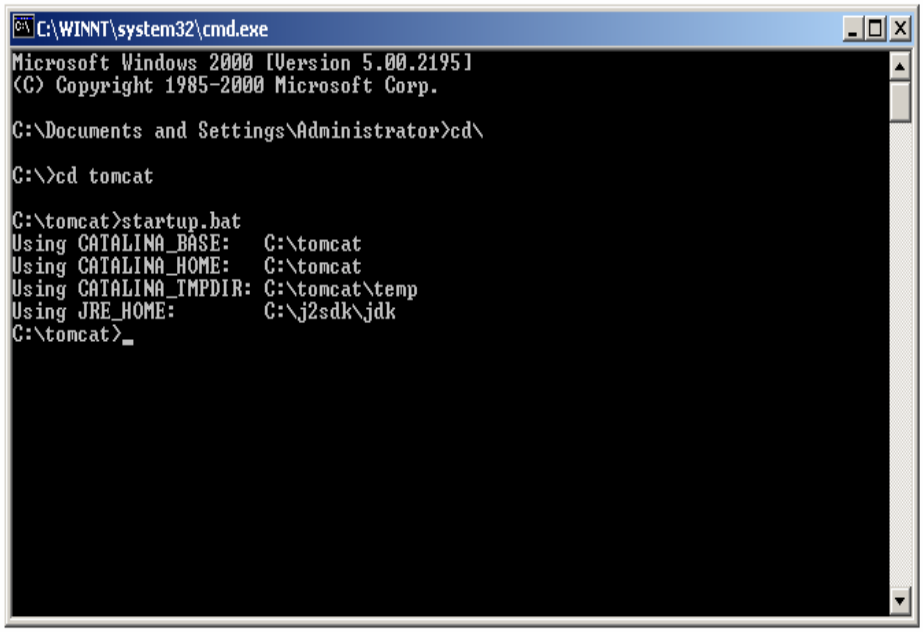


Figure 5.10: Starting Tomcat

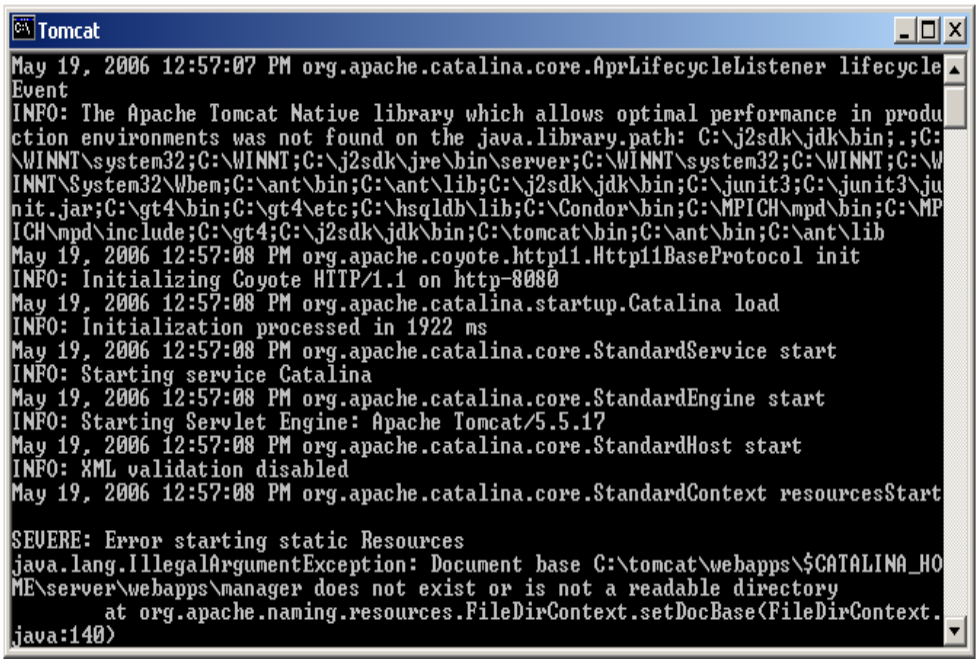


Figure 5.11: Tomcat Running

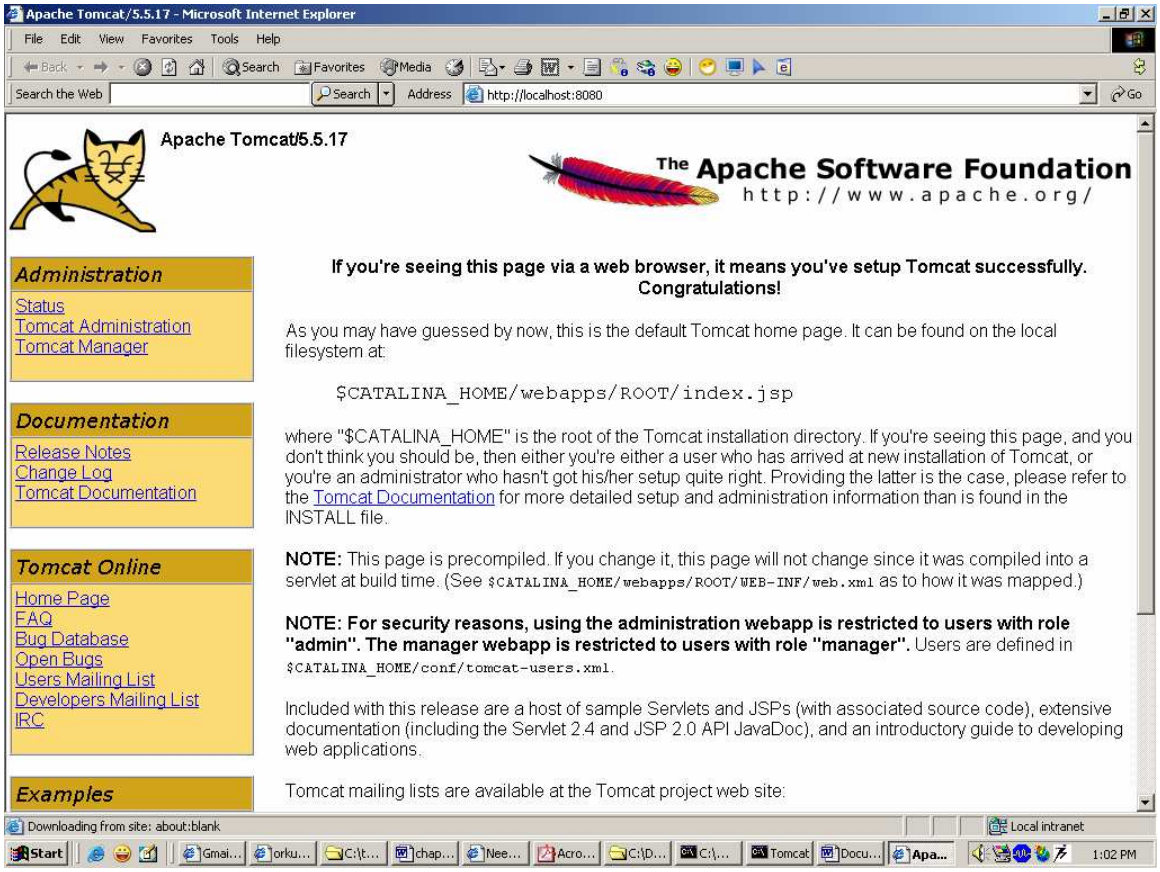


Figure 5.12: Testing Tomcat

After it is confirmed that Tomcat is running properly, we can then migrate to

http://localhost:8080/gridsphere/gridsphere .

