A Framework for Resource Management over a GRID – An Investigative Study

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

Doctor of Philosophy
in
Computer Science

By

D. Sudheer Kumar Reddy

Under the supervision of

Prof. Arun Agarwal

Department of Computer & Information Sciences

School of Mathematics and Computer/Information Sciences

University of Hyderabad

Hyderabad – 500 046, INDIA

March, 2009

CERTIFICATE

This is to certify that the dissertation entitled **"A Framework for Resource Management over a GRID – An Investigative Study"** being submitted to the University of Hyderabad, by **Mr. D. Sudheer Kumar Reddy (Reg. No. 03MCPC01)** in partial fulfillment for the award of **Doctor of Philosophy** in **Computer Science** is a bonafied work carried out by him under our supervision. The matter embodied in this thesis has not been submitted to any other University for award of any degree or diploma.

Prof. Arun Agarwal,
(Supervisor)
Department of Computer and
Information Sciences,
University of Hyderabad,
Hyderabad.

Prof. Arun Agarwal,
Head,
Department of Computer and
Information Sciences,
University of Hyderabad,
Hyderabad.

Prof. T. Amaranath,
Dean,
School of Mathematics and
Computer/Information Sciences,
University of Hyderabad,
Hyderabad.

DECLARATION

I hereby declare that the work embodied in this thesis has been carried out by me under the supervision of **Prof. Arun Agarwal**, in the department of Computer and Information Sciences, University of Hyderabad, Hyderabad and has not been submitted at any other University. Information derived from the published and unpublished work of others has been acknowledged and a list of references is given.

Place: Hyderabad (D. Sudheer Kumar Reddy)
Date:

ACKNOWLEDGEMENTS

I would like to express my deep gratitude to my supervisor, mentor and Head of the Department **Prof. Arun Agarwal**, for his incessant support, encouragement, guidance and advice that made this dissertation possible. His emphasis on the quality of research and writing has been extremely valuable in writing this dissertation. His valuable suggestions helped me a lot to learn both technically and personally. I thank him for providing the opportunity to work with him.

I would like to thank my Doctoral Review Committee members, **Dr. Rajeev Wankar**, **Dr. Atul Negi** and **Dr. Vineet Nair** for their suggestions and valuable criticism on my research work.

I thank **Prof. T. Amaranath,** Dean, School of Mathematics & Computer/Information Sciences for providing facilities to pursue my doctoral studies.

I would like to thank staff members of **Department of Computer & Information Sciences** and **School of Mathematics & Computer/Information Sciences** who have been kind enough to advise and help in their respective roles.

I thank my managers **K. V. Rama Krishna** and **NDT Venkateswar** at work for understanding my research interest and for their encouragement. I also thank my earlier managers **Sridhar** and **Vijay Kumar** for their support.

I greatly appreciate **Phanindra Kasturi**, my friend and ex-colleague for his generous time and timely insightful suggestions. I would like to express my special thanks to him for being a great soul. He is always ready to help me with a smile.

I would like to thank my colleagues **Rakesh**, **Vijay Duddu**, and **Saradhi** who have shared their knowledge and for their time in proof reading of thesis. In this regard a special mention needs to be made of **Rakesh** for his patience in reviewing all my draft thesis chapters and papers. I would like to thank another ex-colleague **Chandra Sekhar**, who helped me in developing User Interfaces.

I wish to thank all my co researchers whose cheerful company made my stay enjoyable during the research work, especially **Uma Devi** for her timely information on DRC reviews and semester registrations.

I also thank M.Tech students, **Kalyani**, **Aditya**, **Saran**, **Nilakanta** and **Naveen** for their help and cooperation during my research work at University of Hyderabad.

My special thanks to my wife **Swapna** and daughter **Shravya**, for their love, patience, and understanding during my research work, especially during my thesis writing.

Finally, I thank one and all, who have directly or indirectly contributed to my success.

D. Sudheer Kumar Reddy

CONTENTS

Abstract

Grid computing technology is an emerging technology where computational and information resources are shared and managed by diverse organizations in widespread locations which are referred as virtual organizations. Grid computing offers valuable services to work groups such as auto and airplane designers, scientific researchers, drug-research firms. In Grid environments, resources are generally owned by different people, communities or organizations with varied administrative policies, and capabilities. The management of these resources i.e., discovery, monitoring and brokerage in such a large and distributed environment is a complex task due to geographical distribution and dynamic behavior of resources. Managing grid infrastructure plays a vital role in effectively utilizing the capabilities provided by grid computing technology.

The complexity of grid environment is growing due to increase in number of projects and applications that are being pursued in this domain. The demands exerted on grid resources by these applications are not only highly dynamic in nature but also varies substantially from application to application. For example, one application may need large number of CPUs and another application is memory intensive. This makes it difficult to identify idle resources, and the resource to application mapping. In such scenarios it is imperative that grids are managed by a well defined and scalable resource management system. Hence new paradigms and frameworks that deeply impact the possibility of discovering, monitoring and brokerage the resources need to be developed, with main focus on distributed management, efficiency, scalability, adaptability and reliability. As nodes and/or clusters appear and disappear quickly on the grid, automatic resource and service discovery should be performed. Continuous monitoring of the changes in grid behavior shall reflect the current state of resource availability. Brokerage need to be done based on the resource availability. To address these challenges, we have developed an architectural framework called Grid Management Information Server (GMIS).

The GMIS framework provides services for discovery, monitoring and brokerage of resources in a grid network. It includes distributed resource management architecture by employing remote monitoring of dispersed resources. The architecture distributes

resource management functionality across grid networks using GMIS-GMIS communication, where each GMIS manages a set of grids independently. A single user interface is provided to view resource information available at a single GMIS or all connected GMISs. Inadequacies of earlier approaches are clearly brought out with a comparative study and our contributions to this domain are presented.

List of Figures

List of Tables

CHAPTER 1

Introduction

This chapter provides introduction to Parallel Computing, Distributed Computing and Grid Computing. It briefly discusses about types of Grids, Grid characteristics and key components of Grid computing. It presents the motivation and objectives of the research work followed by summary of key contributions. It ends with a discussion on the organization of the rest of this dissertation.

1.1 Parallel Computing

Traditionally, software has been written for serial computation, to be executed by a single computer having a single Central Processing Unit (CPU) and problems are solved by a series of instructions, executed one after the other by the CPU. Only one instruction executes at any moment in time.

Parallel computing is an evolution of serial computing that attempts to emulate what has always been the state of affairs in the natural world. It addresses many complex, interrelated events happening at the same time, in a sequential fashion. Some examples are:

- Planetary and galactic orbits
- Weather and ocean patterns

In the simplest sense, parallel computing [Parallel Computing] is the simultaneous use of multiple resources to solve a computational problem. The resources can include, a single computer with multiple processors. There are two primary reasons for using parallel computing:

- Save time – wall clock time
- Solve larger problems

Other reasons might include:

- **Taking advantage of non-local resources** – using available compute resources on a wide area network, or even the Internet when local compute resources are not sufficient for a computation.
- **Cost savings** – using multiple "cheap" computing resources instead of paying more on a super computer.
- **Overcoming memory constraints** – isolated computers have very finite memory resources. For large problems, using the memory available with multiple computers may overcome this obstacle.

However, multiple tasks cannot run through parallel computing, for which distributed computing has evolved.

1.2 Distributed Computing

Distributed Computing is a type of computing in which different components and objects comprising an application can be located on different computers connected to a network. For example, a word processing application might consist of an editor component on one computer, a spell-checker application on a second computer, and a thesaurus on a third computer. In some distributed computing systems, each of the three computers could even be running a different operating system.

One of the requirements of distributed computing is a set of standards that specify how objects communicate with one another. There are currently two distributed computing standards: CORBA and DCOM [Distributed Computing].

Various distributed technologies are:

- Distributed Computing systems
- World Wide Web
- Application and Storage Service Providers
- Peer-to-Peer Computing systems

- Cluster Computing

The difference between parallel computing and distributed computing is, parallel computing brings many CPUs together in one box, and where as distributed computing refers to the linking up of many individual computers to perform the task.

1.3 Problems with Parallel Computing & Distributing Computing

Problems with traditional parallel computing are:

- Expensive software
- Very high starting cost
- High maintenance cost
- Costly to upgrade

Because of these the computing power of an organization is generally not enough to finish its task in a reasonable amount of time.

Problems with various distributed technologies are:

- Cluster computing is local to domain and not amenable to resource sharing among participant from different domain.
- Lack of security features
- Respecting the policies of individual organizations participating in resource sharing.

Grid Computing addresses the above problems by having:

- **Resource sharing:** Resources in a grid belong to many different organizations that allow other organizations (i.e. users) to access them. Non local resources can be used by applications, promoting efficiency and reducing costs of the resources while allowing its scalability.

- **Multiple administrations:** Each organization may establish different security and administrative policies under which their resources can be accessed and used.

1.4 Grid Computing

Grid computing is a form of distributed computing; the pioneers in this field are Dr. Ian Foster of Argonne National Laboratory and the University of Chicago, and Dr. Carl Kesselman of the Information Sciences Institute and the University of Southern California. In their 1998 seminal work, The Grid: Blueprint for a New Computing Infrastructure, they defined Grid computing as follows:

"A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities." [Ian Foster, Carl Kesselman, 1999]

Grid computing is distributed computing taken to the next evolutionary level. The goal is to create the illusion of a simple yet large and powerful self-managing virtual computer out of a large collection of connected heterogeneous systems sharing various combinations of resources. The grid is a new approach to high-performance and large scale networked computing that has the potential to alter how computing resources are allocated for large projects. Built on the Internet and the World Wide Web, it is a new class of infrastructure comprising a set of high-speed computers, storage systems and networks, plus a set of grid services (or middleware) to coordinate the ensemble of resources. 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 [Arun Agarwal, 2004].

Grid computing area has attracted a number of researches in recent years due to the unique ability of marshalling collections of heterogeneous computers and resources, enabling easy access to diverse resources and services that could not be possible without a grid model.

A grid in reference to grid computing refers to loose resources and users that may be geographically separated and may be under different administrative domains. A grid uses the resources of many separate computers connected by a network (usually the Internet) to solve large scale computation problems and use most of idle time on many resources throughout the world. Such arrangements permit handling of data that would otherwise require the power of expensive supercomputers or would have been impossible to analyze.

1.4.1 Types of Grids

From an application perspective, grids are categorized into two types: **compute grids** and **data grids**. Although from a topology perspective, it can be argued that there are additional types that include clusters, intra-grids, extra-grids, and inter-grids. In reality, clusters, intra-grids, extra-grids, and inter-grids are better defined as stages of evolution that fall under compute grids, data grids, or a combination of both. The majority of the early grid deployments have focused on enhancing computation, but as data grids provide easier access to large, shared data sets, data grids are becoming increasingly important.

A **compute grid** is essentially a collection of distributed computing resources, within or across locations that are aggregated to act as a unified processing resource or virtual supercomputer. These compute resources can be either within or between administrative domains. Collecting these resources into a unified pool involves coordinated usage policies, job scheduling, queuing characteristics, grid-wide security, and user authentication. The benefit is faster, more efficient processing of compute-intensive jobs, while utilizing existing resources. Compute grids also eliminate the drawback of tightly binding specific machines to specific jobs, by allowing the aggregated pool to most efficiently service sequential or parallel jobs with fine-grained user attributes.

A **data grid** provides wide area, secure access to data. Data grids enable users and applications to manage and efficiently use database information from distributed locations. Similar to compute grids, data grids also rely on software for secure access and usage policies. Data grids can be deployed within one administrative domain or across multiple domains. In these cases the grid software and policy management

becomes critical. Data grids eliminate the need to unnecessarily move, replicate, or centralize data, translating into cost savings. Initial data grids are being constructed today, primarily serving collaborative research communities. Software vendors and large enterprises are currently investigating data grid solutions and services for business applications.

1.4.2 Grid Characteristics

Ten definitions extracted from main grid literature sources have been examined to find out the essential characteristics that a grid is supposed to have in order to be of any practical use.

1. ***Large scale*:** a grid must be able to deal with a number of resources ranging from just a few to millions. This raises the very serious problem of avoiding potential performance degradation as the grid size increases.

2. ***Geographical distribution*:** grid's resources may be located at distant places.

3. ***Heterogeneity*:** a grid hosts both software and hardware resources that can be varied ranging from data, files, software components or programs to sensors, scientific instruments, display devices, personal digital organizers, computers, super-computers and networks.

4. ***Resource sharing*:** resources in a grid belong to many different organizations that allow other organizations (i.e. users) to access them. Non-local resources can thus be used by applications, promoting efficiency and reducing costs of the resources while allowing its scalability.

5. ***Pervasive access*:** the grid must grant access to available resources by adapting to a dynamic environment in which resource failure is commonplace. This does not imply that resources are universally available but that the grid must tailor its behavior as to extract the maximum performance from the available resources.

6. ***Multiple administrations*:** each organization may establish different security and administrative policies under which their resources can be accessed and used. As a

result, the already challenging network security problem is complicated even more with the need of taking into account all different policies.

7. ***Resource coordination*:** resources in a grid must be coordinated in order to provide aggregated computing capabilities.

8. ***Transparent access*:** a grid should be seen as a single virtual computer.

9. ***Dependable access*:** a grid must assure the delivery of services under established Quality of Service (QoS) requirements. The need for dependable service is fundamental since users require assurances that they will receive predictable, sustained and often high levels of performance.

10. ***Consistent access*:** a grid must be built with standard services, protocols and interfaces thus hiding the heterogeneity.

1.4.3 Key Components of Grid Computing

Grid Computing contains the following major components [IBM developerWorks]:

- **Security:** Computers on a grid are networked and running applications. They can also be handling sensitive or extremely valuable data, so the security component of grid computing is of paramount concern. This component includes elements such as encryption, authentication, and authorization.

- **User Interface:** Accessing information on the grid is also quite important, and the user interface component handles this task for the user. It often comes in one of two ways:

    - An interface provided by an application that the user is running
    - An interface provided by the grid administrator, much like a Web portal that provides access to the applications and resources available on the grid in a single virtual space.

- **Workload management:** Applications that a user wants to run on a grid must be aware of the resources available. This is where a workload management service comes in handy. An application can communicate with the workload manager to discover the available resources and their status.

- **Scheduler:** A scheduler is needed to locate the computers on which to run an application and to assign the jobs required. This can be as simple as taking the next available resources, but this task often involves prioritizing job queues, managing the load, finding workarounds when encountering reserved resources, and monitoring progress.

- **Data Management:** If an application is running on a system that doesn't hold the data the application needs, a secure, reliable data management facility takes care of moving that data to the right place across various machines, encountering various protocols.

- **Resource Management:** To handle core tasks such as discovering resources, monitoring resource status, launching jobs with specific resources, monitoring the status of those jobs, and retrieving results, a resource management facility is necessary.

1.5 Motivation

Why should we be interested in **Resource Management over a GRID?** A resource on a grid could be any entity that provides access to a service. This could range from compute servers to databases, scientific instruments and applications. In a heterogeneous environment like a grid, resources are generally owned by different people, communities or organizations with varied administrative policies, and capabilities. Naturally, obtaining and managing access to these resources is not a simple task.

The aim of resource management is to simplify the process of handling the resources used by users to complete their task. In the grid computing environment resource management is a collection of software components that allow users to access

heterogeneous resources transparently, without having to worry about availability, access methods, security issues and other policies.

1.6 Objectives

In a grid computing environment, as resources are widely dispersed, there is a possibility that resources can go down at any time. Dynamic resource addition, deletion and modification are possible. There are separate administrative policies across organizations in managing resources. We need to address the following questions, to use grid resources:

- How to know where the resources are?
- How to identify resources?
- How to get permission to use them?
- How to know what are the applications available in these resources?
- How to use the resources?
- How to submit remote jobs?
- How to get access to resources to all the machines simultaneously?
- What happens if a resource fails?
- How input/output files are managed?

To address these questions an efficient resource management system is required. Current resource management systems in grid computing, suffer from several limitations. They need a dedicated resource management monitoring station, which must be on a specific type of platform. Access to resource management system remote locations is accomplished by using adhoc schemes such as X-host applications in Linux based resource management system. This permits export of the presentation and running the management system from a remote workstation. However, the remote workstation also needs to be a Linux system.

The information provided by the existing systems is required but not sufficient because they lack in the topology information and integrated way of looking at various tasks involved in management of grid resources. These problems are addressed in our

Grid Management Information Server (GMIS) framework. In addition, questions on communication issues, computation issues, performance issues, subscription to the service data, etc., are also addressed in this framework.

The main objective of this research work is providing an efficient resource management framework for Discovery, Monitoring and Brokerage of resources in grid computing environment. Discovery, Monitoring and Brokerage are vital functions in a grid computing environment, particularly when that system spans multiple locations, as in that context no one is likely to have detailed knowledge of all components. Discovery allows us to identify resources or services with desired properties, while Monitoring allows us to detect and diagnose many problems that can arise in such contexts. Brokerage allows users to submit jobs over grid. These tasks require the ability to collect information from multiple, perhaps distributed information sources.

To summarize, Resource Discovery, Monitoring and Brokerage in grid are required for the following main reasons:

- To get the information about resources health status
- To construct and execute applications
- To describe grid resources
- For resource scheduling, allocation and usage

Following are the important design considerations made in developing GMIS framework.

- **Seamless and transparent user access to resource information** – All information related to resources in a grid can be accessed by the user by selecting options in the viewer without any need of understanding of the grid infrastructure.
- **Easy to use web-based interface** – The viewer is web based and platform independent. User can access the viewer from any operating system and web browser. The information and controls available in the viewer are presented to the user in an intuitive way, so that the user can perform all his activities with ease.

- **Collation of information from multiple resources** – The information of resources presented under various grids or managed by different GMIS can be viewed by the user within a single Viewer.
- **Efficient resource management** - In order to gain maximum benefit from resources, the GMIS provides appropriate resources based on the job requirements. GMIS maintains latest resource information along with their status.
- **Scalable and Adaptable framework** - GMIS doesn't impose any kind of limitations on the grid or jobs submitted by the user. If large number of resources/jobs is to be managed then only hardware needs to be upgraded. GMIS can be integrated with other grid technologies like Globus by agreeing on data exchange formats.
- **Poll and Event based information gathering mechanisms** – The information about resources will be gathered using poll mechanism where GMIS send a request to resource agent. In case of events mechanism, the resources agents automatically send information to GMIS whenever there is change in resource status.

1.7 Contributions

The GMIS framework is developed as part of development of Grid Middleware for Integration of Dispersed Resources at University of Hyderabad (UH), Hyderabad. We claim that this is a parallel effort along with peers who are working in this domain. In this development already a scheduler [Pramod Kumar, 2006] is developed which can be used in GMIS framework to submit jobs.

Currently UH Grid group is working on various research areas of grid computing. Some of them are Resource Management in Grid Computing environment, Schedulers, High Availability Grids, High Capacity High Latency Transport Protocols for Grid Computing, Geo Science applications on Grid and Heterogeneous Distributed Shared Memory Computing.

Recent contributions made by this group are, developed an algorithm for high availability of data in grids [Chidambaram, 2008], Extending OGSA DQP framework for

high availability [Narasimha Rao, 2008], Design and Implementation of a structured fine-grained access control mechanism for authorizing grid resources [Mustafa, 2008], Web Grid Services for Ocean Science Applications [Srinivas, 2008], Dynamic Ants Resource Discovery System for a Grid [Naveen, 2008], Design of a Grid Framework for Scientific Applications [S. Baburao, 2008], An efficient design method for searching resources in multiple grid portal nodes [Utla Raju, 2008], Adding Meta-scheduling to Portable Batch Scheduler (PBS) [Hrudayanath, 2008].

Following is the summary of main contributions of the thesis:

1. Proposed distributed framework architecture called Grid Management Information Server (GMIS) for resource management in grid environment, which is a robust integrated comprehensive framework, compared to available frameworks.
2. Identified key requirements and design considerations that GMIS framework needs to support.
3. Compared and contrasted GMIS framework with the traditional resource management frameworks.
4. Developed a scalable and adaptable GMIS framework which can manage resources that are distributed over a Wide Area Network (WAN). One instance of GMIS can communicate with another instance of GMIS so that the user can get a seamless view of available distributed resources over the WAN.
5. Designed resource discovery algorithms with three different strategies i.e., dynamic, event based and user based resource discovery.
6. Designed topology manager service, for maintaining resource information as well as support for virtual clustering where users can group the resources any where as a single logical/virtual cluster.
7. Developed monitoring and data collection manager services to provide users with wide range of options to select resources based on their capabilities and historical performance.
8. Developed push and pull based algorithms for gathering resource information, so that grid user will have latest status about the resources as well as submitted jobs.

9. GMIS provides support for activation graph creation based job submission and execution i.e., submitted jobs can be divided into multiple tasks and results can be analyzed at each task level.
10. GMIS is developed in such a way that to provide all necessary infrastructure services and environment for supporting MDO framework.
11. Developed agent software that resides on each resource to respond to the GMIS requests and to generate events on behalf of resources.
12. Developed a single intuitive web based interface to provide access to all GMIS services in an integrated fashion.

1.8 Thesis Organization

Chapter 2 presents overview of Grid computing and extensive literature survey on basic grid technologies and current grid activities. It discusses about resource management and some of the available resource management frameworks in detail. It gives a comparative study of GMIS framework with other resource management frameworks.

Chapter 3 proposes details of the framework called Grid Management Information Server (GMIS) architecture for resource management in grid environment for effective management of distributed resources. This framework architecture addresses the resource management requirements like Discovering, Monitoring and Brokerage of resources in grid environment.

Chapter 4 emphasizes on the design aspects of GMIS framework with data flow diagrams and data models. Algorithms for various tasks like discovery, polling and parsing, event handling are discussed in detail. It discusses how job execution can be done using GMIS framework. Message passing among GMIS components and communication between resources' agent software and GMIS through communicator server is also discussed in this chapter.

Chapter 5 discusses about GMIS processes and interface modeling details of GMIS framework along with Graphical User Interfaces. It also discusses about testing environment for GMIS. Some experiments and results are presented in this chapter.

Chapter 6 presents case studies of GMIS framework. Multidisciplinary Design Optimization (MDO) application is being looked at as a case study for the GMIS framework. This chapter gives literature survey on MDO, UH-MDO framework over grid and then shows how MDO framework can use GMIS framework. GMIS framework can be applied for virtual clustering. This also presented in this chapter.

Chapter 7 concludes the thesis and proposes potential future work areas.

CHAPTER 2

A Survey on Grid Technologies and Resource Management Systems

This chapter presents an overview of grid computing, basic grid technologies and current grid activities. It discusses about resource management and some of the available resource management frameworks. It gives a comparative study of GMIS framework with other resource management frameworks.

2.1 Overview of Grid Computing

The term grid was coined to define the idea of exploiting the considerable power represented by the number of computers on the planet and the existence of networks to interconnect them, in order to carry out very long computations for workgroups who need brute force to complete their work, such as auto and airplane designers, scientific researchers, or drug-research firms. To describe it, picture the electricity supply via the electrical grid. Similarly, if we need lots of computing power, we should be able to tap into those resources without worrying about how those resources are generated. Simply plug into the grid and compute.

Grid computing enables the virtualization of distributed computing and data resources such as processing, network bandwidth and storage capacity to create a single system image, granting users and applications seamless access to vast IT capabilities. Just as an Internet user views a unified instance of content via the Web, a grid user essentially see a single, large virtual computer.

Grid computing is based on an open set of standards and protocols e.g., Open Grid Services Architecture (OGSA) that enables communication across heterogeneous, geographically dispersed environments. With grid computing, organizations can optimize computing and data resources, pool them for large capacity workloads, share them across networks and enable collaboration.

Grid computing appears to be a promising trend for three reasons:

(i) Its ability to make more cost-effective use of a given amount of computer resources

(ii) As a way to solve problems that can't be approached without an enormous amount of computing power

(iii) It suggests that the resources of many computers can be cooperatively and perhaps synergistically harnessed and managed as collaboration towards a common objective.

Benefits of grid computing:

- Infrastructure optimization
- Increase access to data and collaboration
- Highly available infrastructure
- Accelerate time to results
- Enable collaboration and promote operational flexibility

To further illustrate this environment and often times very complex set of technology challenges, let us consider a scenario, which will begin to examine the many values of a grid computing solution environment.

A group of scientists studying the atmospheric ozone layer collect huge amounts of experimental data, each day. These scientists need efficient and complex data storage capabilities across wide and geographically dispersed storage facilities, and they need to access this data in an efficient manner based on the processing needs. This ultimately results in a more effective means of performing important scientific research. Grid offers a very lucrative solution for this type of problems.

What is not a grid? A cluster, a network attached storage device, a scientific instrument, a network; these are not grids. Each might be an important component of a grid, but by itself, it doesn't constitute a grid.

2.1.1 Ancestors of the Grid

Although the grid can trace its roots back to the operating system Multics [Multics], the immediate ancestor of the Gird is “meta computing” [Meta computing], a term that dates back to around 1990, and was used to describe projects aimed at interconnecting supercomputer centers in order to combine the processing power of multiple supercomputers. Two cutting edge Meta computing projects both conceived in 1995 were called Factoring via Network-Enabled Recursion (FAFNER) [FAFNER] and Information Wide Area Year (I-WAY) [IWAY, 1996]. Each in its own way influenced the evolution of some of the key grid technology projects ongoing today.

FAFNER was a project, which aimed at factorizing very large numbers, a computationally intensive challenge which is very relevant to digital security. Since this is a challenge which can be broken into small parts, even fairly modest computers can contribute useful power. Many of the techniques developed for splitting up and distributing a big computational problem were forerunners of the technology used for “cycle scavenging” software.

I-WAY was a project that aimed at linking supercomputers, but using only existing networks, not building new ones. One of the innovations of the I-WAY project was developing a computational resource broker, conceptually very similar to the resource brokers being developed for the grid today. I-WAY strongly influenced the development of the Globus project, which is at the core of most grid activities, as well as the LEGION [Grimshaw, 1997] project, an alternative approach to distributed super computing.

The grid itself was born at a workshop at Argonne National Laboratory in 1997, called “Building a Computational Grid”, which was followed by the publication of “The Grid: Blueprint for a New Computing Infrastructure” by Ian Foster of Argonne National Laboratory and Carl Kesselman of the University of Southern California [Ian Foster, Carl Kesselman, 1999]. Ian Foster is often known as the father of grid computing.

2.1.2 Architecture of Grid Computing

The architecture for grid computing is defined in the Open Grid Services Architecture (OGSA), developed through the Global Grid Forum (GGF) [GGF]. OGSA defines what grid services are and the overall structure and services to be provided in grid environments.

The Open Grid Services Infrastructure (OGSI) is a formal specification of the concepts described by the OGSA. More than two dozen working groups at the Global Grid Forum are busy defining an array of grid standards in areas like applications and programming models, architecture, data management, security, performance, and scheduling and resource managements.

The architecture of the grid is described in terms of layers, each providing a specific function. In general, the higher layers are focused on the user (user-centric) whereas the lower layers are more focused on computers and networks (hardware-centric). The OGSA grid architecture is shown in the following figure 2.1.

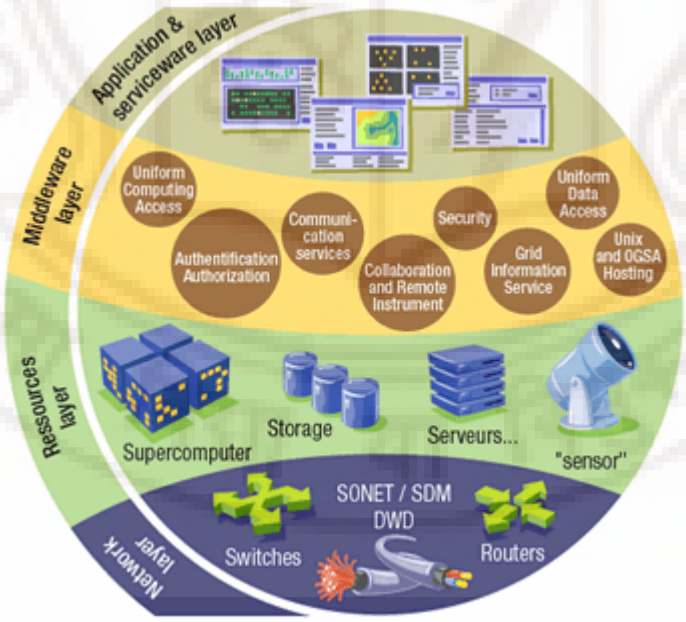

Figure 2.1: OGSA Grid Architecture [Foster, I, C. Kesselman, 2002]

As shown in above figure 2.1, at the base of everything, the bottom layer is the network layer, which assures the connectivity for the resources in the grid. On top of it lies the resource layer, made up of the actual resources that are part of the grid, such as computers, storage systems, electronic data catalogues, which can be connected directly to the network.

The middleware layer provides the tools that enable the various elements (servers, storage, networks, etc.) to participate in a unified grid environment. The middleware layer can be thought of as the intelligence that brings the various elements together.

The highest layer of the structure is the application layer, which includes all different user applications (science, engineering, business, financial), portals and development toolkits supporting the applications.

In most common grid architectures, the application layer also provides the serviceware, the sort of general management functions such as measuring the amount a particular user employs the grid, billing for this use, and keeping accounts of who is providing resources and who is using them - an important activity when sharing the resources of a variety of institutions amongst large numbers of different users. The serviceware is in the top layer, because it is something the user actually interacts with, whereas the middleware is a "hidden" layer which is transparent to the user. In order to achieve the benefits of grid computing the applications architecture need to consider grid related services and protocols available. If traditional applications that run on a single PC or server have to run in grid environment then they should be adapted to grid environments which require users to invest some effort. How ever once traditional applications are adapted to grid environment thousands of people will be able to use the same application and run in trouble free on this grid.

2.2 Basic Grid Technologies

The grid is inherently very complicated. Factors that contribute to this complication include many network types, incompatible hardware architectures, different operating system security mechanisms and different protocol support in many

programming languages. Hence, the solution to create a grid is understandably complicated and is a nontrivial task. Many efforts have been made to construct a homogeneous view of this heterogeneous environment. Some of such popular efforts are discussed in this section.

2.2.1 Globus Toolkit

Globus toolkit [Foster, I, C. Kesselman, 2001] [Foster, I, C. Kesselman, 2002] is a free and open source grid toolkit. It implements the Open Grid Services Infrastructure (OGSI) standards and provides tools to build, develop, deploy, and manage grid services. The open source Globus Toolkit is a fundamental enabling technology for the "Grid," letting people share computing power, databases, and other tools securely online across corporate, institutional, and geographic boundaries without sacrificing local autonomy.

The toolkit includes software for security, information infrastructure, resource management, data management, communication, fault detection, and portability. It is packaged as a set of components that can be used either independently or together to develop applications. Every organization has unique modes of operation, and collaboration between multiple organizations is hindered by incompatibility of resources such as data archives, computers, and networks. The Globus toolkit was conceived to remove obstacles that prevent seamless collaboration. Its core services, interfaces and protocols allow users to access remote resources as if they were located within their own machine room while simultaneously preserving local control over whom and when these resources can be used.

Grid tools that are available in the toolkit can be broadly classified into the following:

- **Infrastructure:** Infrastructure components include file systems, schedulers and resource managers, messaging systems, security applications, certificate authorities, and file transfer mechanisms like GridFTP.

- **Directory Services:** Systems on a grid must be capable of discovering what services are available to them. This means that the grid systems must be able to define and monitor a grid's topology in order to share and collaborate. Many grid directory services implementations are based on past successful models, such as Lightweight Directory Access Protocol (LDAP) [OpenLdap], Domain Name Service (DNS), network management protocols, and indexing services.

- **Schedulers and load balancers:** One of the main benefits of a grid is its ability to maximize efficiency. Schedulers and load balancers provide this function and more. Schedulers ensure that jobs are completed in some order (for instance priority, deadline, priority) and load balancers distribute tasks and data management across systems to decrease the chance of bottlenecks.

- **Security:** Security in a grid environment provides authentication and authorization, controlling who can access a grid's resources. For instance, message integrity and message confidentiality are crucial to financial and healthcare environments.

The Globus toolkit has grown through an open-source strategy similar to the Linux operating systems, and distinct from proprietary attempts at resource-sharing software. This encourages broader, more rapid adoption and leads to greater technical innovation, as the open-source community provides continual enhancements to the product.

The main drawback of Globus toolkit is that it is a very complicated toolkit. While it is a feature rich, it can be very time consuming to configure, install and operate. Usually it takes several hours (approximately 7 hours) to install. It is unstable on some platforms and provides only a base support e.g., Sun Solaris. Globus toolkit is known to have better performance on SuSE or RedHat Linux operating systems.

2.2.2 IBM Grid Tool box

IBM Grid Toolbox, [IBM Redbook, 2004] developed by IBM, is a free grid Integration Development Environment (IDE) and implements OGSI standards. The major

advantage of this toolbox is that its installation process is very simple. It ships both wizard-based and silent installation methods. Users are relieved from the pain to install prerequisite packages like in other open source software. The IBM Grid Toolbox installation on Pentium-based Personal Computer (PC) or xSeries takes around 40 minutes. The tool was built entirely based on the OGSI specification.

The toolbox is provided with enhancements and tools which assist with building, packaging and deploying grid services and applications. This feature greatly reduces the complexity of grid services development and deployment. It consists of the following:

- A hosting environment capable of running grid services and collaborating with other grid participants in running large tasks.
- A set of tools to manage, monitor, and administer grid services and the grid hosting environment, including a web-based interface called IBM Grid Services Manager.
- A set of APIs and development tools to create and deploy new grid services and grid applications.
- A set of tools to simplify the installation process and the integration of the embedded middleware, such as IBM WebSphere Application Server.

The IBM Grid Toolbox is a collection of components. The following overview of components suggests the scope of features that the IBM Grid Toolbox brings to grid developers and administrators:

- **Single installation process:** The IBM Grid Toolbox ships both wizard-based and silent installation methods, which provide simple one-off or bulk installation around a network.

- **Grid services runtime based on the OGSI specification:** An embedded version of the IBM WebSphere Application Server – Express V5.0.2 is provided as the grid services container. It replaces the stand-alone container that is provided by Globus toolkit for commercial-grade support.

- **Management interface:** A browser-based interface called the IBM Grid Services Manager provides easy grid-wide management for administrators.

- **Security:** An enhanced certificate-based grid security infrastructure.

- **Configuration and administration commands:** Command-line based scripts for common actions are provided for administrators.

- **Development tools:** Enhancements and tools are provided that assist with building, packaging, and deploying grid services and applications.

- **Additional and enhanced grid services:** IBM provides additional functionality including discovery via service group, policy management, and Common Management Models (CMM) Services.

2.2.3 SUN Grid Engine

SUN's Grid solution, SUN Grid Engine (SGE) [SUN Grid] is an open source but not free grid IDE and does not implement OGSI standards. SGE contains four types of hosts: Master hosts, Execution hosts, Administration hosts, and Submit hosts. The Master host runs master and scheduler daemons, which control all SGE components, such as queues and jobs. Execution hosts have permission to execute SGE jobs. Administration hosts carry out administrative activity for the grid system. Submit hosts allow for submitting and controlling batch jobs only. For example, a logged in user can submit jobs via "qsub" and control job status via "qstat". Besides the command-line interface, SGE system also provides graphical user interface, called "QMON".

SGE allows users to submit jobs to queues. Master and scheduler daemons determine which execution host requires to run the tasks. In case some execution hosts crash, SGE has the facility of using other execution hosts to fail over transparently, where as the other open source tools don't have this facility. In order to achieve reliability and scalability for a grid, a SGE could be used as a grid scheduler.

SGE provides the following features [SGE Features]:

- **Multi-clustering with Service Domain Manager:** The SGE Service Domain Manager allows two or more SGE clusters to share resources. It offers optimal service level to users. It delivers built-in elasticity by adding or removing hosts from each local cluster according to service level objectives.

- **Advance Reservation:** In SGE users can request and reserve grid resources in advance, such as hosts, memory, or licenses, for a specified time window.

- **Scalability:** SGE scales up to 63,000 core CPUs.

- **Improved Interactive Job Support:** SGE enables better accounting and monitoring of the interactive users in the grid. It provides a faster and more robust connection.

- **Support massively parallel jobs:** SGE supports robust parallel jobs spanning tens of thousands of CPU cores. It is faster in start-up of large parallel jobs.

- **Array Task Dependency:** SGE allows for faster production times. It enables special effects and rendering to be done faster than with competitive software.

- **Accounting and Reporting Console:** SGE provides a clear report of the usage of all SGE clusters throughout the entire organization.

- **Better Integration with Sun Software:** SGE provides better manageability and quality of service when the SGE software runs on the Solaris 10 operating system.

SGE is a system for cluster management, rather than real grid resource management. It does not implement OGSI standards, nor support grid services. Unlike IBM Grid Toolbox, SGE could not allow grid or web services deployment, but only support script-based job management.

2.2.4 Avaki

Avaki [IBM Redbook, 2003] is a commercial grid IDE developed by Avaki inc. Ltd. Avaki was formed to commercialize the technology derived from the Legion project [Grimshaw, 1997]. It implements the OGSI standards and provides the tools to build, develop, deploy, and manage grid services. The AVAKI grid solution pools heterogeneous resources across an enterprise, making them easily accessible on-demand. With AVAKI grids, research and development organizations can overcome problems such as, the need for secure access to fresh and consistent data; a uniform environment that is easy for researchers to use and for IT professionals to administer and a highly available computing infrastructure that delivers maximum application performance and reliability.

At a high level, Avaki's vision is one of a ubiquitous, pervasive computing infrastructure, one that enables secure and transparent access to all computing resources. To that end, Avaki delivers grid software solutions, also known as middleware, that bring together data, compute, and application resources from multiple locations and administrative domains for industries such as life sciences, manufacturing, and oil & gas. They have products for both data grids and compute grids, but are focusing a majority of their technology on data grids (80 percent of the projects they're working on these days are data grid related projects). In addition, although they solve problems across verticals, they are focused primarily on domestic pharmaceutical companies.

Avaki's products include Avaki Data Grid, Avaki Compute Grid, and Avaki Comprehensive Grid.

Avaki Data Grid provides wide-area access to distributed data, making it easily available for users, applications, and business process. Data grid also manages security and usage policies such as enabling multi-company data access and collaboration which are key to the forward-looking vision of virtual organizations. A data grid, with its distributed nature, also enhances remote processing by efficiently bringing the data closer to the application.

Avaki Compute Grid aggregates processing resources within or across locations, leveraging existing infrastructure. Authentication, security, and usage policies for compute grid are similar to those for data grid. Job scheduling and queuing are also important characteristics of compute grid, aiding in the processing of multiple jobs over a common and shared processing infrastructure.

Avaki Comprehensive Grid combines features of compute grid and data grid to provide wide-area access to processing resources, data, and applications.

The components of the Avaki solution include:

- **Grid Servers** contain house authorization information. Each grid domain has a grid server called a Grid Domain Controller
- **Share Servers** authorize sharing of data on the grid
- **Data Grid Access Server** for High-performance caching
- **Proxy Server** is for Firewall proxy
- **Client Components** like Web User Interface, command-line client

Much of the underlying protocol and messaging is done via Hyper Text Transfer Protocol (HTTP) and Secure Socket Layer (SSL). Remote Procedure Calls provide credentials to the access control lists.

One of the keys to Avaki's technology is its intelligent caching approach. This is key in keeping the authoritative copy of the data with the owner, while still allowing sharing and access with remote users for very large data sets. A simplistic view of Avaki is as, basically, a distributed database solution. Avaki is soon to announce a deal with a top 10 pharmaceutical that will be one of the largest data grid deployments to date (over 500 collaborations with multi-tier partnership relationships involved with outsourcing pharmaceutical processes). Due to the data-intensive nature of Avaki's solution, their primary opportunities are with enterprise grids and wide area grids.

2.2.5 Microsoft .NET

The Microsoft .NET [Microsoft .NET] platform provides a useful way to construct grid systems that make use of the emerging interest in web services. Various tools are available for reducing the complexity of implementing web services, and providing support for messaging between such services. For instance, Visual Studio .NET provides a development environment for writing such services in C# and J# (Microsoft's version of Java). The importance of such tools is essential to enable a wider adoption and usage of web services-oriented grid software.

The .NET platforms also support a number of additional features that could be useful to construct grid systems, particularly for enabling a number of different platforms to interact. The "Common Language Runtime" (CLR) in .NET provides an execution environment that can be ported to a number of different machine architectures, and supports the execution of programs written in a variety of different languages. The CLR adopts a similar approach to the Java Virtual Machine, for enabling code developed on one machine to be ported (without re-compilation) to another. The .NET platform also provides a Just-In-Time (JIT) compiler that allows dynamic performance improvements during the execution of a program, and is facilitated in this by the CLR. Microsoft is also aiming to make a version of the CLR open-source, to enable community input into this activity.

2.2.6 Comparison of Various Grid Technologies

The below table 2.1 shows a side-by-side comparison of above discussed grid technologies:

	Globus	**IBM Grid Toolbox**	**SUN Grid Engine**	**Avaki**	**Microsoft .NET**
Open Source	Yes	No	Yes	No	No
Price	Free	Free	Commercial	Commercial	Commercial
OGSI Standards	Support	Support	No	Support	Support
Grid services	Yes	No	No	Yes	Yes
Platforms	Linux, All Unix	Linux, AIX	Linux, Solaris	Linux, Windows	Windows, All UNIX

Installation	Not Easy	Easy	Easy	Easy	Easy
Development Tools	Some Scripts	Enhanced Tooling	No	No	No

Table 2.1: Comparison of Different Grid Technologies

2.3 Current Grid Activities

There are innumerous grid activities going on in the world. Among them, some concentrate on defining standards, some on development of software tools, and others on services, or different scientific applications. Following is a brief mention of some of the important ones:

- **Grid Standards Projects** – focusing on refining the grid standardization process and defining best practice guidelines for the scientific and industry usage of grid. The most prominent among such organizations is Global Grid Forum (GGF).

- **Grid-Tech Projects** - primarily involved in development of grid-enabling technology, such as middleware and hardware. Some examples are CONDOR, BIOGRID etc.

- **Test Bed Projects** - devoted to develop and maintain a working test beds using existing grid technology like European Union: DataGrid, EUROGRID etc.

- **Grid Fora Projects** - devoted to catalyze, stimulate and foster collaboration on grid related projects. They are aiming to share technology, resources and knowledge in order to promote grid technologies and applications like PRAGMA.

- **Field specific Projects** - projects devoted to explore and harness grid technology in the context of specific fields of scientific research such as BIOINFORMATICS and E-SCIENCE program.

- **Grid Portals** - Internet portals to grid related activities. One such portal is EnterTheGrid which is an extensive dictionary to grid related activities.

- **...@home** - Internet computing projects such as Search for Extra Terrestrial Intelligence (SETI) experiment uses Internet distributed computing to search for intelligent life outside Earth.

Some popular grid projects like GGF, CONDOR, European Union DataGrid, PRAGMA, and Grid Bus are briefly discussed in subsequent sections.

2.3.1 Global Grid Forum

Global Grid Form (GGF) [GGF] was established as a public community forum for the discussion of grid technology issues. The GGF enables a means of coordinating grid computing technology efforts, promoting reuse and interoperability, and sharing the results. As of now, there are more than 400 organizations involved with GGF from around the world. This includes scientific research institutions, universities, and commercial organizations.

The GGF's primary objective is to promote and support development, deployment, and implementation of grid technologies and applications via creation and documentation of best practices – specifications, use cases, architecture, and implementation guidelines.

The basic goals of the GGF are to:

- Create an open process for the development of grid agreements and specifications
- Create grid specifications, architecture documents, and best practice guidelines
- Manage and version controls the documents and specifications
- Handle intellectual property policies
- Provide a forum for information exchange and collaboration
- Improve collaboration among the people involved with grid research, grid framework builders, grid deployment, and grid users

- Create best practice guidelines from the experience of the technologies associated with grid computing
- Educate on advances in the grid technologies and share experiences among the people of interest

The organization consists of different work areas, with research groups and work groups for each area. The work groups are the main activity centers of the GGF. These work groups are created to address a research, implementation, and operational area related to the infrastructure for building any grid.