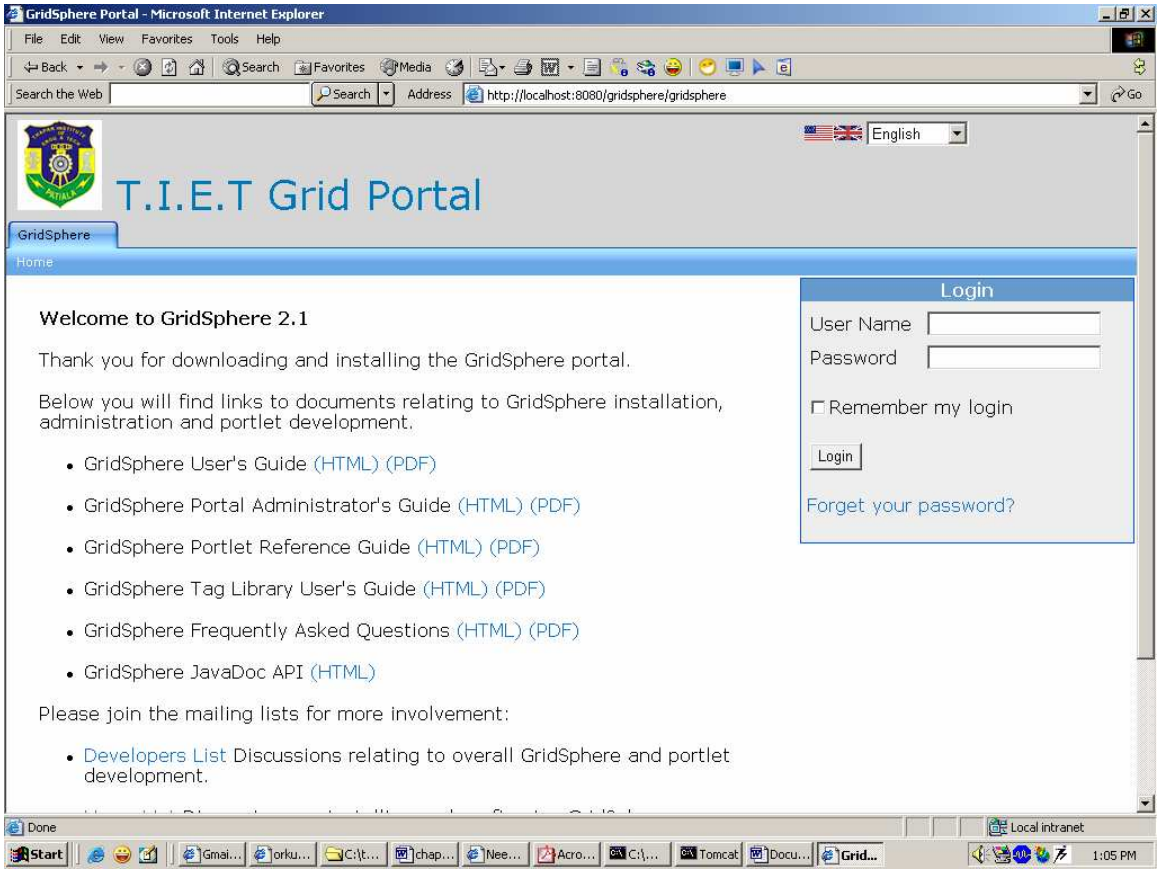


Figure 5.13: TIET Portal Main Screen

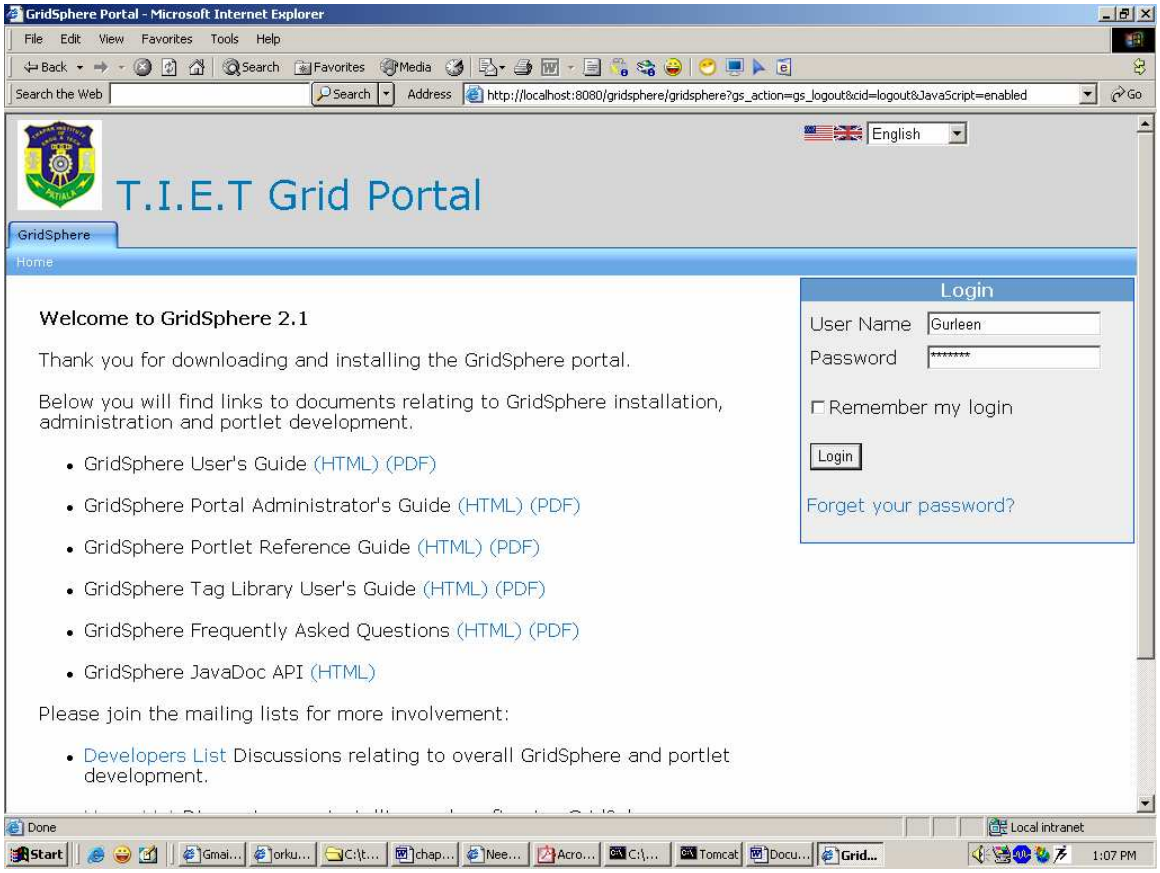


Figure 5.14: TIET Portal Login Portlet

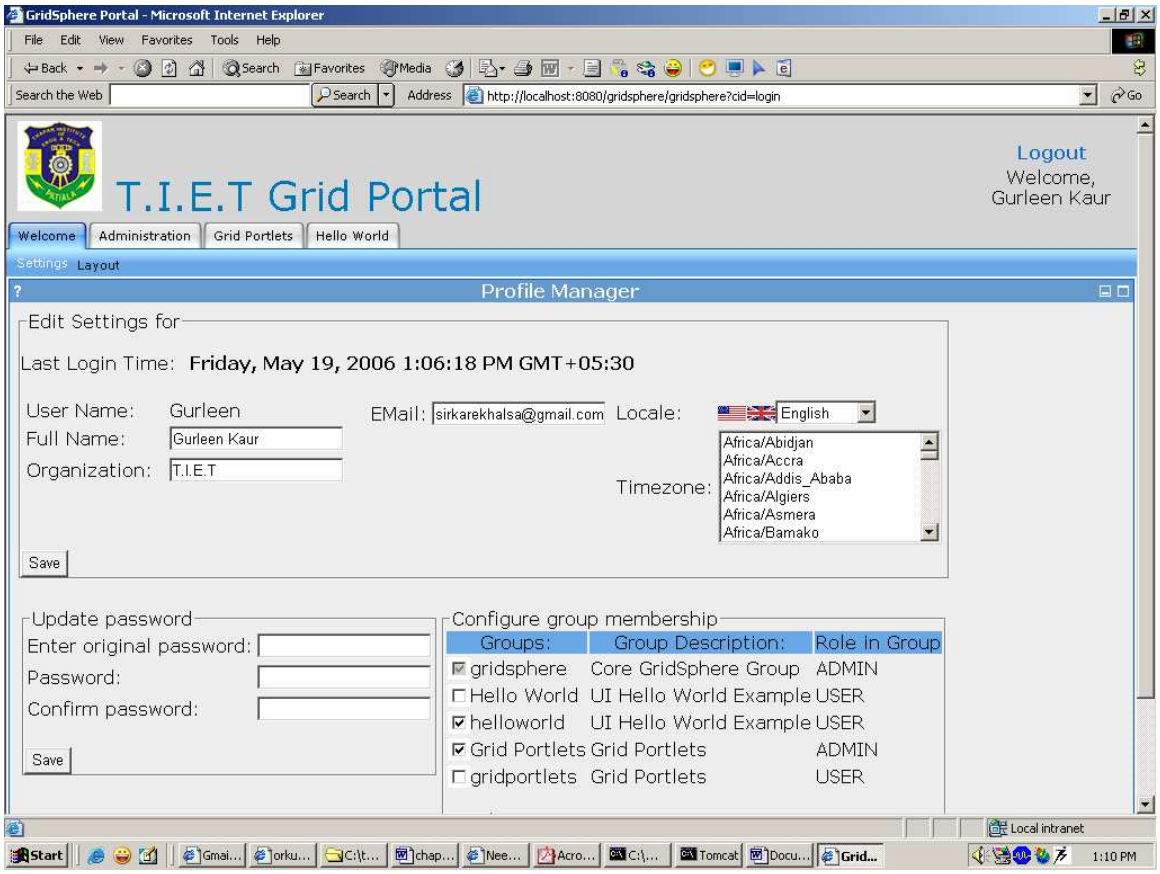


Figure 5.15: TIET Portal Profile Manager Portlet

Next we have to develop a portlet for our HelloWorldService. In GRIDSPHERE_HOME run 'ant new-project'.

- You will be asked for a project title, enter *Hello World***.**
- You we will be asked for a project name, this will be used for the web application name, enter *helloworld* (this will be used as directory name for the project).
- You will be asked whether we want this to be a GS or JSR portletwebapp, enter jsr.

All template files are now created in GRIDSPHERE_HOME\projects\helloworld. We demonstrate all the above steps with the following screen shots.

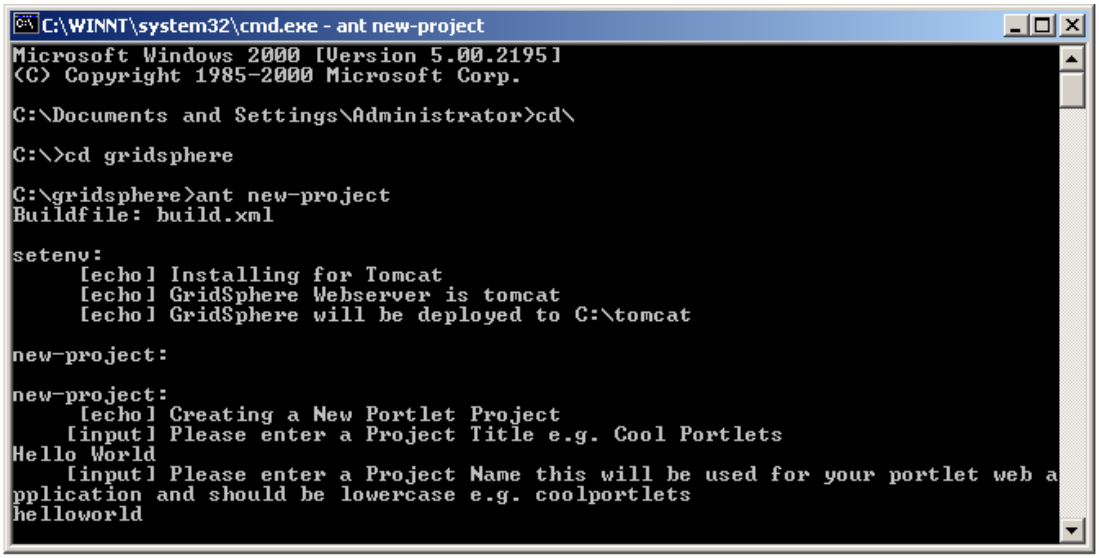


Figure 5.16:Creating a Gridsphere Portlet

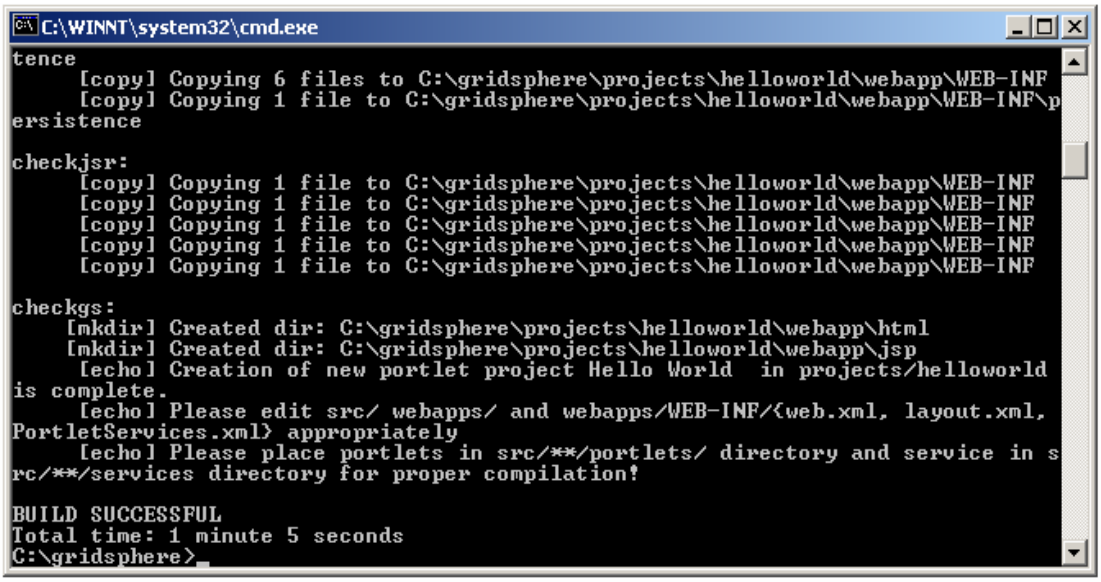


Figure 5.17: Portlet Built

Towards the end of this step all template files will be created in GRIDSPHERE_HOME\projects\helloworld.

After the portlet template has been developed, we need to follow some steps to build the actual portlet:

- **Creating the JSP (Java Server Pages)**
  - JSP's are stored in webapp\jsp subdirectory of the just created helloworld project.
  - Create a subdirectory helloworld in the jsp directory.
  - Create a jsp file *helloworld.jsp*:

- **Creating the portlet**
  - Create a directory src\org\gridlab\gridsphere\jsrtutorial\portlets\helloworld.
  - The client code for the Globus service should be incorporated into the processAction () method of the GenericPortlet class.
  - Create the following portlet code in HelloWorld.java: The code for the service is drawn from the GT4 container.
- All files are located in GRIDSPHERE_HOME/webapp/WEB-INF. We need to edit a couple of deployment descriptor files.
  - **portlet.xml** - JSR 168 standard, describing the portlet. Edit the</portlet> section of portlet.xml shown below:
  - **layout.xml** - GridSphere file, describing the layout of the portlet within a page.
  - **group.xml** - GridSphere file, Describing a collection of portlets. The contents are shown below:

Run *'ant install'* at command prompt to deploy the portlet to GridSphere. Start GridSphere. Subscribe to the HelloWorld group in the Profile Manager as shown below:

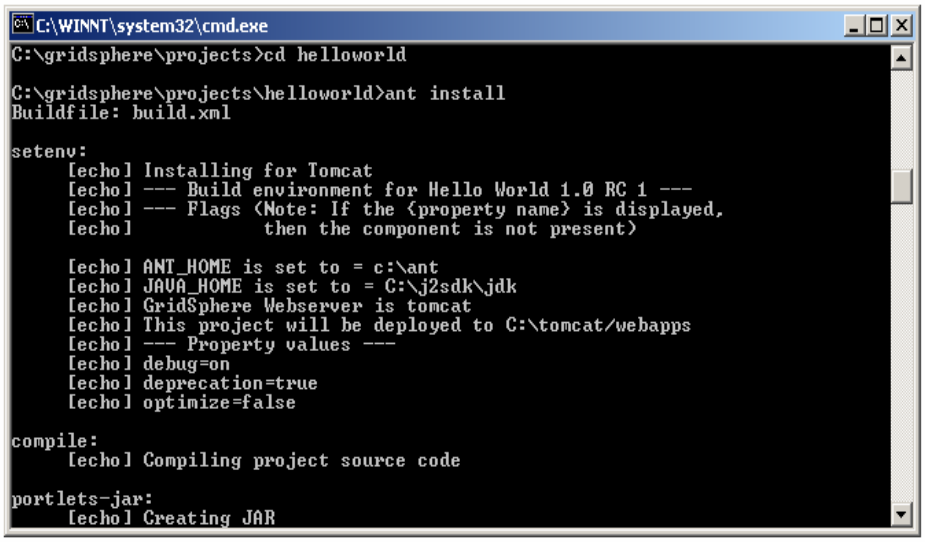


Figure 5.18: Portlet Configuration

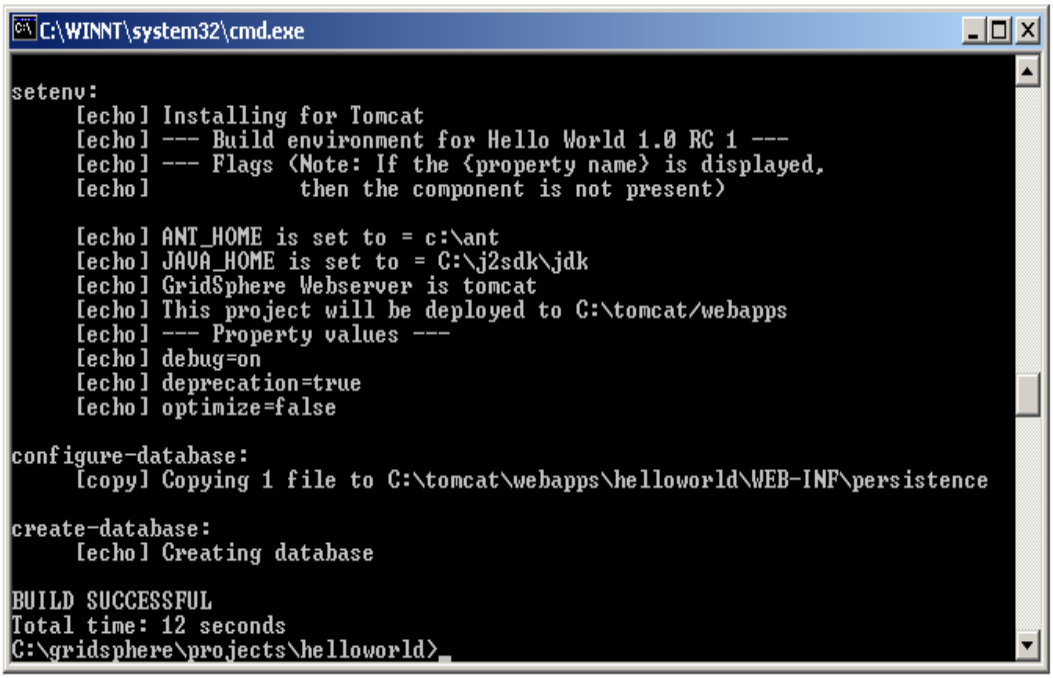


Figure 5.19: Portlet Configuration Complete

After “BUILD SUCCESSFUL” pops up on the screen we can now restart Tomcat and go to gridsphere.
Under the administration tab, click Groups.

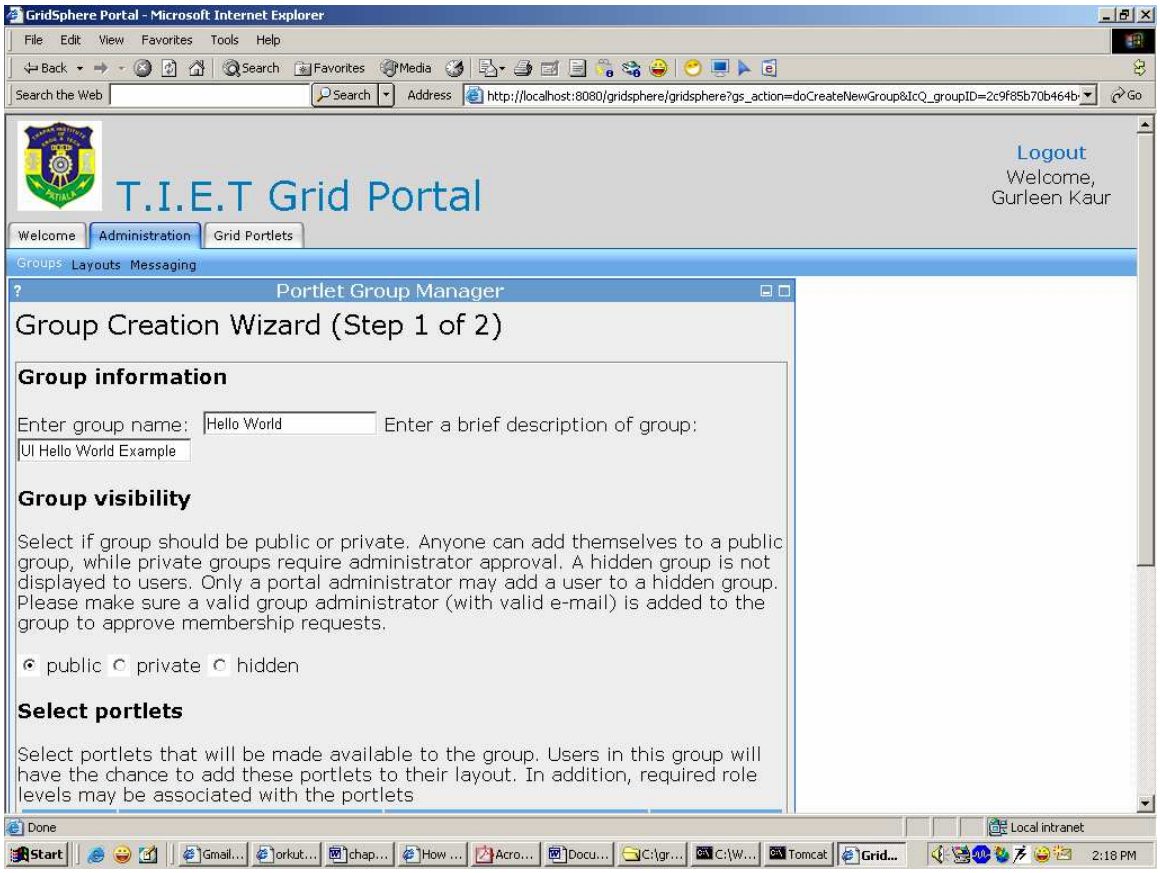


Figure 5.20: Group Creation Wizard (Step1)