The major work areas of the GGF are:

- Application and programming environments
- Architecture
- Data
- Information systems and performance
- Peer-to-peer: desktop grids
- Scheduling and resource management
- Security

One of the major activities in GGF that is attracting the grid community is the architecture model based on the open standard web service architecture, called Open Grid Service Architecture (OGSA). With open standards as the foundation and software integration, OGSA has emerged as the core grid technology for future resource sharing, especially with the newly added commercial dimension to grid solutions.

2.3.2 Condor

Condor [Condor] is a tool for harnessing the capacity of idle workstations for computational tasks. Condor is well suited for parameter studies and high throughput computing, where jobs generally do not need to communicate with each other.

Condor can be classified as a specialized workload management system for computation-intensive jobs. Like other full-featured batch systems, Condor provides a job queuing mechanism, scheduling policy, priority scheme, resource monitoring, and

resource management. Upon receiving serial or parallel jobs from the user, the Condor system places them into a queue, chooses when and where to run the jobs based upon a policy, carefully monitors their progress, and ultimately informs the user upon completion.

Condor can manage a cluster of dedicated compute nodes. It is suitable for effectively harnessing the CPU power from idle workstations. Condor has mechanisms for matching resource requests (jobs) with resource offers (machines).

While Condor software tools focus on harnessing the power of opportunistic and dedicated resources, Condor-G is a derivative software system, which leverages the software from Condor and Globus with major focus on the job management services for grid applications. This is a combination of inter-domain resource management protocols of Globus (GRAM, Index services) with the intra-domain resource management methods of Condor.

Condor software is used by both scientific and commercial organizations. The major scientific initiative that uses Condor includes NSF Middleware Initiative (NMI), Grid Physics Network (GriPhyN), International Virtual Data Grid laboratory (iVDGL), TeraGrid, and so on. Some of the prominent commercial uses of Condor software involve solving computational grid computing problems, as done by Micron Technologies, CORE Digital Pictures, and NUG30 Optimization problem solver.

2.3.3 European Union: DataGrid

DataGrid [DataGrid, 2002] is project funded by the European Union that aims to enable access to geographically distributed computing power and storage facilities belonging to different institutions. This will provide the necessary resources to process huge amounts of data coming from scientific experiments in different disciplines.

The three real data-intensive computing applications areas covered by the project are:

- High Energy Physics
- Biology and Medical Image Processing

- Earth Observations

High Energy Physics: One of the main challenges for High Energy Physics is to answer long standing questions about the fundamental particles of matter and the forces acting between them. In particular, the goal is to explain why some particles are much heavier than others, and why particles have mass at all. To that end, CERN is building the Large Hadron Collider (LHC), on of the most powerful particle accelerators.

The serial on LHC will generate huge amount of data. The DataGrid project is providing the solution for storing and processing this data. A multi-tiered, hierarchical computing model is adopted to share data and computing power among multiple institutions. The Tier-0 center is located at CERN and linked by high-speed networks to approximately 10 major Tier-1 data processing centers. These will fan out the data to a large number of smaller ones (Tier-2).

Biology and Medical Image Processing: The storage and exploitation of genomes and the huge flux of data coming from post-genomics puts growing pressure on computing and storage resources within existing physical laboratories. Medical images are currently distributed over medical image production sites (radiology departments, hospitals). As of today, since there is no standard for sharing data between sites, there is an increasing need for remote medical data access and processing.

The DataGrid project's biology test-bed is providing the platform for the development of new algorithms on data mining, databases, code management, and graphical interface tools. It is facilitating the sharing of genomic and medical imaging databases for the benefit of international cooperation and health care.

Earth Observations: The European Space Agency missions download 100 gigabytes of raw images per day from space. Dedicated ground infrastructures have been set up to handle the data produced by instruments onboard the satellites. The analysis of atmospheric ozone data has been selected as a specific test-bed for the DataGrid. Moreover, the project demonstrates an improved way to access and process large volumes of data stored in distributed European-wide archives.

2.3.4 Pacific Rim Application and Grid Middleware Assembly (PRAGMA)

The Pacific Rim Application and Grid Middleware Assembly (PRAGMA) [PRAGMA] was formed to establish sustained collaborations and advance the use of the computational grid among a community of investigators at the leading institutions around the Pacific Rim. Applications are the key focus of PRAGMA, with the intent of using applications to bring together the key infrastructure and middleware necessary to advance the goals of the application. **University of Hyderabad** has become one of the founding partners of such an international collaboration.

PRAGMA accomplishes its mission primarily by,

- conducting joint projects that develop grid middleware to advance applications,
- sharing resources to create a test bed,
- addressing scheduling and allocation issues across institutional and international boundaries.

In addition, PRAGMA is committed to disseminating the results of its efforts to the broader community and to work with regional and international groups to enhance the overall grid infrastructure and to promote global collaboration.

2.3.5 Grid Bus

The Gridbus project [Buyya, VenuGopal, 2004] is engaged in the design and development of grid middleware technologies to support eScience [T.Hey, A.E. Trefethen, 2002] and eBusiness applications. These include visual grid application development tools for rapid creation of distributed applications, competitive economy-based grid scheduler, cooperative economy based cluster scheduler, web-services based Grid Market Directory (GMD), grid accounting services, Gridscape for creation of dynamic and interactive test-bed portals, G-monitor portal for web-based management of grid applications execution, and the widely used GridSim toolkit for performance evaluation. Gridbus project has developed Windows .NET-based desktop clustering software and grid job web services to support the integration of both Windows and Unix-

class resources for grid computing. A layered architecture for realization of low-level and high-level grid technologies is shown in figure 2.2.

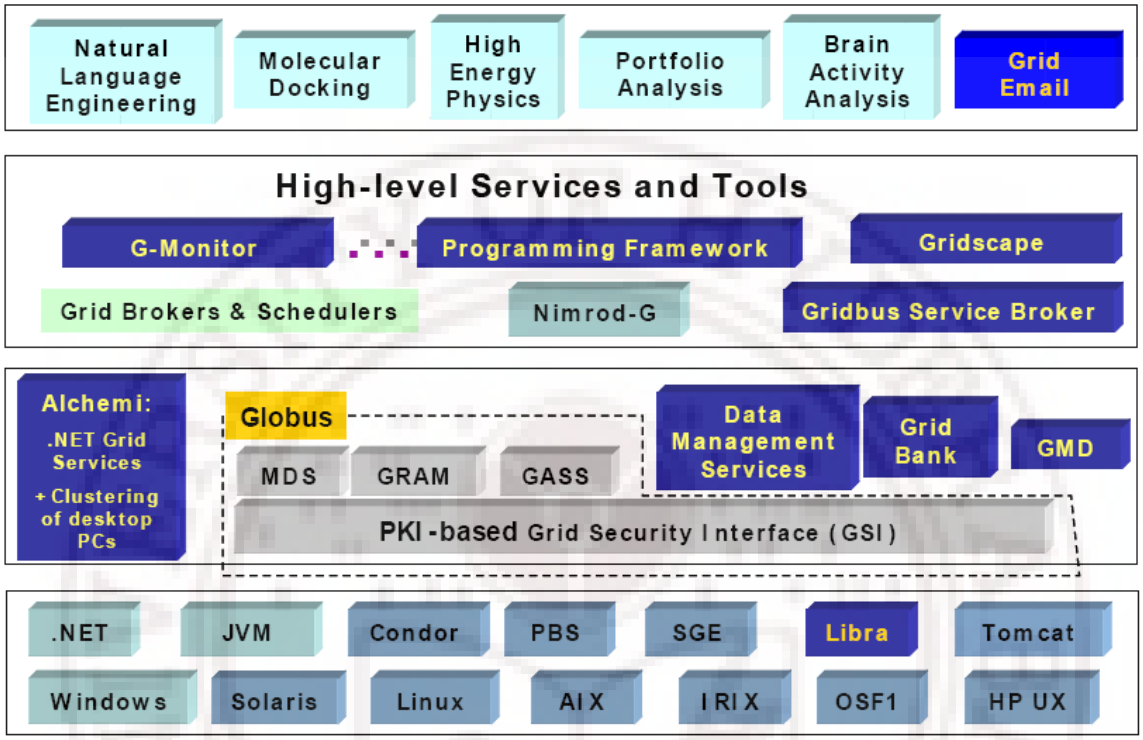

Figure 2.2: A Grid technology stack (items in the dark blue box are pursued by the Gridbus project) [Buyya, VenuGopal, 2004]

GridBus contain following components:

- **Visual Parametric Modeler:** A graphical environment for application parameterization.

- **G-Monitor:** A web portal to manage execution of applications on grids using remote brokers.

- **Grid Service Broker:** An economy-based data grid broker for scheduling distributed data oriented applications across Windows and Unix-variant grid resources.

- **Grid Market Directory:** A directory for publication of grid service provides and their services.

- **Grid Bank:** A grid accounting, authentication, and payment management infrastructure.

- **Gridscape:** A tool for the creation of interactive and dynamic grid test bed web portals.

- **Alchemi:** A .NET-based desktop grid framework.

- **Libra:** An economy based scheduler for clusters.

- **GridSim:** A toolkit for modeling and simulation of global grids.

2.4 Resource Management

A resource on a grid could be any entity that provides access to a service. This could range from compute servers to databases, scientific instruments and applications. In a heterogeneous environment like a grid, resources are generally owned by different people, communities or organizations with varied administrative policies, and capabilities. Resource management simplifies the process of naturally obtaining and managing access to resources to all its users. In the grid computing environment resource management is a collection of software components that let users to access heterogeneous resources transparently, without having to worry about availability, access methods, security issues and other policies.

Consider a job management system, where the resource management feature identifies the job, allocates the suitable resources for the execution of the job, partitions the job if necessary, and provides feedback to the user on job status. This job scheduling process includes moving the data needed for various computations to the appropriate grid computing resources, and mechanisms for dispatching the job results.

It is important to understand multiple service providers can host grid computing resources across many domains, such as security, management, networking services, and application functionalities. Operational and application resources may also be hosted on different hardware and software platforms. In addition to this complexity, grid

computing middleware must provide efficient monitoring of resources to collect the required matrices on utilization, availability, and other required information.

The following brief and partial list provides a resource specific characterization of capabilities.

- ***Computational resources management*:** Mechanisms are required for starting programs for monitoring and controlling the execution of the resulting processes. Management mechanisms that allow control over the resources allocated to processes are useful, as are advance reservation mechanisms. Enquiry functions are needed for determining hardware and software characteristics as well as relevant state information such as current load and queue state in the case of scheduler-managed resources.

- ***Storage resources management*:** Mechanisms are required for storing and retrieving of files. Third-party and high-performance (e.g., striped) transfers are required. These mechanisms used for reading and writing subsets of a file and/or executing remote data selection or reduction functions. Management mechanisms that allow control over the resources allocated to data transfers (space, disk bandwidth, network bandwidth, CPU) are required. Enquiry functions are needed for determining hardware and software characteristics as well as relevant load information such as available space and bandwidth utilization.

- ***Network resources management*:** Management mechanisms that provide control over the resources allocated to network transfers (e.g., prioritization, reservation) can be useful. Enquiry functions should be provided to determine network characteristics and load.

- ***Code repositories management*:** Code repositories are specialized form of storage resources require mechanisms for managing versioned source and object code: for example, a control system such as Concurrent Versions System (CVS).

- ***Catalogs management*:** Catalogs are specialized form of storage resources that require mechanisms for implementing catalog query and update operations: for example, a relational database.

Usually the resource management scenarios include resource discovery, monitoring and brokerage.

2.4.1 Resource Discovery

For resource sharing and integration in grid computing environment, the abilities of discovering, allocating, negotiating, monitoring and managing network accessible resources are essential to achieve various end to end or global qualities of services. Thus grid resource discovery becomes a basic component of grid resource management, which is a core of grid and provides a transparent global resource view for the resource schedule and application.

In a grid system, the resource discovery service operates in conjunction with the resource management service. Resource discovery is the process of identifying and locating existing resources on grid. It is used to search information about resources available in the grid network. Schedulers or brokerage service use the resources information discovered by the resource discovery service.

2.4.2 Resource Monitoring

In order to construct and execute applications on grid resources, an understanding of the type and availability of grid resources is necessary. Grid monitoring services provide information about the status of grid resources. These services are used to describe grid resources. Lack of knowledge about resources will hamper resource scheduling, allocation and usage. Hence it is important that grid information service failures are reported and dealt with in a timely fashion.

Resource monitoring is a vital function in a distributed system, particularly when that system spans multiple locations as in that context no one is likely to have detailed knowledge of all components. Resource monitoring allows us to detect and diagnose the

many problems that can arise in such contexts, for example CPU load of a computing node is more than 80%. This task requires the ability to collect information from multiple, perhaps distributed, information sources.

2.4.3 Resource Brokerage

Resource Brokerage is defined as the process by which a node or machine in a distributed system becomes aware of the attributes or capabilities of the other nodes that are part of the system. In grid systems, the resource brokerage process is mainly used by a node to find out the “best” set of candidate resources that can execute a job or provide a specific service. The efficiency of the performance of brokerage process can be improved by observing that a grid is likely to consist of a variety of heterogeneous machines and networks.

Brokerage service provides pairing services between the service requester and the service provider. This pairing enables the selection of best available resources from the service provider for the execution of a specific task.

The pairing process in a brokerage service involves allocation and support functions such as:

- Allocating the appropriate resource or a combination of resources for the task execution.
- Supporting users’ deadline and budget constraints for scheduling optimizations.

2.4.4 Schedulers

A grid application that is organized as a collection of jobs is usually designed to have these jobs execute in parallel on different machines in the grid.

Schedulers are responsible for sending a job to a given machine to be executed. More advanced grid systems are including a job scheduler of some kind that automatically finds the most appropriate machine on which to run any given job that is waiting to be executed. Brokerage service is used in place of scheduler.

The user can query the grid system to see how his application and its sub jobs are progressing. If a job is successfully executed then the results are transferred to the specified location and report the job completion status to the user. A job execution may fail due the following reasons:

- **Programming error:** The job stops part way with some program fault.
- **Hardware or Power failure:** The machine or devices being used stop working in some way.
- **Communications interruption:** A communication path to the machine has failed or is overloaded with other data traffic.
- **Excessive slowness:** The job might be in an infinite loop or normal job progress may be limited by another process running at a higher priority or some other form of contention.

In case of job execution failures, schedulers, have to resubmit the jobs either on the same nodes or other nodes for execution.

2.5 Overview of some of Available Resource Management Frameworks

We studied four monitoring and information services for grid computing environments: the Globus Toolkit [IBM Red Book, 2003] Monitoring and Discovery Service2 [Czajkowski, K., 2001], the European Data Grid Monitoring Architecture (R-GMA) [RGMA] [Tierney, B., R. Avdt, 2002], Hawkeye [Hawkeye], part of Condor [Condor] project and Gridbus Broker [Buyya, VenuGopal, 2004]. The following subsections, briefly discusses about these frameworks.

2.5.1 Monitoring and Discovery Service (MDS)

The Monitoring and Discovery Service (MDS), [[Czajkowski, K., 2001], [MDS]] is the grid information service used in the Globus Toolkit [Foster, I, C. Kesselman, 1997]. MDS is built on top of the Lightweight Directory Access Protocol (LDAP) [OpenLdap].

MDS is used primarily to address the resource selection problem, namely, how a user identifies the host or set of hosts on which to run an application.

MDS provides access to static and dynamic information of resources. Basically, it contains the following components:

1. Grid Resource Information Service (GRIS)
2. Grid Index Information Service (GIIS)
3. Information Provider
4. MDS Client

The following figure 2.3 represents the conceptual view interconnection of the MDS components. The information provider obtains the resource information and it is passed to GRIS. GRIS registers its local information with the GIIS, which also registers with another GIIS, and so on. MDS clients can get the resource information directly from GRIS (for local resources) and/or a GIIS (for grid-wide resources). The MDS uses LDAP, which provides the decentralized maintenance of resource information.

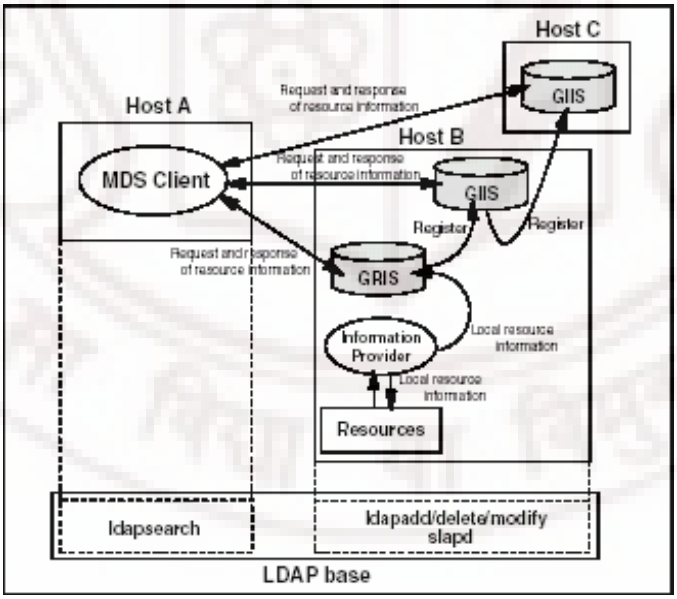

Figure 2.3: MDS Architecture [IBM Red Book, 2003]

GRIS is the repository of local resource information derived from information providers. GRIS is able to register its information with a GIIS, but GRIS itself does not receive registration requests. The local information maintained by GRIS is updated when requested, and cached for a period of time known as the time-to-live (TTL). If no request for the information is received by GRIS, the information will time out and be deleted. If a later request for the information is received, GRIS will call the relevant information provider(s) to retrieve the latest information.

GIIS is the repository that contains indexes of resource information registered by the GRIS and other GIIS. It can be seen as a grid wide information server. GIIS has a hierarchical mechanism, like DNS, and each GIIS has its own name. This means client users can specify the name of a GIIS node to search for information.

2.5.2 RGMA

The Relational Grid Monitoring Architecture (RGMA) [DataGrid, 2002] monitoring system is an implementation of the Grid Monitoring Architecture (GMA) [Tierney, B., R. Avdt, 2002] defined within the Global Grid Forum (GGF) [Global Grid Forum]. It is based on the relational data model [Fisher, S., 2001] and Java Servlet technologies [Java Servlet Technology]. Its main use is the notification of events i.e., a user can subscribe to a flow of data with specific properties directly from a data source. For example, a user can subscribe to a load-data data stream, and create a new Producer/Consumer pairing to allow notification when the load reaches some maximum or minimum.

The following figure 2.4 shows the GMA architecture. GMA is an architecture for monitoring components that specifically addresses the characteristics of grid platforms. GMA consists of three components: Consumers, Producers, and a Registry. Producers register themselves with the Registry, and Consumers query the Registry to find out what types of information are available and to locate the corresponding Producers. Then the Consumer can contact a specific Producer directly. GMA as defined currently does not specify the protocols or the underlying data model to be used.

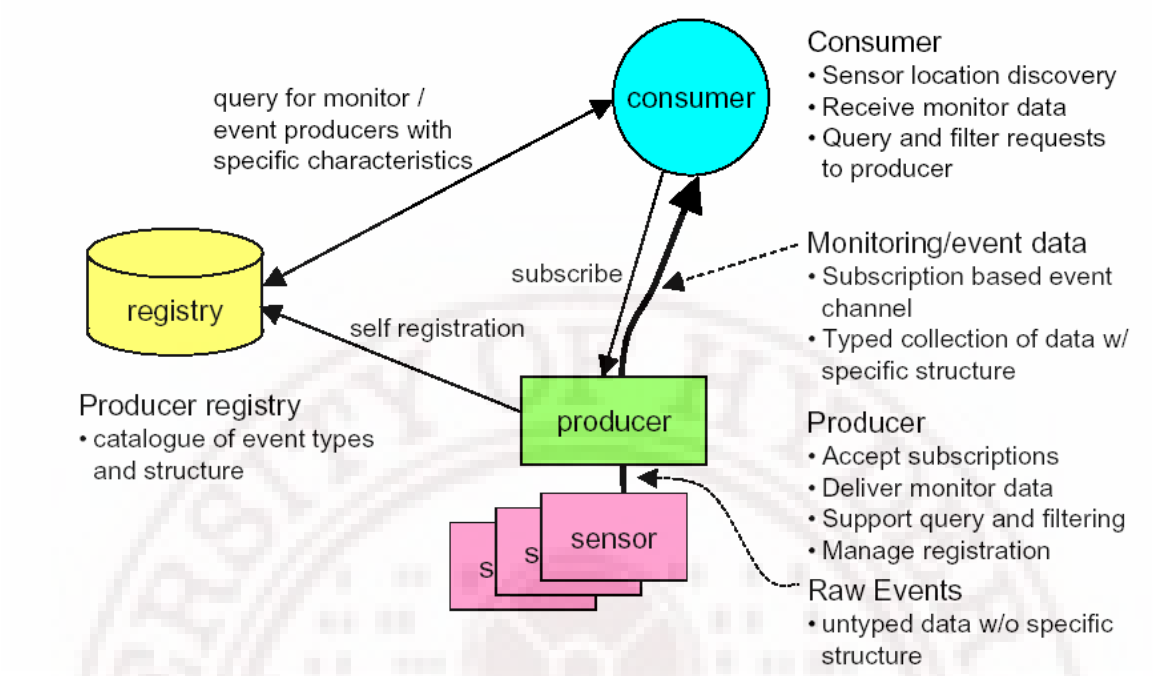

Figure 2.4: Grid Monitoring Architecture [William E. Johnston, 2002]

2.5.3 Hawkeye

Hawkeye [Hawkeye] is a tool developed by the Condor [Condor] group and designed to automate problem detection, for example to identify high CPU load, high network traffic, or resource failure within a distributed system. Its underlying infrastructure builds on the Condor [Condor] and ClassAd [Raman, R., 2001] technologies. The main use case that Hawkeye was built to address is that of being able to offer monitoring information to anyone interested and to execute actions in response to conditions. It also allows for easier software maintenance within a pool. Hawkeye involves two fundamental ideas: its use of the Condor *ClassAd* Language to identify resources in a pool, and *ClassAd Matchmaking* to execute jobs based on attribute values of resources to identify problems in a pool. A *ClassAd* is a set of attribute/value pairs (e.g., “operating system” and “Linux”). The Manager performs *ClassAd Matchmaking* between a *Trigger ClassAd*, submitted by a client, and all *Startd ClassAds.* A *Trigger ClassAd* specifies an event and a job to execute if the event occurs. For example, consider the case in which a Trigger ClassAd specifies an event in which the CPU load is greater than 50 and a job that will kill a Netscape client running on the

matched machine; if any machine advertises a Startd ClassAd with a CPU load value of greater than 50, the Manager will kill that machine's Netscape process. The following figure 2.5 shows the architecture of Hawkeye.

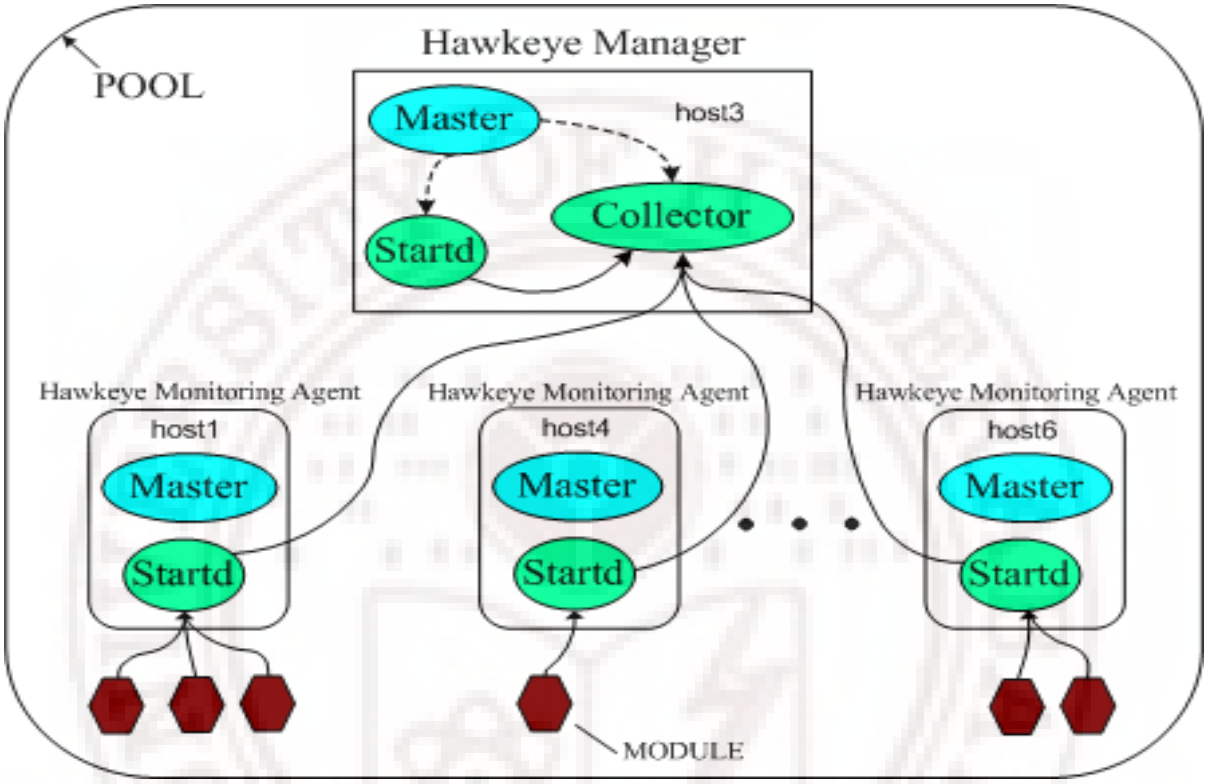

Figure 2.5: Hawkeye Architecture [Xuehai, Jeffrey, Jennifer, 2003]

The architecture of Hawkeye comprises four major components: Pool, Manager, Monitoring Agent, and Module. The components are organized in a four-level hierarchical structure. A pool is a set of computers, in which one computer serves as the Manager and the remaining computers serve as Monitoring Agents. A Manager is the head computer in the pool that collects and stores (in an indexed resident database) monitoring information from each agent registered to it. It is also the central target for queries about the status of any pool member. A Monitoring Agent is a distributed information service component that collects ClassAds from each of its Modules and then integrates them into a single *Startd ClassAd*. At fixed intervals, the agent sends the *Startd ClassAd* to its registered Manager. An agent can also directly answer queries about a particular Module; however, the client must first consult the Manager for the

agent's IP address. A Module is simply a sensor that advertises resource information in a *ClassAd* format.

2.5.4 Gridbus Broker

The Gridbus Broker [Srikumar, Rajkumar 2004] makes scheduling decisions on where to place the jobs on the grid depending on the computational resources characteristics (such as availability, capability, and cost), the user's Quality-of-Service requirements such as the deadline and budget, and the proximity of the required data or its replicas to the computational resources. Catalogues of replicated data describe the size of the file, the location of the file, the date it was produced, the number of events and other such attributes. Given a job and the input file(s) it requires, the broker looks up the Replica Catalogue at the local site to locate the sites where the required input file is and its size. Then it takes into account various other factors such as the cost, the computing power available at the site, the network bandwidth, the resource reputation, and the account information to make a decision on where to dispatch the job. The broker identifies resource service prices by querying the Grid Market Directory (GMD).

If an application needs to access remote databases, Gridbus provides transparent data access mechanisms and a catalogue that supports logical mapping of data files to a distributed storage devices. The broker performs discovery and online extraction of data-sets from the closest data sources and then farms out analysis jobs to optimal resources. The brokers evaluates whether to process jobs on a resource where the data is available by moving the application code or move data to a resource where the application is available, or move both of them to a suitable computing resource.

The architecture of the Gridbus broker is shown in figure 2.6. The inputs to the broker are the tasks and the associated parameters with their values. These can be specified within a "plan" file that specifies the tasks and the types of the parameters and their values for these tasks.

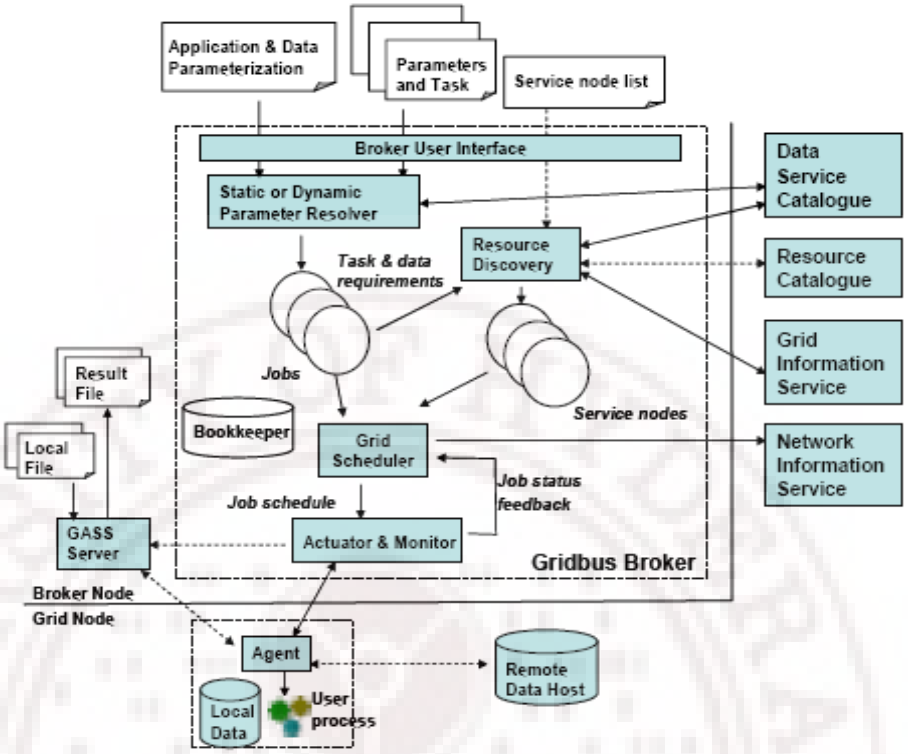

Figure 2.6: Grid bus Broker Architecture [Srikumar, Rajkumar 2004]

A task is a sequence of commands that describe the user's requirements. For example, the user may specify an application to be executed at the remote site, by providing an input file to be copied over before execution and the results to be returned back after required execution. A task encapsulates this information within its description. A task is accompanied by parameters which can either be static or dynamic. A static parameter is a variable whose domain is well-defined either as a range of values, as a single static value or as one among a set of values. A dynamic parameter has either an undefined or an unbounded domain whose definition or boundary conditions respectively, are required to be established at runtime. As an example, in the current implementation, a parameter type has been defined which describes a set of files over which the application has to be executed. This set can be described as a wildcard search within a physical or a logical directory, to be resolved at runtime, thus creating a dynamic parameter.

The task requirements drive the discovery of resources such as computational nodes and data resources. The resource discovery module gathers information from

remote information services such as the Grid Index Information Service (GIIS) [Czajkowski, K., 2001] for availability of compute resources. Optionally, the list of available compute resources can be provided by the user to the broker. The broker also interacts with the information service on each computational node to obtain its properties. Data files can be organized as Logical File Names (LFNs) within a virtual directory structure using a Replica/Data Service Catalog. Each LFN maps to one or many Physical File Names (PFNs) somewhere on the grid, usually specified via URLs. The broker will resolve the LFNs to the appropriate physical file location(s) by querying the catalog.

The task description, i.e. the task along with its associated parameters, is resolved or “decomposed” into jobs. A job is an instantiation of the task with a unique combination of parameter values. It is also the unit of work that is sent to a grid node. The set of jobs along with the set of service nodes are an input to the scheduler. For jobs requiring remote data, the scheduler interacts with a network monitoring service to obtain the information about current available bandwidth between the data sources and the compute resources. In the current implementation, the Network Weather Service (NWS) [Wolski, R] has been used to obtain this information.

The jobs are dispatched to the remote node through the Actuator component. The Actuator submits the job to the remote node using the functionality provided by the middleware running on it. The Actuator has been designed to operate with different grid middleware frameworks and toolkits such as Globus 2.4 [Foster, I, C. Kesselman, 1997]. The task commands are encapsulated within an agent who is dispatched to and executed on the remote machine. If a data file has been associated with the job and a suitable data host identified for that file, then the Agent obtains the file through a remote data transfer from the data host. Additionally, it may require some configuration or input parameter files that it obtains from the broker through a mechanism such as a Globus Access to Secondary Storage (GASS) server [Bester J, Foster I, 1999]. These files are assumed to be small and in tens or hundreds of kilobytes which impact the overall execution time of a job negligibly whereas the data files are in the range of megabytes or larger. On the completion of execution, the agent returns any results to the broker and provides debugging information. The monitoring component keeps track of job status –

whether the jobs are queued, executing, finished successfully or failed. The bookkeeper keeps a persistent record of job and resource states throughout the entire execution.

2.6 Comparison of GMIS with MDS, RGMA, Hawkeye and Gridbus Broker

In the study of resource management services discussed above, we found that each approach has different behaviors, often due to their different design goals. To take advantage of these approaches, we included all features of these services in our Grid Management Information Server (GMIS) framework.

The following table 2.2 shows the comparison among MDS2, R-GMA, Hawkeye, GridBus Broker and GMIS framework. Following parameters are used to compare these along with the questions and design considerations mentioned in Chapter 1, Section 1.6.

- **Primary objective:** The capabilities provided or problems solved by the frameworks.
- **Decentralized:** Support for distributed deployment of framework and providing a single user interface for all the activities.
- **Dynamic Grid Network Discovery:** Capability to discover and utilize all the available grids, clusters, nodes and applications automatically and maintain hierarchical information.
- **Dynamic Event based Discovery:** Capability to discover and utilize a newly available resource such as a node or application based on an event sent by the resource to the framework.
- **User based Discovery:** Capability to discovery and utilize grid resource based on the input provided by the user.
- **Push Model:** Information about the various resources and jobs is updated based on the events sent by the resources.
- **Pull Model:** Information about the various resources and jobs is updated based on the periodic polling the resources and parsing the responses.
- **Data Server:** Centralized repository of all the resource and jobs information.
- **Directory Server:** Service provider that maintains the hierarchical information of the resources and provide information as needed by other services or middleware.

- **Multi selection capability:** Based on the job requirements, the user will be provided with a list of available resources and user has a choice to select a particular resource for job execution.
- **Data subscription support:** Any application or middleware can subscribe for existing resource or job information and also any changes to them.
- **Resource Mining:** Capacity to display resource information based on historical data about job execution and resource capabilities.
- **MDO Support:** Support for Multidisciplinary Design Optimization, which is used in design of complex engineering systems by performing system level analysis.

Parameter	MDS2	RGMA	Hawkeye	Gridbus Broker	GMIS
Primary objective	Brokerage	Notification of events & Monitoring	Automate problem detection & Monitoring	Brokerage & Scheduling	Discovery, Monitoring & Brokerage
Decentralized	Yes	No	No	Yes	Yes
Dynamic Grid Network Discovery	No	Yes	No	Yes	Yes
Dynamic Event based Discovery	No	Yes	No	No	Yes
User based discovery	Yes	No	No	No	Yes
Push model	No	Yes	Yes	Yes	Yes
Pull model	Yes	Yes	No	No	Yes
Data Server	Yes	Yes	Yes	Yes	Yes
Directory Server	Yes	Yes	Yes	Yes	Yes
Multi selection capability	No	No	No	Yes	Yes
Data subscription support	No	Yes	No	Yes	Yes
Resource mining	No	No	No	No	Yes
MDO support	No	No	No	No	Yes
How to know where the resources are?	Information provider	Notification of events	Collector	Grid Information Service	Dynamic network discovery, Dynamic Event based discovery and user based discovery.
How to identify resources?	GIIS and GRIS	Registry	Class Ad Objects	Grid Information Service	Database and Topology manager MIT

How to get permissions to use them?	Information providers	Producers	Manager and Agent	Grid Resource Broker	Agent Software
How to know what are the applications available in the resources?	Not addressed	Not addressed	Not addressed	Data Catalogue	Agent software
How to use the resources?	GRIS and GIIS	Consumers	Manager	Grid Resource Broker	Resource Selector
How to submit remote jobs?	Middle ware	Middleware	Middleware	Grid Resource Broker	Using user interface provided in Viewer and as well as Middleware.
How to get access to resources to all the machines simultaneously?	GRIS and GIIS	Producers	Agents	Grid Resource Broker	Resource Selector
What happens if a resource fails?	Not addressed	Not addressed	Not addressed	Addressed	Addressed
How input/output files are managed?	GridFTP	Grid Sandbox	Database tables	GridFTP	FTP
Seamless and transparent user access to resource information	No	No	No	No	Yes
Easy to use web-based interface	No	No	No	No	Yes
Collation of information from multiple resources	No	No	No	Yes	Yes
Efficient resource management	No	No	No	No	Yes
Scalable and Adaptable framework	No	No	No	No	Yes
Poll and Event based information gathering mechanisms	No	Partially (event based)	No	No	Yes

Table 2.2 : Comparison of MDS2, R-GMA, Hawkeye, Gridbus Broker and GMIS

2.7 Summary

In this chapter we presented an overview of grid computing, ancestors of the Grid and OGSA architecture of grid computing. It discussed about basic grid technologies like Globus Toolkit, IBM Grid Tool box, SUN Grid Engine, Avaki, and Microsoft .NET technologies. Comparison among these technologies is presented. It extensively discussed about grid activities of Global Grid Forum, Condor, European DataGrid, PRAGMA and GridBus.

It discussed about the resource management and various resource specific characterization capabilities. It presented a detail literature survey on some of the popular resource management frameworks MDS2, R-GMA, Hawkeye and GridBus Broker. Finally, comparison of these frameworks with proposed GMIS framework is presented. Several features not available in the other popular resource management frameworks are also been discussed. Next chapter will discuss about the GMIS Framework architecture.

CHAPTER 3

GMIS Framework Architecture for Resource Management

This chapter presents the Grid Management Information Server (GMIS) architecture for resource management in grid environment. GMIS framework addresses the resource management requirements like resource discovery, monitoring and brokerage of resources in grid environment. It also discusses about distributed GMIS architecture.

3.1 GMIS Architecture Overview

The GMIS framework has been developed to address growing complexity and manageability in grids. Very few frameworks have been attempted to solve the grid management issues in real time. The architecture of GMIS framework described in this thesis is an attempt to provide solutions to the issues in resource management; discovery, monitoring and brokerage as mentioned in chapter 1, section 1.6.

The GMIS framework provides services for discovery, monitoring and brokerage of resources in a grid network. Discovering resources on a grid is an automated process which keeps track of addition and deletion of resources from the grid. To gather the information about resources, communicator server and communicator agents are used. Resources details are stored in a relational database on GMIS.

Monitoring service is used to monitor resources on grid level, cluster level and node level. Using this service, grid resources are visually represented as icons on a viewer with resource status. GMIS framework allows viewing of summary information of the individual node and summary view of cluster which is comparable and analyzable.

Brokerage service allocates the best possible resource or a combination of resources for a job execution based on set of attributes given by grid users.

GMIS architecture has been modeled as nine distinct components with specific functionalities and interfaces. The following figure 3.1 shows the architecture of GMIS.

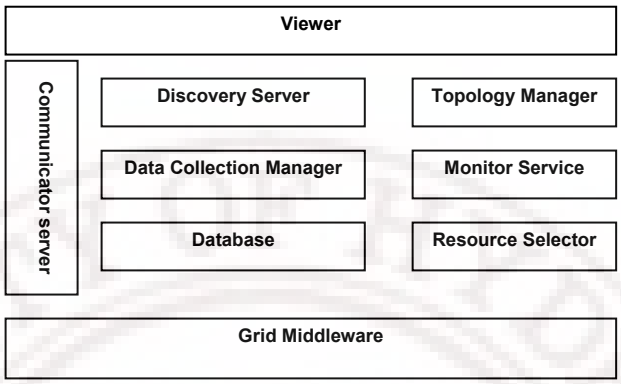

Figure 3.1: GMIS Architecture

(1) Discovery Server: Discovery Server discovers resources on a grid. Resource discovery is at three levels: (i) Node level (ii) Cluster level and (iii) Grid level. In node level, an Internet protocol address or Fully Qualified Domain Name (FQDN) is used to discovery resource information. In cluster level and grid level, a range of Internet protocol addresses are used to discover resources. User provides the information on Internet protocol addresses or FQDN, using user interface on Viewer. Discovery Server sends a request to resource agent software using communicator server. Upon receiving the request, resource discovery agents on resources gather resource information and sends to Discovery Server. The resource information is stored in the relational database through Topology Manager.

(2) Data Collection Manager: Data Collection Manager provides the procedures to communicate with the agent software for information about the resources. It receives notifications from agents, and initiates actions accordingly. It polls resources for current status of resources and parses the poll responses. It updates the Topology Manager with the current resource information. It also handles events like addition of resource, deletion of resource and modification of resource.

(3) **Database:** Database contains information about the resources on grid. The data will get updated by Topology Manager and retrieved by other services like Monitor Service and Resource Selector.

(4) **Topology Manager:** Topology Manager maintains all resources information in a tree structure called Management Information Tree (MIT) as managed objects. Managed objects are created when there is a notification from the Discovery Server. Managed objects are modified or deleted based on the messages from Data Collection Manager. It also updates Database when ever there is a change in resource information.

(5) **Monitor Service:** Monitor Service provides information to the viewer by retrieving the resource information from Database. It controls and maintains the various users' access levels. The access information is persistently stored in the database. Each grid can have different or same access levels for the same user. Viewer provides interface for 'user access' configuration.

(6) **Resource Selector:** Resource Selector is responsible for generating a set of best possible resources for the given set of resource attributes. The entire resources information is in database and Topology Manager updates the information. Resource Selector takes inputs from user and searches in the database. If appropriate resources data is available in the database, those details will be provided to the user. Users can use these resources for their job execution. A user interface is provided on viewer to submit jobs. Key features of Resource Selector are:

   i. Providing Brokerage Service
   ii. Generate Activation Graph
   iii. Job execution

(7) **Communicator Server:** Communicator server establishes and maintains communication between the GMIS and resources agent software. Agent software on resources is designed to provide resource information and to communicate with GMIS on behalf of resource.

(8) **Grid Middleware:** Grid Middleware like schedulers, resource allocation managers interact with the GMIS services using Application Programming Interfaces (APIs). For instance, when a scheduler needs a resource for a particular application, it interacts with the Topology Manager through the resource selector, and a suitable resource is provided.

(9) Viewer: Viewer interacts with the GMIS services through monitor service e.g., status information of resources. Viewer is a web based application uses Tomcat Apache as a Web Server. Icons are used to represent grid, cluster and node. Viewer gets resource information from database using Java Servlets. Viewer contains user interfaces for user management, discovery of resources, monitoring services, resource selection and job submission.

3.2 Discovery Server

Discovery server discovers all resources dynamically in a grid network using communicator agents. Discovery Server is an automated process to discover resources on a grid. Resource details are stored in relational database on GMIS through Topology Manager. The following figure 3.2 shows how the resources are discovered and stored in GMIS database and provides information to services like monitor service.