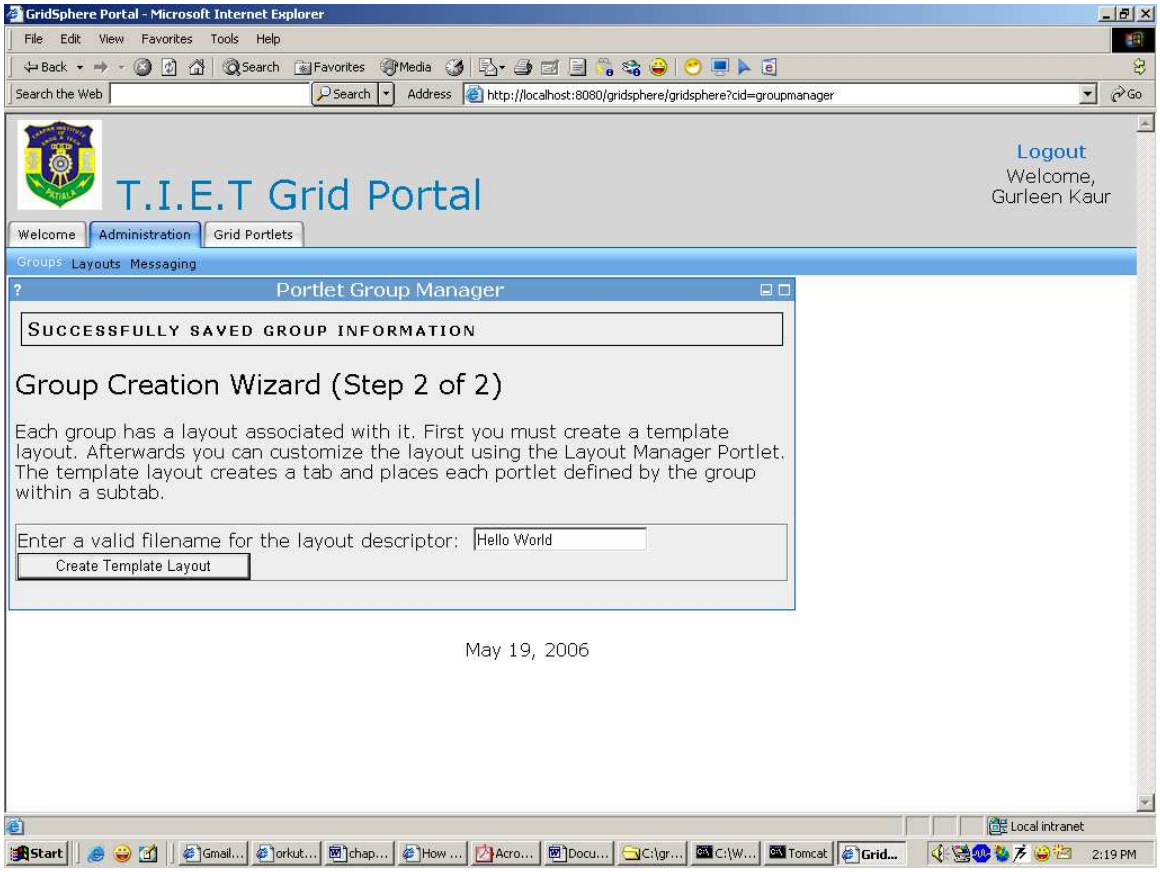


Figure 5.21: Group Creation Wizard (Step2)

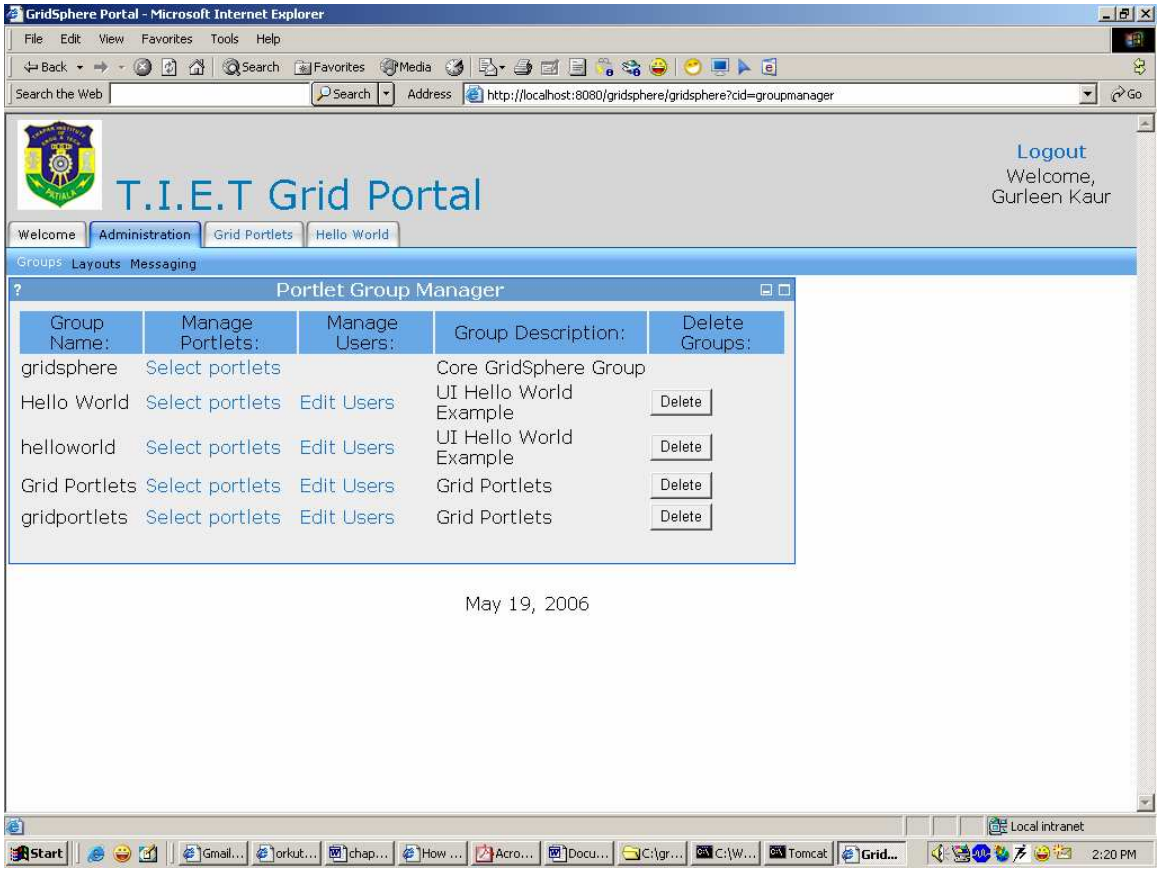


Figure 5.22: Group Created Successfully

Finally we have the portlet ready.