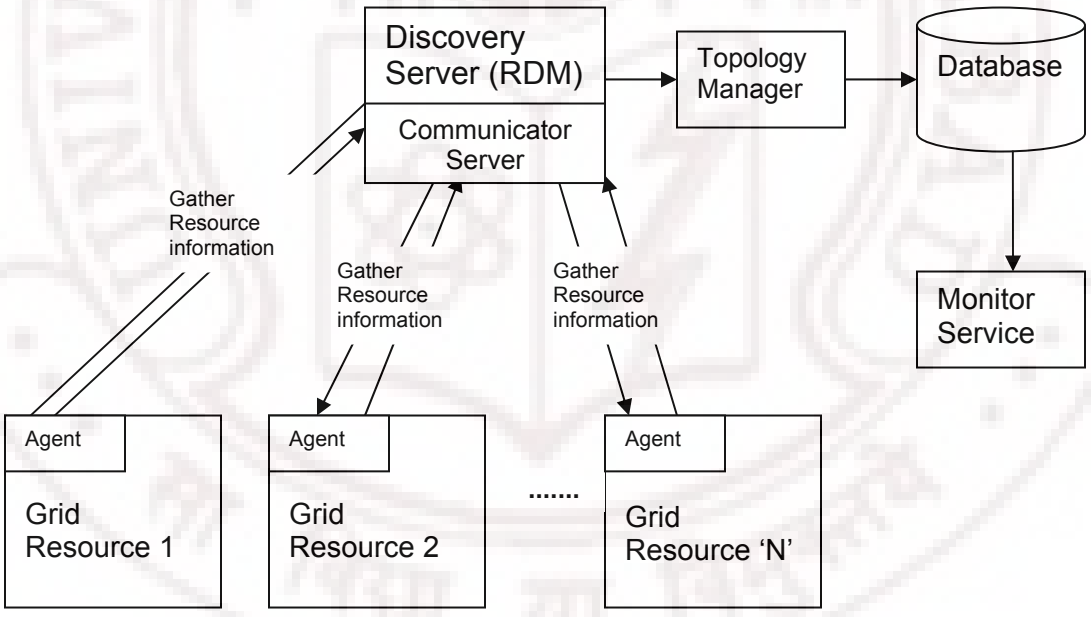

Figure 3.2: Discovery Server

Two types of discoveries are supported in GMIS system. One is hardware resources discovery and other one is services discovery. Hardware resource discovery is at three levels: (1) Node level (2) Cluster level and (3) Grid level. At node level, an Internet protocol address or Fully Qualified Domain Name (FQDN) is used to discover

resource information. At cluster level and grid level, a range of Internet protocol addresses are used to discover resources. A user interface is provided on Viewer to give Internet protocol addresses or FQDN. Discovery server process called Resource Discovery Manager (RDM) runs at the server which is part of GMIS. RDM requests communicator agents for the hardware resource information through communicator server. Resource discovery communicator agents running on resources will gather the resource information and sends it to RDM. The resource information is stored in the relational database through Topology Manager.

Similarly, hardware resource discovery methodology has been used for services discovery also. How ever, there is one limitation i.e., services should be in a particular directory, which is configurable, or services name should start with “UH_”.

Computing resources have two types of attributes: (i) static attributes such as type of operating system installed, network bandwidth, processor speed and storage capacity (including physical and secondary memory) and (ii) dynamic attributes such as processor utilization, physical memory utilization, free secondary memory size, and network bandwidth utilization; and applications have many types of attributes.

Static attributes information will be gathered only one time by Discovery Server, since this data will not change with time. At the time of initialization dynamic attributes information will be gathered by Discovery Server. As dynamic attributes information changes with time periodic information gathering is required. Data Collection Manager will collect the resource dynamic attributes information periodically.

3.3 Data Collection Manager

Data Collection Manager (DCM) sends request to communicator agents running at resources to collect resource information periodically. Communicator agents respond to the DCM request with the latest resource information. Data Collection Manager updates this information in topology and database through Topology Manager. A communicator agent, Critical Information Agent (CIA) sends notification events to DCM when there is a change in critical resource information. For example, if CPU usage exceeds 80% then CIA sends a critical event message to DCM. If a node is

added/modified/deleted then also an event message will come from CIA to DCM. The main functionalities of DCM are:

- Poll resources
- Parsing resource responses
- Event handling

3.3.1 DCM Polling and Parsing Mechanism

DCM periodically polls resources for current status of resource dynamic attributes. Communicator agents running at resources respond to DCM poll request. DCM parses the responses coming from communicator agents and updates Database through Topology Manager (TM). The following figure 3.3 shows the polling-parsing approach for CPU load which is a dynamic attribute of a computing resource.

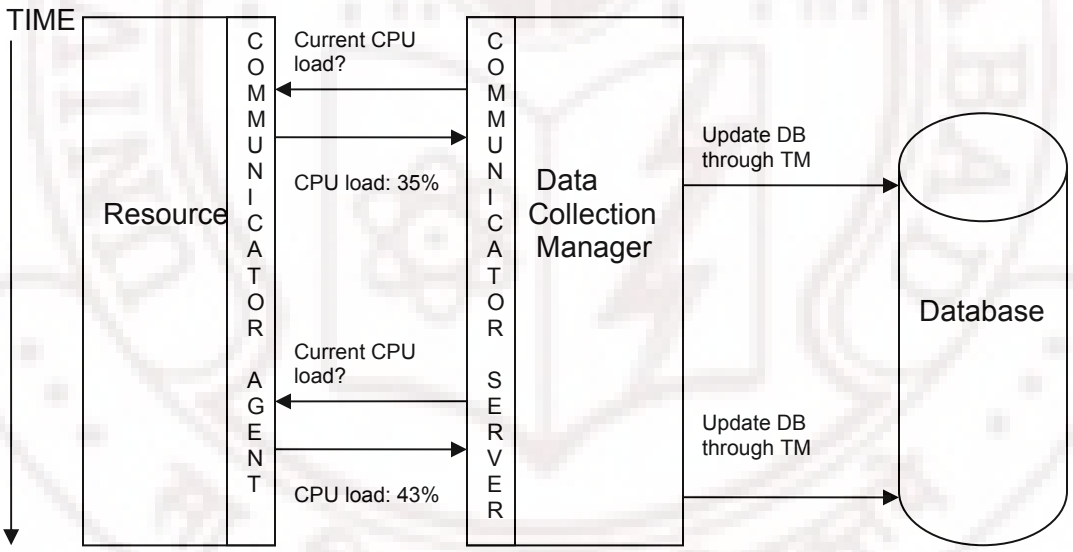

Figure 3.3: Polling and Parsing Mechanism in GMIS

1. The DCM sends a request to the communicator agent for current CPU load using Communicator Server.
2. The communicator agent finds the current CPU load, responds with CPU load value.
3. DCM parses the response that comes from communicator agent and updates Database through Topology Manager.

4. DCM waits for a while before making another call according to the polling interval.
5. Steps 1-4 repeats for the next time interval.

3.3.2 DCM Event Handling

If any critical change like addition of resource, deletion of resource or modification of resource, occurs then a resource communicator agent called Critical Information Agent (CIA) sends a notification event to DCM through Communicator Server. The following figure 3.4 shows a notification event handling.

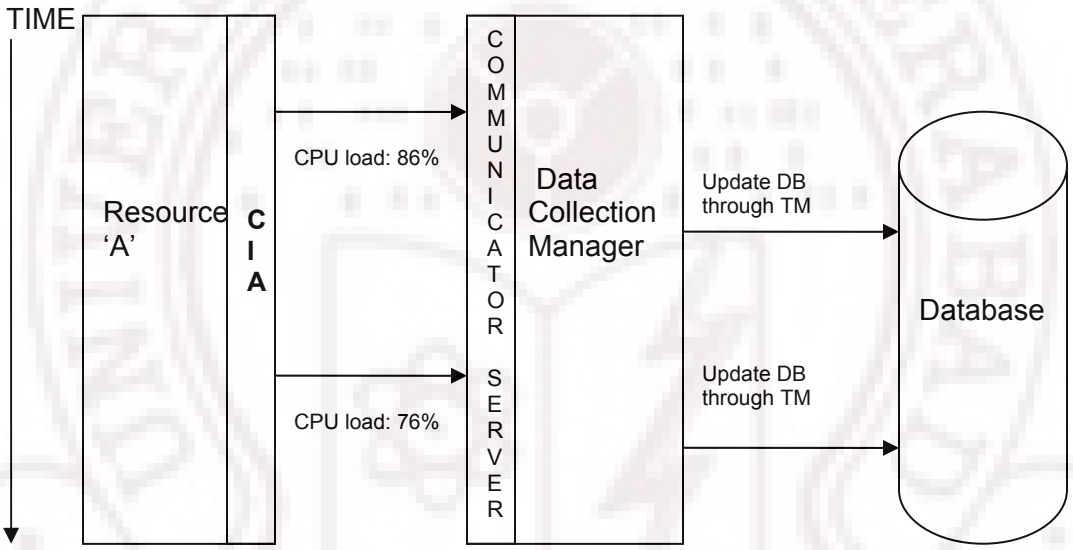

Figure 3.4: Event Handling in GMIS

1. If a resource 'A', CPU load exceeds 80%, then Critical Information Agent (CIA) which runs on resource sends a notification event to DCM through communicator server.
2. DCM updates the Database, through Topology Manager.
3. Once the resource 'A', CPU load drops below 80%, then again a notification event comes to DCM.
4. DCM updates the Database, through Topology Manager.

3.4 Database

GMIS framework database is a relational database maintains resource information. It maintains resource information tree-like hierarchical representation of the relationships between the node, cluster and grid. Here nodes are physical computing resources. Cluster is a logical container which contains a group of nodes. Similarly Grid is a logical container which contains a group of clusters. For example, in the following figure 3.5 node ‘n’ belongs to cluster ‘c’, so in node ’n’ resource information parent of node ‘n’ is cluster ‘c’. Similarly cluster ‘c’ belongs to grid ‘g’, so in cluster ‘c’ resource information parent of cluster ‘c’ is grid ‘g’.

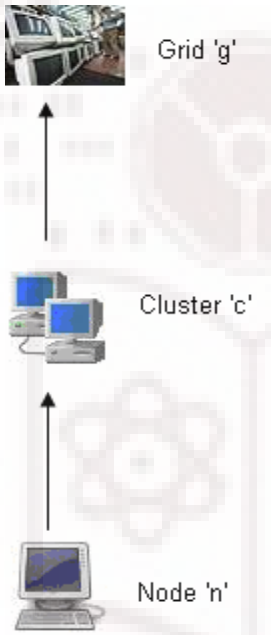

Figure 3.5: Relationship among Node – Cluster – Grid

3.4.1 Data Handler

Data Handler is a single interface for all Database data handling. This acts as a wrapper for all SQL queries and updates to database. Some of the handlers are:

- Handling the parameters associated with each resource:
  - Creating the instance of resource.
  - Retrieving the instance of resource.

- Modifying the parameters associated with each resource.
- Retrieving the parameters associated with each resource.

- Handling the hierarchy of the resources:
  - Traversing through the hierarchy
    - Retrieving the parent for a given resource.
    - Retrieving the child of a given resource.

3.5 Topology Manager

Topology Manager is backbone of the GMIS framework. It provides mechanism to create and maintain resource object hierarchy for the resources in grid network. Each resource is represented as a managed object. Topology manager stores, retrieves and manipulates data of managed objects based on updates from Discovery Server and Data Collection Manager.

Every resource on grid is considered as a Managed Object [RFC 1155] in Topology Manager. These objects are represented by one or more objects in the GMIS database. Communicator agent running at the resources is relied on to get the status of the resources.

Although a managed object need not be a physical object that can be seen, touched, and felt, it is convenient to use a physical representation to understand the characteristics and operations associated with a managed object.

Following figure 3.6 shows the topology manager associated with grid network configuration through Communicator Server, Discovery Server and Data Collection Manager. The topology manager has a database (DB) which contains the measured or administratively configured value of the resources of a grid network. This topology manager database is nothing but the GMIS database.

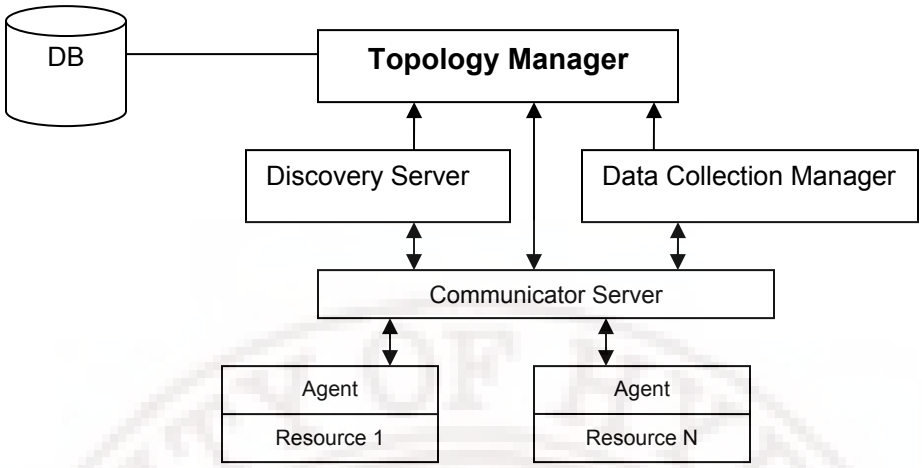

Figure 3.6: Topology Manager

3.5.1 Topology Modeling

Topology is a central repository for all the resource managed objects in the grid network. During initialization topology is created based on inputs from discovery server or retrieving resource data from Database. Topology gets updated based on data collection manager or discovery server updates. Topology modeling provides a model to access, create, modify and manipulate the data in the topology. In topology modeling Management Information Base (MIB) is used to represent all managed objects.

An object identifier (or object ID) uniquely identifies a managed object in the MIB hierarchy. The MIB hierarchy can be depicted as a tree with a nameless root, the levels of which are assigned by different organizations.

Managed objects are uniquely defines by a tree structure specified by the Open Systems Interconnection (OSI) model and are used in the Internet model. The following figure 3.7 shows the generic representation of the Management Information Tree (MIT). There is a root node and well-defined nodes underneath each node at different levels. Each managed object occupies a node in the tree. In the OSI model, the managed objects are defined by a containment tree that represents the MIT.

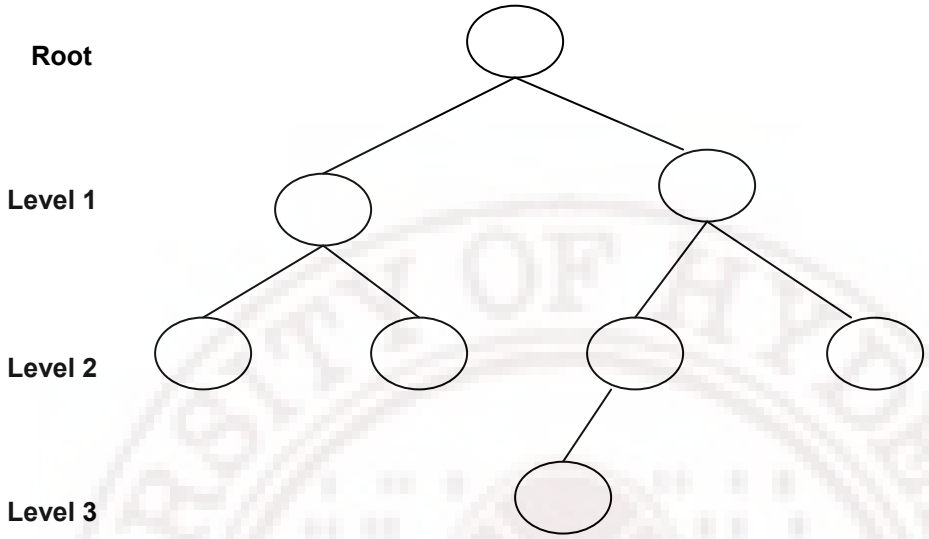

Figure 3.7: Generic Representation of Management Information Tree

The following figure 3.8 shows the internationally adopted OSI MIB tree. The root node does not have an explicit designation. There are three nodes in the layer beneath the root: iso, ccitt (itu), and iso-ccitt, (iso-itu). The iso defines the International Standards Organization and itu defines the International Telecommunications Union (the old name is ccitt). The two standards organizations are on the first layer and define management of objects under them. The joint iso-itu node is for management objects jointly defined by the two organizations. The number in each circle identifies the designation of the object in each layer. Thus, iso is designated as 1 and org as 1.3, dod (Department of Defense) as 1.3.6 and the internet as 1.3.6.1. All Internet managed objects will be that number followed by more dots and numbers.

In this MIB tree, the top-level MIB object IDs belong to different standards organizations, while lower-level object IDs are allocated by associated organizations. Vendors can define private branches that include managed objects for their own products. MIBs that have not been standardized typically are positioned in the experimental branch.

The managed object *at Input* can be uniquely identified either by the object name—*iso.identified-organization.dod.internet.private.enterprise.cisco.temporary variables.AppleTalk.atInput*—or by the equivalent object descriptor: 1.3.6.1.4.1.9.3.3.1.

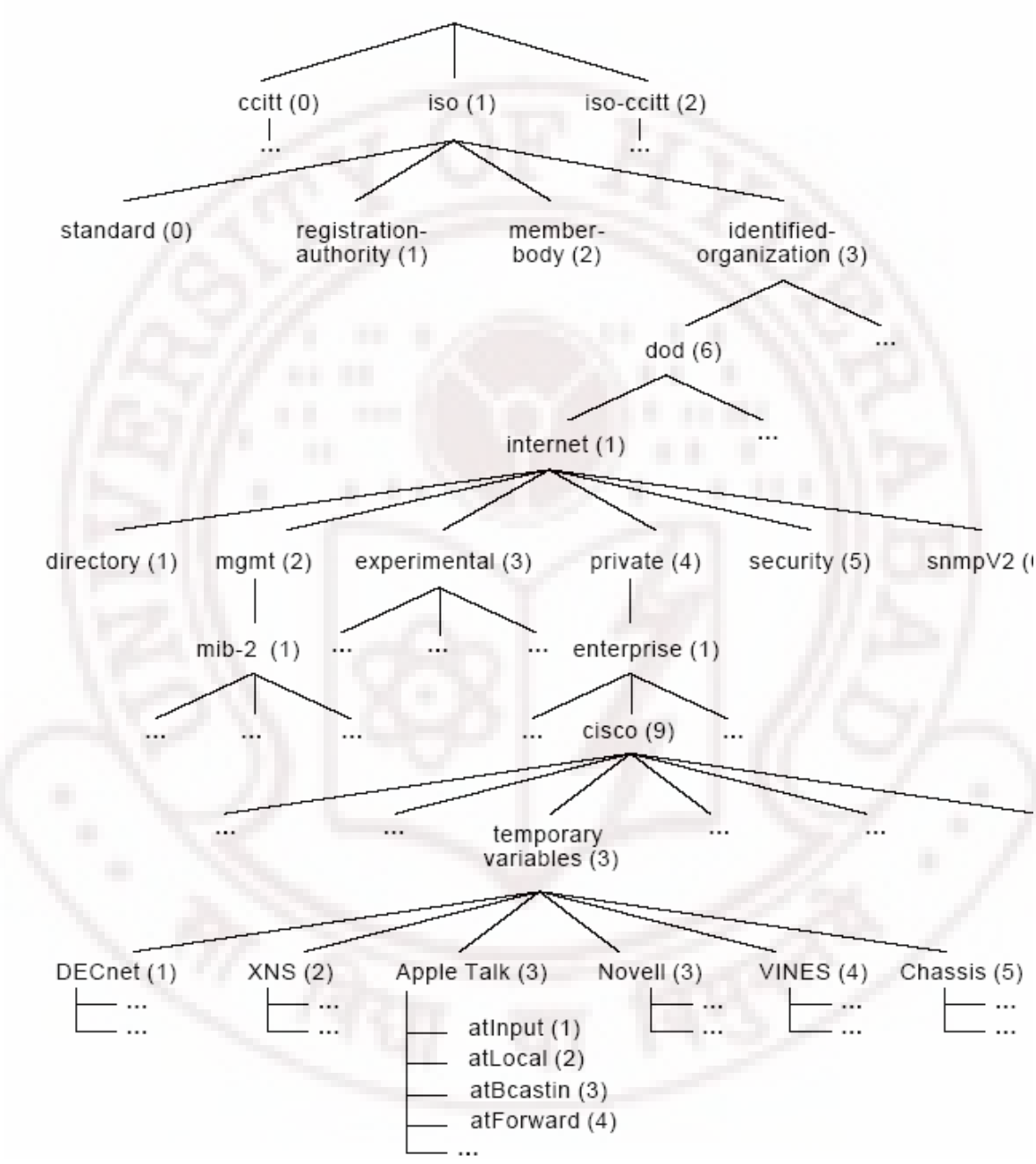

Figure 3.8: MIB Tree illustrates various hierarchies assigned by different organizations [Internetworking, 2001]

Topology model deals with the structure and the storage of information. The representation of objects (resources) and information relevant to their management form the topology model. As discussed in earlier sections, resource information is passed between the resource communicator agent and topology manager through GMIS components like Communicator Server, Discovery Server and Data Collection Manager. The topology model specifies the information base to describe managed objects and their relationships. The Structure of Management Information (SMI) defines the syntax and semantics of management information stored in the MIB. The MIB is used by both communicator agent and topology manager to store and exchange management information. The MIB associated with a communicator agent is called the agent MIB and the MIB associated with a topology manger is designated the manager MIB. A manager MIB consists of information on all the grid network resources that it manages, whereas an agent MIB needs to know only its local information.

Following figure 3.9 expands the topology manager configuration shown in figure 3.6, to include the MIB associated with the topology manager. Thus, the topology manager has both the database (DB) and the MIB. It is important to distinguish between the DB and MIB. The DB is a real database and contains the measured or administratively configured value of the resources of the grid network. On the other hand, the MIB is a virtual database and contains the information necessary for processes to exchange information.

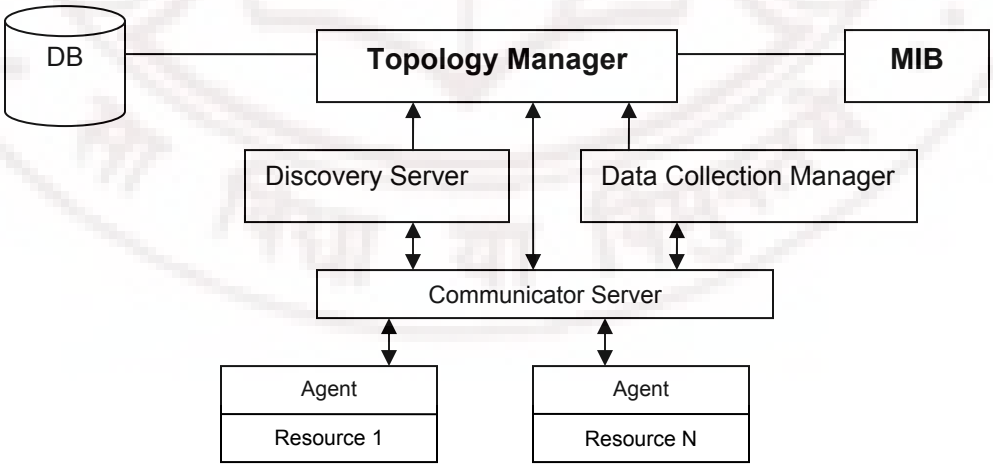

Figure 3.9: Topology Manager with MIB

Let us illustrate the distinction between MIB and DB by considering the scenario of adding a new computing resource to the network. Assume all the computing nodes in the network are made by a single vendor, say SUN. In the above figure 3.9 the Topology manager's knowledge about SUN computing nodes and their associated parameters are in its DB. For example, number of CPUs in the computing nodes is a parameter associated with the computing node (MIB information) and if they are dual-CPU nodes, the value associated with the number of CPUs is 2 (DB information). Suppose we added another SUN computing node to the grid network. The Topology manager would recognize the addition of resource to the grid network through Data Collection Manager. The new computing node is another instance of the computing node with a new IP address, and its MIB information is already in the manager's MIB. The node IP address and the number of CPUs associated with it are added to the DB by the topology manger.