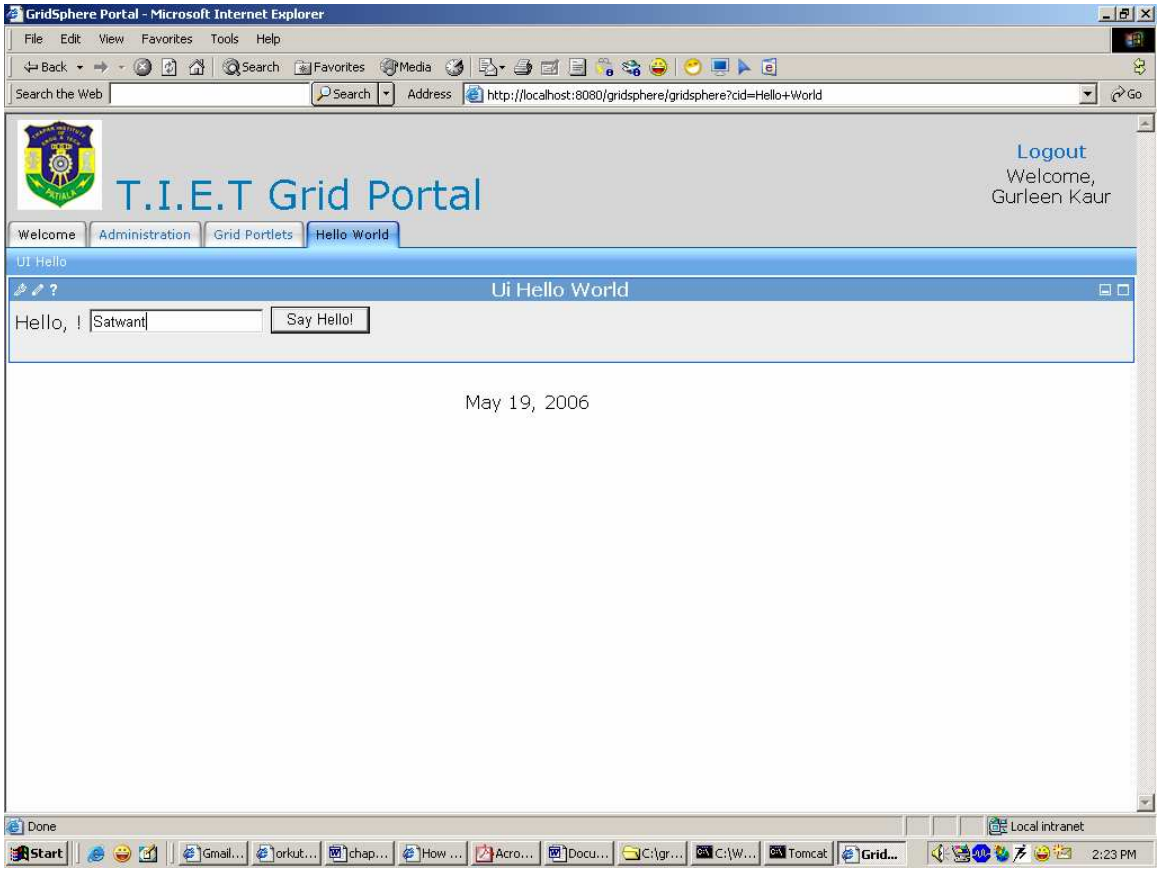


Figure 5.23: Hello World Portlet

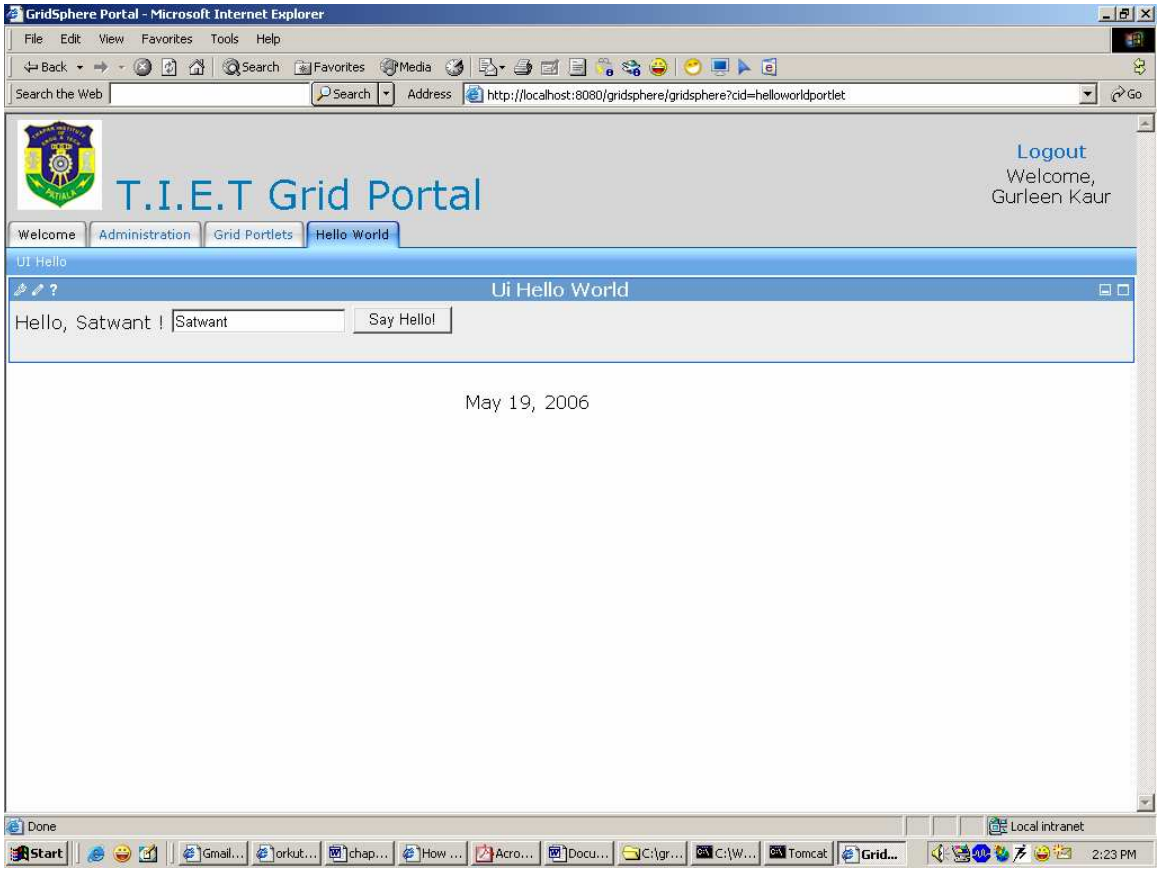


Figure 5.24: Hello World Portlet at work

CHAPTER 6
CONCLUSION & FUTURE SCOPE

6.1 Conclusion

The thesis explores the field of grid computing, which is emerging as a new infrastructure of the 21st century. We have addressed all the information (characteristics, architecture, nature of applications and history of grid computing) required to develop a rapport with this budding technology.