Now, let us add an IBM computing node to the network. Let this be the first time that an IBM computing node is added to the network. The topology manager would recognize the addition of a new resource to the network through data collection manager. However, it would not know what component has been added until the MIB information on the computing node is added to the manager's MIB. This information is actually compiled into the manager's MIB schema. After the information on the IBM computing node has been added to its MIB, then only data collection manager can communicate with the communicator agent residing in the IBM computing node. DCM then retrieves the values for the type of computing node, the number of CPUs, and so on, and adds them to its DB.

The MIB that contains data on managed objects need not be limited to physical elements. For example, in grid network management, management information extends beyond that associated with the description of network elements or objects. Here are some examples of information that can be stored in the MIB:

- **Grid Network Elements:** hubs, bridges, routers, transmission facilities
- **Software Processes:** programs, algorithms, protocol functions, database
- **Administrative Information:** contact person, account number

Host Resources MIB [RFC2790, 2000] has been used in the GMIS framework. The Host Resources MIB defines a uniform set of objects useful for the management of host computers. Host computers are independent of the operating system, network services, or any software application. The Host Resources MIB defines objects which are common across many computer system architectures.

3.6 Monitor Service

Monitor service provides resource data from the database to viewer as shown in the following figure 3.10. Monitor service has two functionalities, one is access control and other is status monitor.

Monitor Service controls and maintains the various users' access levels. The access information is persistently stored in the database. Each grid can have different or same access levels for the same user. Viewer provides a user interface for 'user access' configuration. At the time of opening viewer a user interface obtains user's 'login' and passes it to the Monitor Service for verification. If verification succeeds then only user is able to view the grid network information.

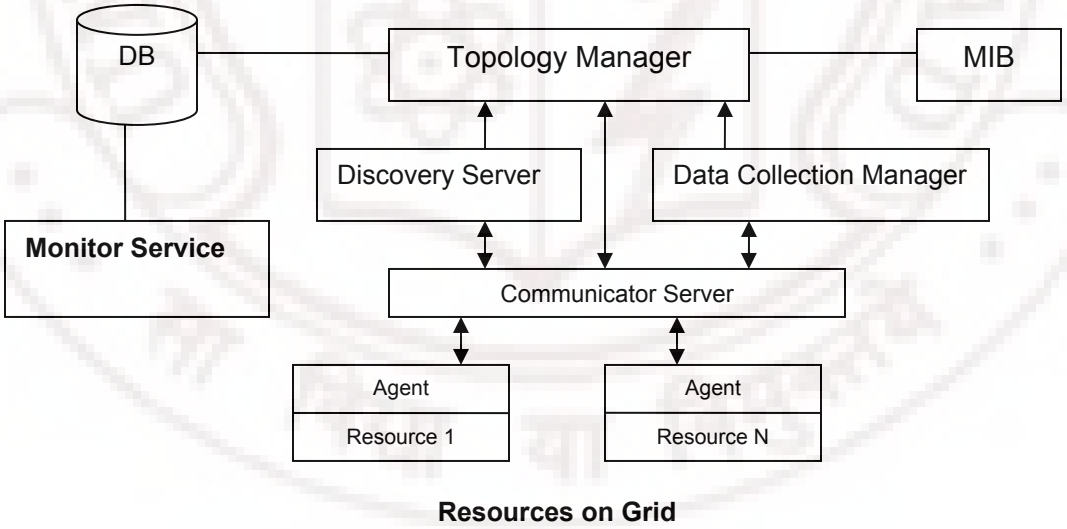

Figure 3.10: Monitor Service in GMIS

Monitor Service handles the viewer interface for resource status display. This keeps track of all the users that are registered with the GMIS and takes the responsibility of updating all the clients whenever there is change of resource state on the GMIS.

3.7 Resource Selector

Grid resource brokering is defined as the process of making scheduling decisions involving resources over multiple administrative domains [Schopf, 2002]. This includes searching multiple administrative domains to use a single machine or scheduling a single job to use multiple resources at a single site or multiple sites. A Grid broker must make resource selection decisions in an environment where it has no control over the local resources. The resources are distributed, and information about the resources is often limited.

The resource broker performs a number of basic functions. The first step is the identification and selection of resources that best fit the needs of the Grid application. The broker will then submit jobs in the application to the chosen machines. The broker thus handles submission of jobs but not how the job is actually executed on the resource, as that is part of the management system that resides on the resource involved. The resource broker functionality is done by the Resource Selector component of GMIS. Once jobs are being executed the broker for that application monitors the resources and the progression of the jobs through the Data Collection Manager. The following figure 3.11 shows Resource Selector component in GMIS.

Key features of Resource Selector are:

(1) **Resource Selection:** Based on the user inputs, retrieve appropriate resource information from GMIS database and provide to the user. A user interface is developed on viewer to provide user inputs.

(2) **Generate Activation Graph:** Activation graph is specification of the order of services to be executed. Activation graph will be created or modified using the user interface provided on viewer.

(3) **Job execution:** Grid users submit jobs using the interface provided on viewer. Resource Selector executes these jobs on behalf of the users.

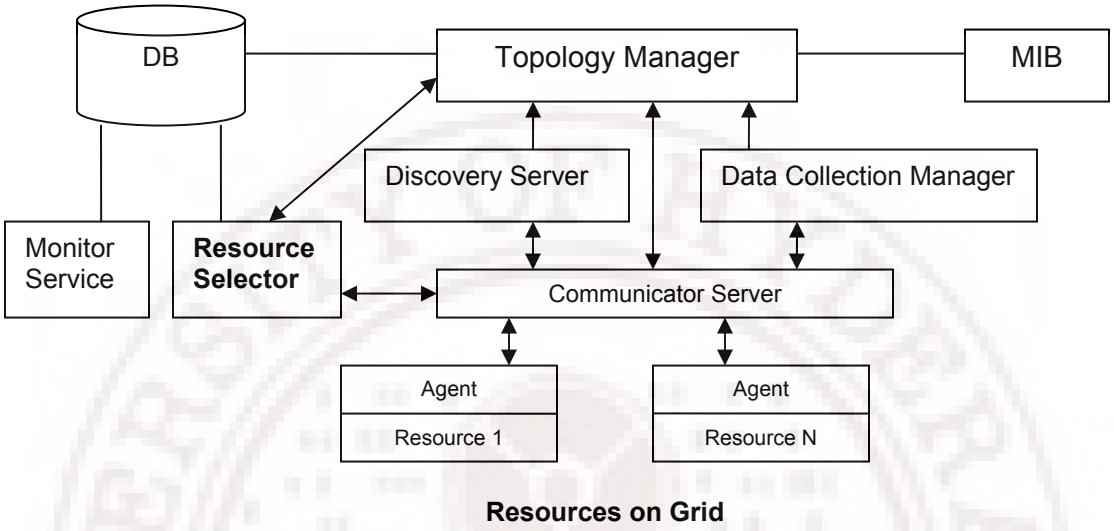

Figure 3.11: Resource Selector Service in GMIS

3.8 Communicator Server

A key component of the framework is Communicator Server (CS). The CS establishes and maintains communication between the GMIS and resources. At GMIS, Communicator Server runs along with GMIS, and communicator agents run on resources. Resources on grid network are classified as managed and unmanaged objects or elements. The managed elements have a management process running in them, called Communicator Agents. The unmanaged elements do not have a Communicator agent. The Communicator Server communicates with the agent in the managed element. The following figure 3.12 illustrates this.

For example, if an object that is managed by GMIS Communicator Server is selected on the viewer to submit a job. Communicator Server invokes a Communicator Agent on the corresponding object resource to execute the job.

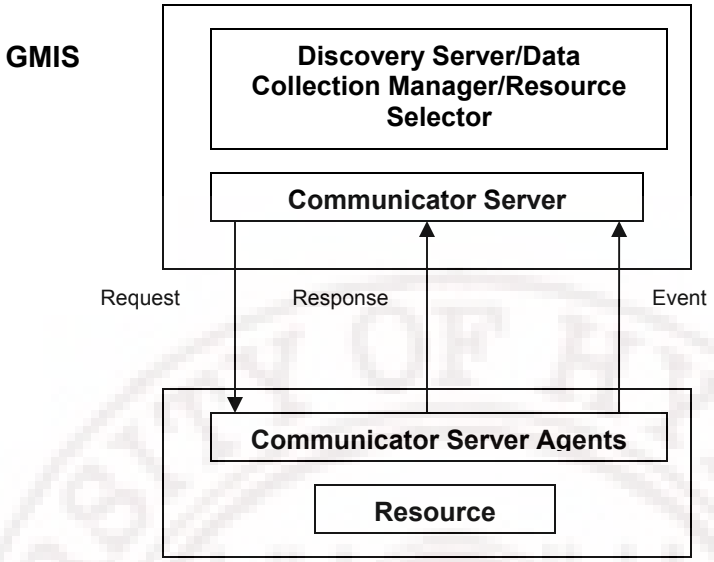

Figure 3.12: GMIS Communicator Server

The Communicator Server messages are exchanged using the connectionless UDP transport protocol in order to be consistent with simplicity of the model, as well as to reduce the traffic. This also avoids the tight linkage between the Communicator Server and agents. However, the mechanisms of the Communicator Server are suitable for a variety of protocols. Following figure 3.13 shows, how event handling is done between Communicator Agent and Communicator Server using UDP transport protocol.

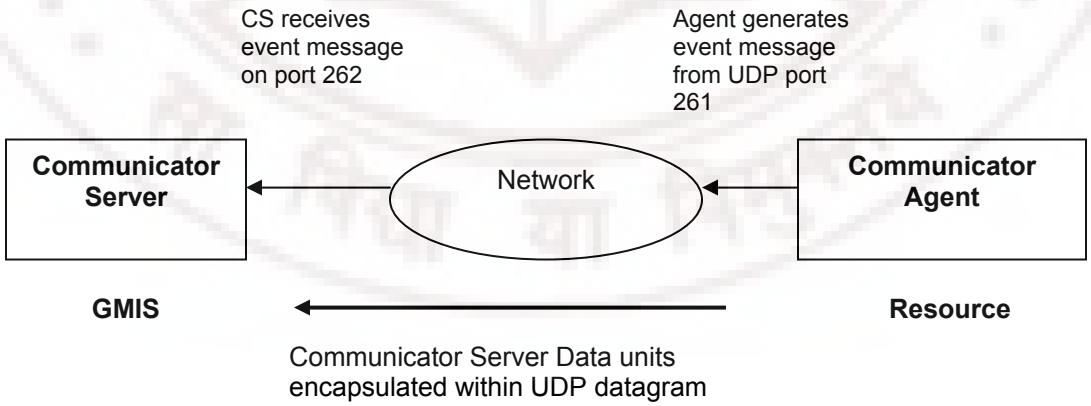

Figure 3.13: UDP Transport – Asynchronous Event

The following figure 3.14 shows how data exchanged between communicator server and communicator agent using UDP transport.

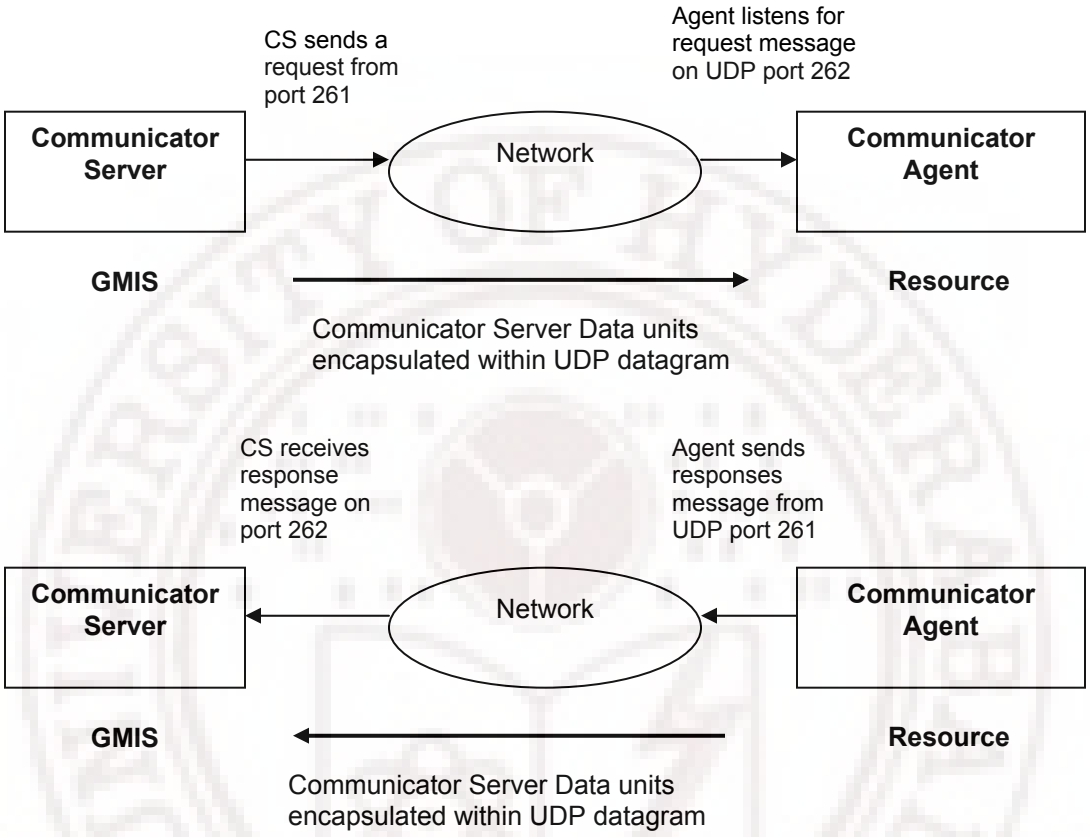

Figure 3.14: UDP Transport – Normal Send – Response exchange

3.8.1 Packet Data Representation in Communicator Server

Management information communicated by Communicator server is represented according to the subset of the Abstract Syntax Notation One (ASN.1) language [ASN.1, 1987] that is specified for the definition of non-aggregate types in the Structure of Management Information (SMI). This extends this tradition by utilizing a moderately more complex subset of ASN.1 for describing managed objects and for describing the protocol data units used for managing those objects. In addition, the desire to ease eventual transition to OSI-based network management protocols led to the definition in the ASN.1 language of an Internet-standard Structure of Management Information (SMI) [RFC1065] and Management Information Base (MIB) [RFC1066].

The Communicator Server uses only a subset of the basic encoding rules of ASN.1. Namely, all encodings use the definite-length form. Further, whenever permissible, non-constructor encodings are used rather than constructor encodings. This restriction applies to all aspects of ASN.1 encoding, both for the top-level protocol data units and the data objects they contain.

ASN.1 is more than just syntax. It is a formal language developed jointly by CCITT (now ITU-T) and ISO for use with application layers for data transfer between systems. It is also applicable within the system for clearly separating the abstract syntax and the transfer syntax at the presentation layer. Abstract Syntax is set of rules used to specify data types and structures for storage of information. Transfer syntax represents the set of rules for communicating information between systems. Thus, abstract syntax would be applicable to the information model and transfer syntax to the communication model. The abstract syntax can be used with any presentation syntax, depending on the medium of presentation. The abstract syntax in ASN.1 makes it independent of the lower-layer protocols. ISO 8824/X.208 standards specify ASN.1. The algorithm to convert the textual ASN.1 syntax to machine readable code is called basic encoding rules (BER) and is defined in ISO-8825/X.209 standards.

ASN.1 is based on the Backus system and uses the formal syntax language and grammar of the Backus-Nauer Form (BNF).

3.9 Grid Middleware

The GMIS framework is generic framework and can be integrated with other grid platforms like Globus [Foster, I, C. Kesselman, 1997] and other grid services like schedulers [Pramod Kumar, 2006]. GMIS provides Application Programming Interfaces (APIs) to access Topology manager and Database to access resource information.

3.10 Viewer

Viewer is a web based application. In the GMIS framework the web server chosen is Tomcat Apache web Server. Icons are used to represent grid, cluster and node. Viewer retrieves resource information through Monitoring Service.

Following are the main user interfaces provided in the viewer:

- **User Management** – used for authentication purpose and to register new users.
- **Discovery of resources** – to discover resources in grid by using Internet protocol addresses.
- **Resource Selector** – to select a resource based on the inputs from user, to generate activation graph and to execute job.
- **Monitoring of resources** – to monitor resources at grid level, cluster level and node level. This provides jobs status also.

3.11 Distributed GMIS Architecture

The distributed GMIS architecture provides the ability to distribute the Management Information Base (MIB) and resource information to multiple Grid Management Information Servers. Every GMIS allows users to have transparent access to management data irrespective of the GMIS on which the data is located. The data may reside in the local GMIS or on a remote GMIS in another geographical region. This architecture is centered on the concept of managed objects, and is shown in Figure 3.15

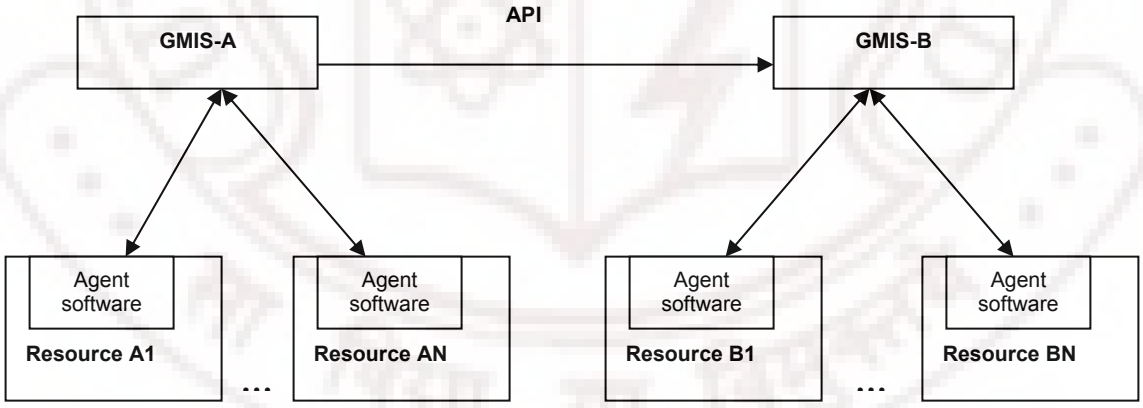

Figure 3.15: Distributed GMIS Architecture

Access to all objects is achieved through a hierarchical structure called Management Information Tree (MIT). The MIT follows the globally defined object naming convention. When a connection is initiated by the user using Application Programming

Interface (API) from GMIS A to GMIS B, the local MIT of GMIS B is mounted into GMIS A, and becomes visible in the viewer of GMIS A.

Distributed GMIS architecture contains the following components:

- **Resources** – Every physical resource on grid network is represented by one or more objects in GMIS topology manager, which we call them as managed objects. Performing an action like retrieving status information on a managed object is similar to performing that action on the corresponding resource.
- **Agent software** – This software provides objects level communication procedures between the managed resources and the GMIS. For instance, performing GET and SET procedures on managed resources.
- **Grid Management Information Server (GMIS)** – GMIS provides information services. It contains a relational database and a Management Information Tree (MIT) in topology manager.

GMIS maintains resource information and relationships of all managed resources on a grid. Multiple GMIS connections are possible so that all Grid Management Information Servers information appear as one GMIS to any user. The GMIS supports dynamic creation, deletion and updating of objects in the MIT.

3.11.1 GMIS-GMIS Communication

The main function of the GMIS is to provide data about managed resources to its client services. The term managed resources includes both the physical resources connected to the network, such as nodes, as well as software resources such as applications.

Naming of managed resource instances is based on a containment relationship. Containment can be visualized as a directed graph with each directed edge pointing from a contained managed resource to a containing managed resource. Containment, as used for naming, implies a hierarchy.

A management system such as the GMIS provides services to a predetermined set of managed objects in the MIT. Since the objects for which the GMIS provides

services are a subset of the objects in the global name space, the GMIS must be able to act as a manager as well as an agent. The division of roles is determined by the configuration of the GMIS. When a request is made for an object which is non-local to the GMIS, it is forwarded to the appropriate agent. In this case the GMIS acts as a manager. If the object is local to the GMIS, the information is retrieved and returned to the requester. In this case, the GMIS acts as an agent. The manager role is achieved by allowing the user to configure parts of the MIT represented by the GMIS as being non-local. Thus, the requests to that part of the MIT are forwarded to the agent configured to manage that part of the tree.

GMIS-GMIS communication is achieved when two or more GMISs are connected and set up to forward requests for managed objects in manager and agent roles. The GMIS forwarding a request is acting in the manager role, while the GMIS receiving the request is acting in the agent role. To an application, it is transparent as to whether the object being accessed is local or non-local. The GMIS-GMIS communication (GGC) between GMISs makes it transparent.

3.12 Summary

This chapter discussed about the GMIS framework architecture and it's components for resource management in grid environment. GMIS has been developed to address the resource management requirements like resource discovery, resource monitoring and brokerage. It also discussed how GMIS framework can be distributed in the grid network.

Next chapter discusses on the design aspects of GMIS framework along with data flow diagrams and data models.

CHAPTER 4

GMIS Framework Design and Resource Management Algorithms

The main objective of GMIS framework is to provide resource management services like discovery, monitoring and brokerage of resources in a grid network. This chapter discusses about the GMIS framework design with data models, data flow diagrams and algorithms. Data flow diagrams provide the communications and interfaces between the GMIS components. It also discusses about the design of resource's agent software.