We have discussed Globus Toolkit4, which is widely adopted as a Grid technology solution for scientific and technical computing, and Web services, which have emerged as a popular standards-based framework for accessing network applications. Next, we discussed briefly about Grid Portals. A Grid Portal is a web-based application that provides personalization, single sign on, content aggregation from different sources and hosts the presentation layer of information systems. Thus a portal liberates the end-user from details of grid middleware programming and allows seamless access to applications and services running on remote resources.

We focused on two specific issues:

- Capturing a Grid application as a Web service.
- Generating an application specific interface that can be provided to the end user through the portal.

6.2 Future Scope

Grid computing has a lot of vested interest by large companies that can use it to connect geographically diverse offices and unify the supply chain. Portals are the way to do this. In industrial sector, IBM is the leader in this field and expending myriads of funds. A number of other organizations such as HP, Oracle and Sun are also getting engaged. Good numbers of academic institutions like IITs are also adding to their grid computing resources.

From the above details we can conclude that the future of Grid services, based on the array of current technologies, is very healthy. We have been able to harness only some capabilities of Gridsphere portal framework. An integration of GT4 and Gridsphere gas been accomplished as a part of this thesis. Whereas in the future, we can work towards integrating GridSphere with other grid middleware such as Alchemi (a .NET based framework), Condor, Nimrod/G and several others.

REFERENCES

[1] I. Foster, C. Kesselman, eds., *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, San Francisco, Calif. (1999).

[2] I. Foster, C. Kesselman, S. Tuecke, *Int. J. High Perform. Comput. Appl.* 15(3), 200 (2001). Also available at http://www.globus.org/research/papers/anatomy.pdf.

[3] I. Foster. What is the Grid? A Three Point Checklist. GRIDToday, July 20, 2002.

[4] Foster, I., Kesselman, K., Nick, J., Tuecke, S., "The Physiology of the Grid. An Open Grid Services Architecture for Distributed Systems Integration", in *Grid Computing: Making the Global Infrastructure a Reality*, Fox, G. Ed. Wiley, 2003.

[5] Johnston, "Implementing Production Grids", in *Grid Computing: Making the Global Infrastructure a Reality*, Fox, G. Ed. Wiley, 2003.

[6] Fox, G., Gannon, D., "Computational Grids", IEEE Computer Science Eng. Vol. 3, No. 4, pp. 74-77, 2001.

[7] Global Grid Forum, http://www.ggf.org.

[8] I. Foster, C. Kesselman, J. Nick, S. Tuecke: " The Anatomy of the Grid: Enabling Scalable Virtual Organization. Accessed from: http://www.mnlab.cs.depaul.edu/seminar/fall2002/GridAnatomy.pdf.

[9] G. von Laszewski and P. Wagstrom, Tools and Environments for Parallel and Distributed Computing, ser. Series on Parallel and Distributed Computing. Wiley, 2004, ch. Gestalt of the Grid, pp.149–187. [Online]. Available: http://www.mcs.anl.gov/_gregor/papers/vonLaszewski--gestalt.pdf.

[10] "The Commodity Grid Project," Web Page. [Online]. Available: http://www.globus.org/cog.

[11] "World Wide Web Consortium," Web Page. [Online]. Available: http://www.w3.org/.

[12] Thompson, R. and Schaeck, T. Enabling Interactive, Presentation-Oriented Content Services Through the WSRP Standard. XML Conference and Exposition 2003. Philadelphia, USA. December 2003. http://www.idealliance.org/papers/dx_xml03/papers/04-06-05/04-06-05.html.

[13] Kropp. A, Leue, C and Thompson, R. Web Services For Remote Portlets

Specification. OASIS. August 2003.http://www.oasis-open.org/committees/download.php/3343/oasis-200304-wsrpspecification-1.0.pdf.

[14] S. Graham et al., *Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI*, Sams Publishing, Indianapolis, Ind. (2001).

[15] GridLab,TheGridSphere Portal http://www.gridsphere.org/gridsphere/gridsphere.

[16] The PORTAL project http://www.fair-portal.hull.ac.uk/.

[17] Linwood, J. and Minter, D. Building Portals With the Java Portlet API (Expert'sVoice) Apress. August 2004 Sample Chapters available at http://www.theserverside.com/articles/content/BuildingPortals/Sample+Booklet_Linwood-Minter.pdf.

[18] Davidson, C. and Coco, C. (Eds). Web Services for Remote Portlets 1.0 Frequently Asked Questions. WSRP-faq-1.0http://www.oasis-open.org/committees/download.php/10953.

[19] K. Czajkowski, S. Fitzgerald, I. Foster, C. Kesselman. Grid Information Services for Distributed Resource Sharing. Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10), IEEE Press, August 2001.

[20] J. Novotny, M. Russell, O. Wehrens. "GridSphere: An Advanced Portal Framework", EUROMICRO, 2004. Also accessed from: http://www.gridsphere.org/gridsphere/gridsphere? cid=presentations.

[21] Globus Toolkits.http://www.globus.org/toolkit/.

[22] R. Armstrong, D. Gannon, A. Geist, K. Keahey, S. Kohn, L.McInnes, S. Parker, and B. Smolinski, "Toward a common component architecture for high performance scientific computing,"in *Proc. 8th IEEE Int. Symp. High Performance Distributed Computing*, 1998, pp. 115–124.globus.org/wsrf.

[23] Anil. Kumar, "Data Grid", submitted at Computer Science and Department, Thapar Institute of Engineering and Technology,Patiala ,2003.

[24] B. Saxena, "Grid Computing,", submitted at Computer Science and Department, Thapar Institute of Engineering and Technology, Patiala ,2003.

PAPER COMMUNICATED/PUBLISHED/ACCEPTED

[1]. Gurleen Kaur and Seema Bawa **"Developing Web Services in Grid Environment"** at 7th IEEE/ACM International Conference on Grid Computing to be held in Barcelona, Spain from September 28-29, 2006 **[Communicated].**