4.1 Introduction

GMIS provides information regarding the operating status of different resources to the grid users. The main services of GMIS framework are, keep track of hardware and software resources in a grid, poll resources and receive the dynamic state of the grid resources attributes such as load, memory and disk information of all the managed resources, process job requests from grid users and provide resource latest information to users such as current jobs running, current load of resources and status of resources. GMIS also does some degree of fault recovery, such as resubmission of the jobs in case of failures. As discussed in Chapter 3, the components contribute to the working of GMIS are Discovery Server, Data Collection Manager, Topology Manager, Data Handler, Monitor Service, Resource Selector, Viewer and Communicator Server. Interactions among these GMIS components are shown in the figure 4.1.

In this figure 4.1, resources are physical resources in the grid network. GMIS Communicator Server communicates with the resources on behalf of GMIS components. Each physical resource on grid network is represented by one or more objects in GMIS topology. Each physical resource contains a communicator agent which will respond to the GMIS communicator server requests. These agents also send events on behalf of resources based on resource status change or job status change.

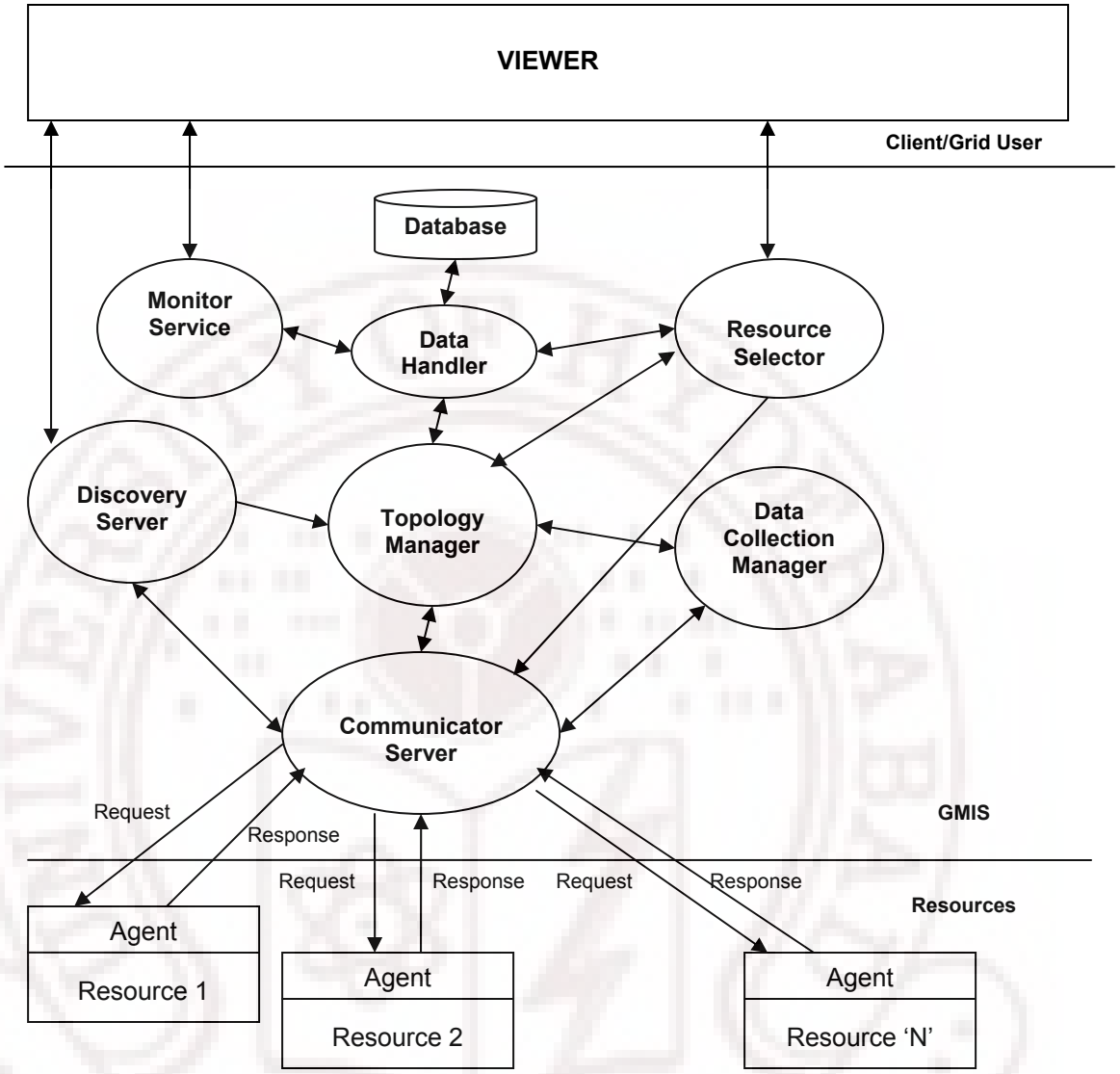

Figure 4.1: GMIS Components Interaction

Viewer provides user interface to access GMIS services. Agents of corresponding resources respond to the GMIS requests. The following table 4.1 shows the GMIS components interaction with other GMIS components.

GMIS Component	Interacts with
Discovery Server	Topology Manager, Communicator Server
Data Collection Manager	Topology Manager, Communicator Server

Database	Data Handler (DH)
Topology Manager	Data Collection Manager, Communicator Server, Data Handler
Monitor Service	Data Handler
Resource Selector	Topology Manager, Data Handler, Communicator Server
Communicator Server	Discovery Server, Data Collection Manager, Resource Selector, Communicator Agents
Agent Software	Communicator Agent (CA), Resource Discovery Agent (RDA), Critical Information Agent (CIA)
Viewer	Discovery Server, Monitoring Service, Resource Selection

Table 4.1: GMIS Component Interaction with other GMIS Components

The important design considerations discussed in Chapter 1 are used in designing GMIS framework.

4.2 Discovery Server

Resource discovery on grid network is a vital function in grid resource management system. Discovery server provides a simple and convenient way to add managed resources to the GMIS topology.

4.2.1 Dynamic Grid Network Discovery

In GMIS dynamic grid network discovery takes place in three levels. The first level is node level, which uses an Internet protocol address or Fully Qualified Domain Name (FQDN) is used to discover resource information. Second and third levels are cluster level and grid level where a range of Internet protocol addresses are used to discover resources. A user interface is provided on viewer to provide Internet protocol addresses or FQDN. Discovery Server (DS) processes requests from the viewer to discover resources with the given IP addresses or FQDN. DS sends a message to the agent software running at resources using these IP addresses, through the communicator server. Upon receiving the request, agent running at the resource

collects the resource information including application information and sends information to the DS through communicator server (CS). DS sends a message to the Topology Manager (TM) to update the topology with the new resource information. Topology Manager updates resource topology and database with the resource information. Data flow diagram of this process is shown in figure 4.2.

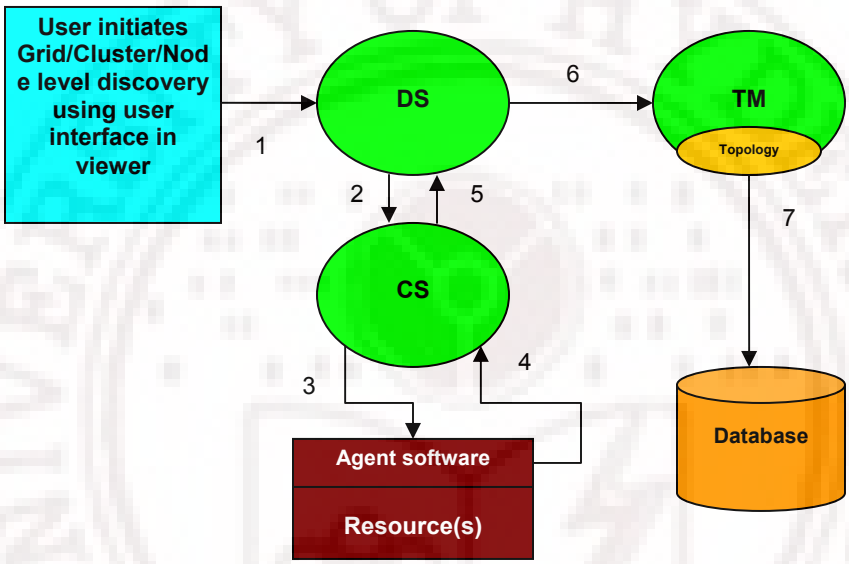

Figure 4.2: Data Flow Diagram of Dynamic Network Discovery

1. To discover resources on grid, user initiates Grid/Cluster/Node level discovery using the interface provided in the viewer.
2. Discovery server sends a message to the communicator server (CS) to discover resources on grid network.
3. CS broadcasts message to all the IP addresses mentioned in user interface.
4. Agent software that runs on the resource(s) responds to the broadcast message sent by the CS with resource information.
5. CS forwards resource information to DS.
6. DS forwards resource information to Topology Manager (TM). Topology Manager updates hierarchical resource management information tree and stores in cache storage.
7. Topology Manager updates the database with resource information.

4.2.2 User based Resource Discovery

In user based resource discovery, resources information is added manually to the GMIS using user interface provided on the viewer. Resource information provided by the user will be sent to Discovery Server (DS). Upon receiving the user information, Discovery Server validates and sends resource information to Topology Manager(TM). TM adds resource information in GMIS topology and data base. Data Collection Manager (DCM) polls the resource through communicator server, to verify the resource information provided by the user and to get the latest status of resource. Resource agent software running on the resource responds to the DCM poll request. DCM forwards the latest resource information to TM. TM updates the topology and database with the latest resource information. This process is shown as a data flow diagram in the following figure 4.3.

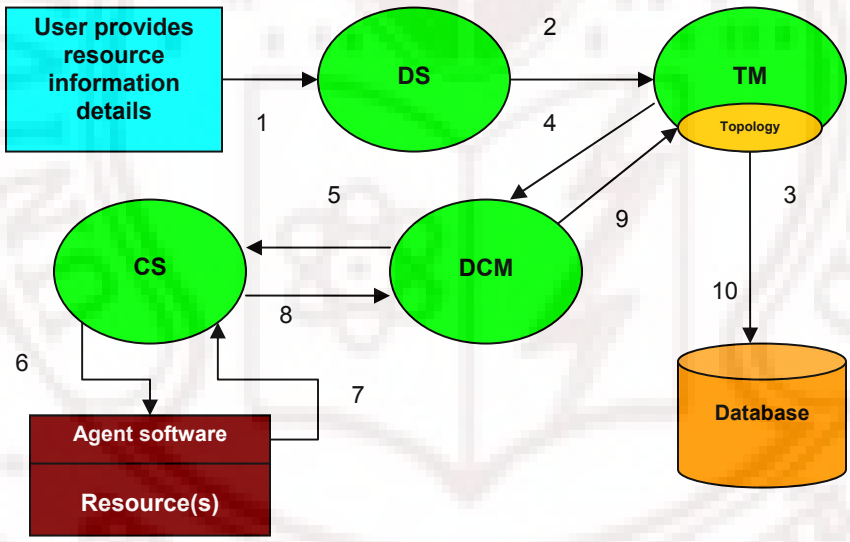

Figure 4.3: Data Flow Diagram of User based Resource Discovery

1. User provides resource information details on user interface provided on the viewer. This information is passed to the Discovery Server (DS).
2. DS validates the data and sends message to Topology Manager (TM). TM adds resource information to the GMIS topology.
3. TM adds the resource information in database.

4. TM sends a request message to Data Collection Manager (DCM) to verify the resource details.
5. DCM sends a request to the Communicator Server (CS) to verify and to get the latest status of the resource.
6. CS sends a request to the resource agent software of that resource.
7. Resource agent software responds to the CS request.
8. CS forwards the resource information to the DCM.
9. DCM updates topology manager with the latest resource information.
10. TM updates the latest resource information in topology as well as Database.

4.2.3 Discovery Server Algorithm

Algorithm Discovery Server:

1. *Initialize the IPC library and create two communication channels. One channel is for receiving messages from the viewer and communicator server. Other channel for sending messages to communicator server and topology manager.*
2. *For each received message*
3. *BEGIN*
4. *Parse the message.*
5. *If (message is from viewer)*
7. *BEGIN*
8. *If (message type is DYNAMIC_RESOURCE_DISCOVERY) then*
9. *BEGIN*
10. *Prepare a message with Message type as RESOURCE_DISCOVERY with resource IP address*
11. *Send the message to communicator server.*
12. *END*
13. *Else*
14. *if (message type is USER_RESOURCE_DISCOVERY) then*
15 . *BEGIN*
16. *Validate the user inputs*
17. *Prepare a message with Message type as ADD_RESOURCE*
18. *Send the message to topology manager*
19. *END*
20. *END*
21. *Else*
22. *if (message is from communicator server)*
23. *BEGIN*
24. *If (resource information is valid information)*
25. *Forward the message to topology manager*
26. *END*
27. *Else*
28. *Ignore the message.*
29. *END*

4.3 Topology Manager

The main objective of topology manager is maintenance of managed resource object data of various resources and maintenance of object hierarchy.

The functionalities of Topology Manager are:

1. Caching of Resource Information at startup for quicker access to other services.
2. Interfaces with DS to get the newly discovered resource information.
3. Updates resource availability (addition/deletion) information based on requests from Data Collection Manager.
4. Correlates event information and poll response from resources through DCM.
5. Updates the resource info in GMIS topology and database.
6. Maintains submitted job information in cache memory.

At the time of initialization GMIS retrieves the resource information from the database, and resource information is populated into the topology by topology manager. It verifies that resource is still available for that GMIS or not through data collection manager. If resource is still available then the resource information will be retained. Otherwise that resource will be deleted from the topology as well as from database. Topology manager updates the resource information in topology and database based on the information retrieved by the Discovery Server or Data Collection Manger.

4.3.1 Topology Manager Algorithm

Algorithm: Topology Manager

1. *Initialize the IPC library and create two communication channels, one for sending and other for receiving messages from other GMIS components.*
2. *Initialize MIB in the shared memory.*
3. *Retrieve grids information from the database using data handler functions.*
4. *If (there is resource information in Database)*
5. *BEGIN*
6. *For each grid in database do*
7. *BEGIN*
8. *Create a topology object with object type as GRID*
9. *Initialize other attributes of GRID object with the retrieved information*
10. *Get all clusters information from the database for this grid.*
11. *For each cluster in database do*
12. *BEGIN*

```
13.                     Create a topology object with object type as CLUSTER
14.                     Initialize other attributes of CLUSTER object with the
                            retrieved information.
15.                     Initialize the parent id with the GRID ID
16.                     Get all node information for this cluster.
17.                         For each node in database do
18.                         BEGIN
19.                             Create a topology object with object type as
                                    NODE
20.                             Initialize other attributes of NODE object
                                    with the retrieved information
21.                             Initialize the parent id with CLUSTER ID
22.                         END
23.                 END
24.   END
25.   For each node added in the topology
26.   BEGIN
27.       Send a request to Data Collection Manager to poll the resource
              information
28.       Get the request id and start timer
29.       Mark as request retry as 1.
30.   END
31. END
32. else /* no resource information in DB */
33. BEGIN
34.       Create a default grid topology object.
35.       Create a default cluster topology object under the default grid topology
              object.
36. END
37. If timer expires then
38. BEGIN
39.       If request retry is less than 3
40.       BEGIN
41.             Send another request to Data Collection Manager to poll the
                    resource information
42.             Increase the request retry to 1
43.       END
44.       else
45.       BEGIN
46.             Get the parent cluster object
47.             Remove the node object from topology
48.             Delete node information in data base
49.             Delete all services/applications information under this node
50.             Get the list of children of cluster object
51.             If (number of children is zero)
52.              BEGIN
53.                 Get the parent grid object
54.                 Remove the cluster object from topology
55.                 Delete cluster information in database
56.                 Get the list of children of grid object
57.                 If (number of clusters is zero)
```

```
58.                         BEGIN
59.                             Remove the grid object from topology
60.                             Delete grid information in database
61.                         END
62.                 END
63.         END
64. END
65.  For each message received do
66. BEGIN
67.         if (message is from Discovery Server)
68.         BEGIN
69.                 Parse the message and retrieve parent id.
70.                 If (parent id exists)
71.                 BEGIN
72.                         Retrieve parent cluster information.
73.                         Add  the new node under this cluster in topology
74.                         Insert the new resource information in database.
75.                 END
76.                 else
77.                 BEGIN
78.                         Add the new node under the default cluster in topology.
79.                         Insert the new resource information in database
80.                 END
81.         END
82.         else if (message from Data Collection Manager)
83.         BEGIN
84.                 if (message type is “POLL_RESPONSE”)
85.                 BEGIN
86.                         Update the topology and the database
87.                 END
88.                 else if (message type is “RESOURCE_UPDATE”)
89.                 BEGIN
90.                         if (resource already exists in topology)
91.                         BEGIN
92.                             Update the topology and the database.
93.                         END
94.                         else
95.                         BEGIN
96.                             Retrieve the parent id of resource
97.                             if (parent id exists in topology)
98.                             BEGIN
99.                                 Get the cluster topology information using
                                        parent id
100.                                Create a new object in topology under this
                                        cluster.
101.                            END
102.                            else
103.                                Create a new resource object under the
                                        default cluster object.
104.                        END
105.                END
```

```
106.        END
107.        else if(message from resource selector)
108.        BEGIN
109.                Create a job object in cache memory.
110.        END
111.        else Ignore the messages
112.  END
```

4.3.2 Data Model for Topology

The data model for topology takes care of grouping and maintaining logical and hierarchical relations of varied information available with different types of resources that will be supported. The objects in each grid are grouped into scheduled poll-able managed object groups and non-pollable object groups. The poll-hierarchy of the managed objects is also modeled to take care of handling poll responses and event handling.

Each resource type is associated with a listId for the list of ManagedObjectGroups. Topology manager reads the data model and for each type of resource, creates resource objects, which will contain a list of ManagedObjectGroups. For those ManagedObjectGroups, which need to be schedule-polled, the flag is set to TRUE in the data model and all such ManagedObjectGroups are registered with the Data Collection Manager to be polled.

The following figure 4.4 shows the data model for topology. The representation is done as class diagram for convenience.

Example usage: To initialize all nodes and their managed objects, the ResourceTable will have a row {Resource_Name = NODE; ManagedObejctGroupListId = 100(some unique id); defaultStatus=UP (so that DCM can start polling without waiting for link checking)}. This row is read and for resource type NODE all instances will be retrieved from Database and as many Node Instance objects are created in the topology with their IP addresses etc.

Next, for the ListId = 100, the ManagedObejctGroupList is read to get all the ManagedObject groups. Note that each resource can have one or more Managed Object

groups. E.g., Node may have two managed object groups: {NodeState – poll every 3 minutes}, {LoadState – poll every 5 minutes}.

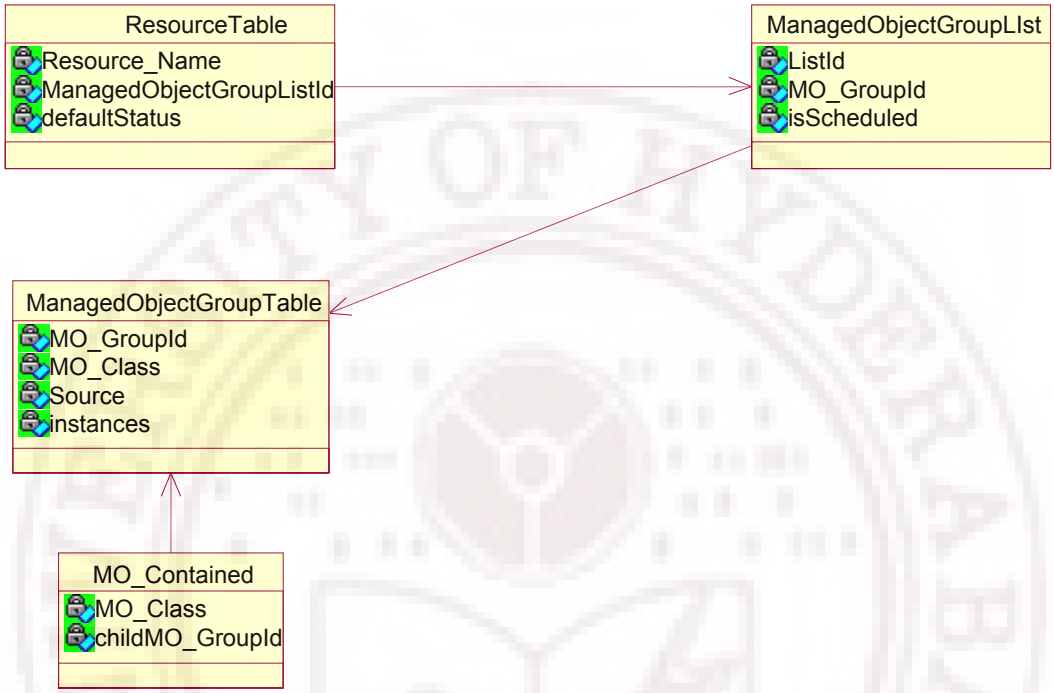

Figure 4.4: Data Model for Topology

Using the ManagedObjectGroupIds create ManagedObjectGroups by taking data from ManagedObejctGroupTable. For each ManagedObjectGroupId, a list of MO classes is given like NodeState, LoadState. If the number of instances is fixed then the source shall say 'FIXED' else need to look at database to get the number of instances.

Each managed object can be a container object like Node state, Hardware container etc. For all such objects, their child objects are added to the object repository by looking at MO_Contained table. For Node state, the child MO_Group will contain HW_Container. These are created recursively until there are no child objects. Later if the OS_container is added to NODE, just a new entry in the MO_Contained and appropriate group details in ManagedObejctGroupTable should be sufficient to create the new objects without change in the code.

4.4 Database

Each GMIS has a Database. This database includes information about the local managed resources. Whenever a resource is added/deleted or modified in grid, then the database gets updated by the topology manager.

Data handler provides an interface between GMIS components and the database. GMIS components can perform query operations on the database and update database tables using data handler. Data handler provides the following data handling functionalities.

- Handle the parameters associated with each resource.
    - Create the instance of resource.
    - Retrieve the instance of resource.
    - Modify the parameters associated with resource.
    - Retrieve the parameters associated with resource.
- Handle the hierarchy of the resources
    - Traversing through the hierarchy
        - Retrieve the parent of a given resource.
        - Retrieve the children of a given resource.

4.4.1 Data Handler Algorithm

Algorithm: DataHanlder

1. *Initialize IPC library and create two communication channels to send and receive messages.*
2. *Check whether GMIS database exists or not.*
3. *If GMIS database does not exist*
4. *Create GMIS database with the GMIS DB schema*
5. *For each received message*
6. *BEGIN*
7. *Parse the message.*
8. *Connect to the database.*
9. *if (connection is successful)*
10. *BEGIN*
11. *if (request type is "RESOURCE_ADD")*
12. *Insert the resource information in the appropriate table.*
13. *else if (request type is "RESOURCE_UPDATE")*
14. *Update the resource information based on resource id.*

```
15.                 else if (request type is “RESOURCE_DELETE”)
16.                         Delete the resource information based on resource id.
17.                 else if (request type is “RESOURCE_RETRIEVE)
18.                         Query the resource information and send the information.
19.         END
20.         else
21.                 Send DB_FAILURE message to the requestor component.
22. END
```

4.5 Data Collection Manager

Data Collection Manager (DCM) collects resource status periodically. DCM performs resource data collection requests based on the contents of topology object. Agent software running on the resources will respond to these requests. After creating a topology object, it schedules a data request for that object. When topology object is deleted, it stops collecting data from the resource.

DCM is responsible for finding faults and failures of managed resources. It polls the status information of the resources, handles events and updates the database through topology manager.

4.5.1 Polling and Parsing

Resource polling is a mechanism to periodically check the status of any resource connected to a GMIS managed grid network with the help of timer. In GMIS polling is done for resource node for every 10 minutes to get the status of the node. After response is received, it parses response and based on received response, topology and database gets updated, so that these changes will get reflected on resource monitoring view of viewer. If received response for a node is not operational, then it polls immediately its descendants i.e., memory, CPU, load and applications.

DCM polls each of the managed resource periodically. DCM gets the resources configuration information from topology manager. It sends resource status requests through Communicator Server to all managed resources to obtain their latest status. Upon receiving the new status information, it sends this information to the viewer through topology manager, data base and monitor service. DCM interacts with topology manager, and communicator server using Inter Process Communication (IPC) messages. This process is shown as a data flow diagram in the following figure 4.5.

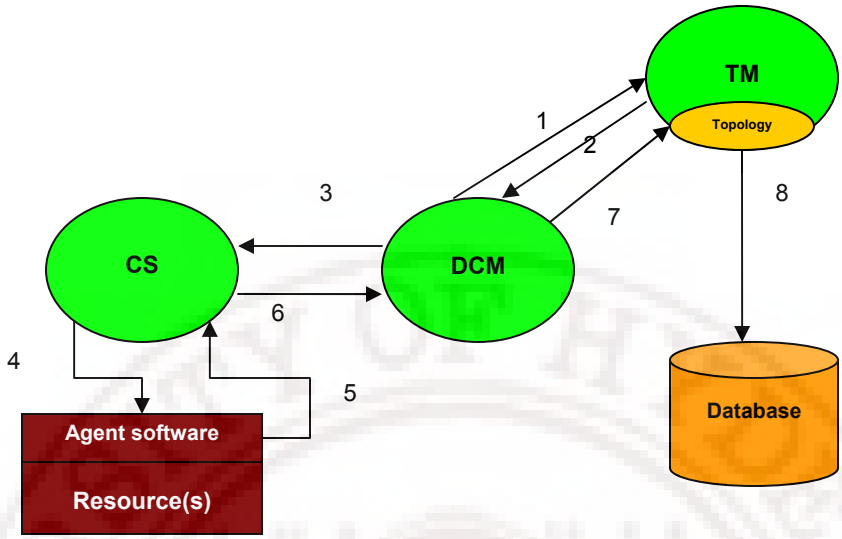

Figure 4.5: Data Flow Diagram for Periodic Monitoring of Resources

1. Data Collection Manger (DCM) requests Topology Manger (TM) for schedule poll able resources and their information.
2. TM responds to the DCM request, with the resources information like FQDN, Internet protocol address, object identifier and hierarchy.
3. DCM sends a message to Communicator Server (CS) to poll for current status of resources using information provided by the TM.
4. CS sends a poll request to resource.
5. Resources' agent software collects resource information and responds to the poll request.
6. CS forwards the poll request response to DCM.
7. DCM parses the response received from the resources' agent software and sends updated resource information through internal message to TM.
8. TM updates topology and database with the latest resource information.

4.5.2 Event Handling

Event handling provides a mechanism at the GMIS for managing events generated by resource agents which are routed to the GMIS Communicator Server. Typical events are:

- Resource update information (addition/deletion/modification)

- Indication of a resource change of status
- Indication of overload
- Utilization of resource components i.e. threshold values exceeded
- Job execution information

Communicator Server provides centralized reporting of all events generated by resource agents across the grid network. Communicator Server sends events to Data Collection Manager (DCM). DCM correlates various events generated by resources to get the latest status of resource. This status is passed on the viewer through topology manager, database and monitor service.

The following types of events are supported in GMIS. Communicator Agents generate these types of events.

- **Resource Update Event** – This event will be generated whenever a component or service is added/deleted/modified to resource.
- **Resource Status Change Event** – This event indicates that the one of resource component's exceeded a threshold value.
- **Equipment Failure Event** – This event will be generated whenever there is a resource failure due to hardware problems.
- **Availability Event** – This event is generated whenever there is a change in resource availability to the grid users.
- **Job Execution Start Event** – This event is generated whenever job execution starts.
- **Job Execution Completed Event** – This event is generated whenever job execution completes.
- **Job Execution Failure Event** – This event is generated by resource when resource is unable to access input files or unable to upload the output files. If it is unable to access the input file for job execution then, the status of job execution set to Job_Input_Download_Failed. If the resource is unable to upload the output file, then Job_Output_Upload_Failed status will be set to the status of job execution.

These events will be generated by the resource agents on behalf of resources. These resource agents are referred as Communicator Agents. For instance, whenever a resource is added, modified or deleted an event will be generated by resource agent software. This event reaches to Data Collection Manager (DCM) through Communicator Server of GMIS. DCM parses the message and updates Topology Manager (TM) to update GMIS topology. Topology manager updates topology and database about the updated information. This process is called as dynamic event based discovery. Dynamic event based discovery process data flow diagram is shown in the following figure 4.6.

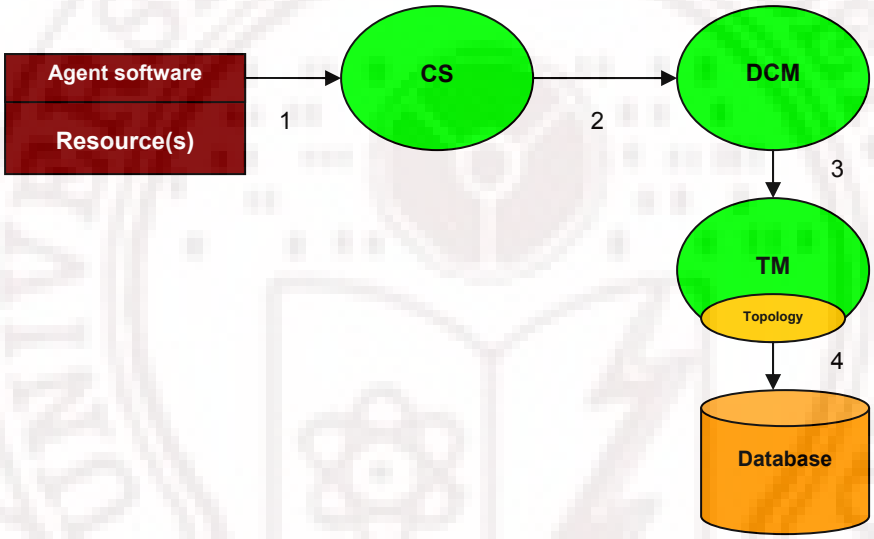

Figure 4.6: Data Flow Diagram of Event Handling in GMIS

1. When a new resource or application is added/deleted/modified a RESOURCE_UPDATE event sent by the agent software which is running on resources to Communicator Server (CS) of GMIS.
2. CS forwards the RESOURCE_UPDATE event to Data Collection Manager (DCM).
3. DCM parses the RESOURCE_UPDATE event and sends a message to Topology Manager (TM) to update topology.
4. TM constructs/updates hierarchical resource management tree and stores in cache storage and updates the database with resource information.

4.5.3 Data Collection Manager Algorithm

Algorithm: Data Collection Manager:

1. *Initialize IPC library and open two communication channels for sending and receiving messages.*
2. *Get all instances of pollable objects from the topology manager.*
3. *For each object do*
4. *BEGIN*
5. *Get IP address of resource*
6. *Prepare the list of attributes to the message parameters list along with object identifiers*
7. *Set a transaction id to the poll request*
8. *Prepare a message structure*
9. *Send message to communicator server*
10. *Start timer*
11. *END*
12. *If timer expires again start polling of resources, follow steps 3 to 10.*
13. *For each message received do*
14. *BEGIN*
15. *Parse the message*
16. *if (message is from Communicator Server)*
17. *BEGIN*
18. *if (message type is "POLL RESPONSE")*
19. *BEGIN*
20. *Delete the transaction id*
21. *Forward message to topology manager to update topology and database*
22. *END*
23. *if (message type is "EVENT")*
24. *BEGIN*
25. *Parse the event*
26. *Prepare a message*
27. *Send message to Topology Manager*
28. *END*
29. *END*
30. *else if (message is from Topology Manager)*
31. *BEGIN*
32. *Poll resources using step 3 to 10.*
33. *END*
34. *else Ignore the messages*
35. *END*

4.5.4 Data Model for Poll Response and Event Handling

Poll-response handling and event handling for Status Change Events (SCE) or resource update events have a common behavior based on the new state of the managed object. Such behavior is captured in the following data model figure 4.7.

In the first part of the data model general event handling is captured, where-in event acknowledgement is covered. If an event required to be acknowledged, then it shall contain an acknowledgement Object Identifier (OID) and value. If it is a status change event the status of the resource Managed Object is updated, else, based on the appropriate action code specific processing is done.

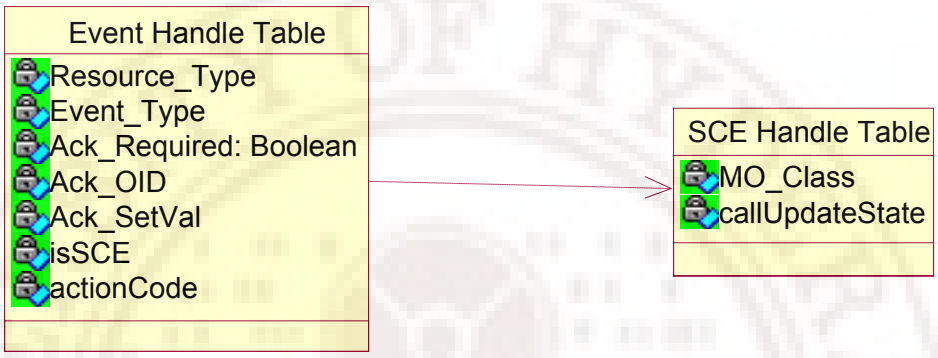

Figure 4.7: Data Model for Poll Response and Event Handling

Second part of the data model looks at updating the status of a managed object, which is common for both status change event and poll-response. Whenever an object status changes, Topology Manager updates topology and as well as database.

Example usage: The Event Handle Table contains the list of all types and events and the action that needs to be taken. For instance, a Node Availability event has an entry in the table with {NE_type = NODE, Event_Type = Node_Avail_event, Ack_Required = TRUE, ACK_OID = 1.2.3.4.3.2.2.22.0, ACK_Value = 1; isSCE = FALSE, actionCode = 0}. Based on this the event handler shall handle the availability event by picking up the Ack_Oid and sending the Ack_value to that Node.

If the event is a status change event, which does not require acknowledgement like a Node-Hardware-Container status change event, it will have appropriate values and the isSCE flag will be TRUE. Based on that the Event Handler looks at the SCE Handle Table and finds that for the MO_Class Node-Hardware_container, and update the object status. The Managed object instance's updateState() method is invoked by Data Collection Manager.

The updateState() method extracts the new status of resource managed object and constructs an internal message. This message will be sent to the Topology Manager to update the topology and database.

4.6 Monitor Service

Monitor Service provides inputs to the viewer to show the status of resources. The main functionalities of the Monitor Service are User Management and to retrieve GMIS managed resource information. When a grid user logged into the GMIS using user interface provided on viewer, authentication request comes to monitor service. Monitor service authenticates user information by checking user details in database through data handler. Monitor Service validates and updates user information in GMIS database for new users. Once the user authentication is done, GMIS managed resource information is provided to the viewer to display the resource information. Monitor Service periodically responds to viewer's resource information request. The basic Monitoring Service process data flow diagram is shown in the following figure 4.8.

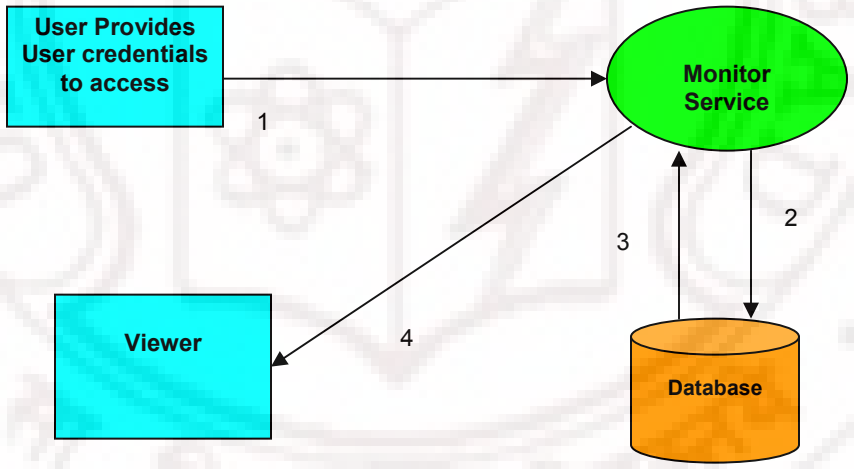

Figure 4.8: Data Flow Diagram for Monitor Service Process

1. User provides user credentials to access GMIS managed resource information. Viewer sends this request to Monitor Service.
2. Monitor Service validates the user information, and if the user is valid user, then it sends a request to Database through data handler for the resource information.

3. Monitor Service retrieves resource information from the database.
4. Monitor Service updates the Viewer with the available resource information.

4.6.1 Monitor Service Algorithm

Algorithm: Monitor Service

```
1. Initialize IPC library and create two communication channels, one for receiving
   and other one for sending messages.
2. For each message received do
3. BEGIN
4.      Parse the message
5.      If (message is from Viewer)
6.      BEGIN
7.          if (message type is USER_AUTHENTICATION)
8.          BEGIN
9.              Validate the user information, by querying user information from
                    database
10.             If (validation is successful)
11.                 Retrieve the resource information
12.                 Convert all resource information into XML format
13.                 Send to the viewer
14.             else
15.                 Send error message to viewer.
16.         END
17.         if (message type is RESOURCE_INFO)
18.         BEGIN
19.             Retrieve the resource information from Database using Data
                    Handler
20.             Convert retrieved information in XML format
21.             Send the retrieved resource information to viewer
22.         END
23.     END
24.    else
25.         Ignore the message.
26. END
```

4.7 Brokerage Service

The Brokerage Service performs a number of functions. The first step is the identification and selection of resources that best fits the needs of grid application. The broker will then submit jobs in the application to the chosen machines. The broker handles submission of jobs but not how the job is actually executed on the resource, as that is part of the resource management system that resides on the resource involved.

Resource Selector component of GMIS takes care of the resource broker functionality. Once jobs are being executed Data Collection Manger monitors resources and the progression of the jobs. The following figure 4.9 shows the brokerage service design in GMIS framework.

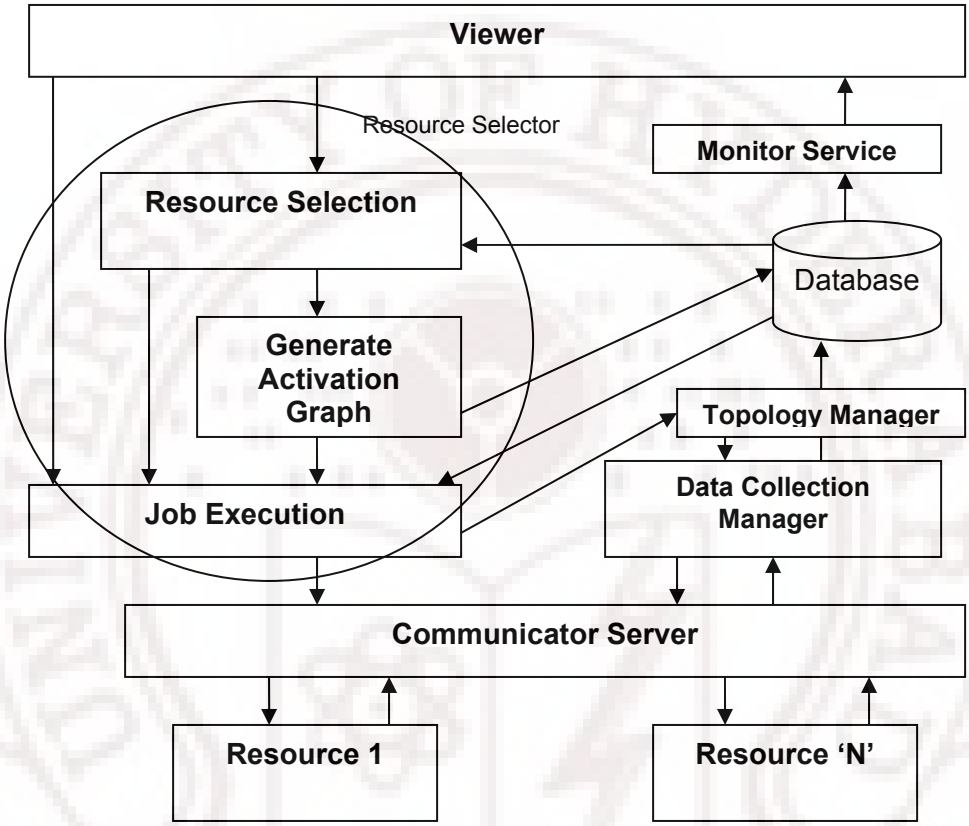

Figure 4.9: Resource Selector Service Design

User can submit the jobs from the user interface provided on viewer. All job requests are processed by the Resource Selector. Resource selector selects the appropriate resources for job execution, provides resource information to create activation graphs and provide interface to execute the jobs. Resource selector gathers all resource information from the database. Job execution request go to the resource agent through the communicator server. Data collection manager monitors the job execution status and status will get updated in database through topology manager. In GMIS framework, three types of job execution are provided:

- **Auto Job Execution** – In this type, user is not bothered which resource executes job. User selects a particular job to be executed and based on the resource availability Job execution will be done by the resource selector.

- **Job Execution on user defined resource** – In this type, user selects a resource on which a particular job requires to be executed.

- **Job Execution through activation graphs** – If a job can be divided into multiple tasks and tasks are to be executed on different nodes, then the user can create activation graphs. Generation of activation graphs can be done by using two methods. One is brokerage service itself which will create activation graphs automatically based on user inputs and other is creation of activation graphs by the user manually, by selecting different resources and applications available.

4.7.1 Resource Selection

Resource selection by grid user process data flow diagram is shown in the following figure 4.10.

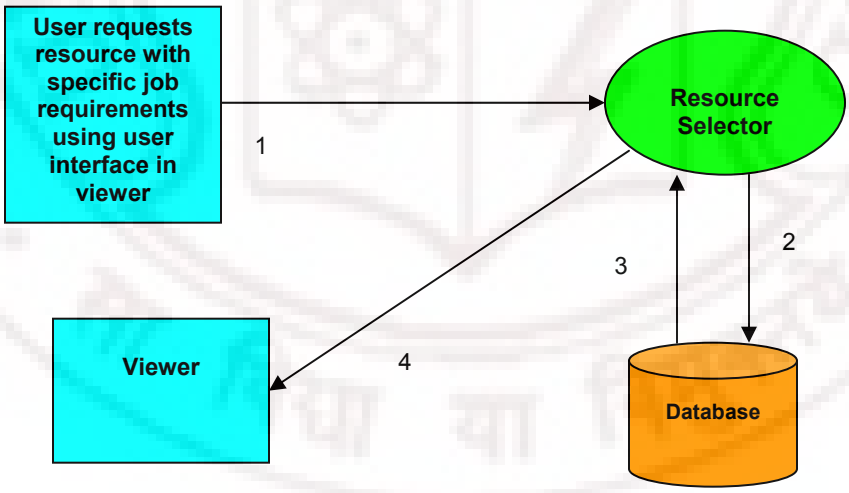

Figure 4.10: Data Flow Diagram for Resource Selection Process

1. Grid user requests for resources with specific job requirements like type of processor, memory size, application name by using user interface on viewer. Viewer sends this request to Resource Selector.
2. Resource Selector queries the database based on the user requirements using Data Handler methods.
3. Data Handler methods retrieves resource information from the database and sends to Resource Selector.
4. Resource Selector updates the Viewer with the available resource information.

4.7.2 Creation of Activation Graph

In a grid computing environment various services work in collaboration with each other. The output of one service acts as an input for another service. Each service after processing a input file will produce an output file. A graph can be created specifying the order in which various services are to be executed. The condition on execution and the input/output files of various services could also be specified. Such a graph is called as an Activation Graph.

A user interface is provided on viewer to invoke grid services. The following steps are required to be followed to create an activation graph:

- Choose a node on which task need to be executed.
- Choose a service on this node which is used for task execution.
- Specify input/output files
- Specify the dependency of execution for each task.

Activations graphs can be created manually or automatically. In Manual creation, user requires to select resources and has to specify all the required inputs, where as in auto generation, activation graph will be created automatically by just specifying a minimal inputs like application names, input files and output files.

4.7.3 Job Execution

Job execution process data flow diagram is shown in the following figure 4.11.

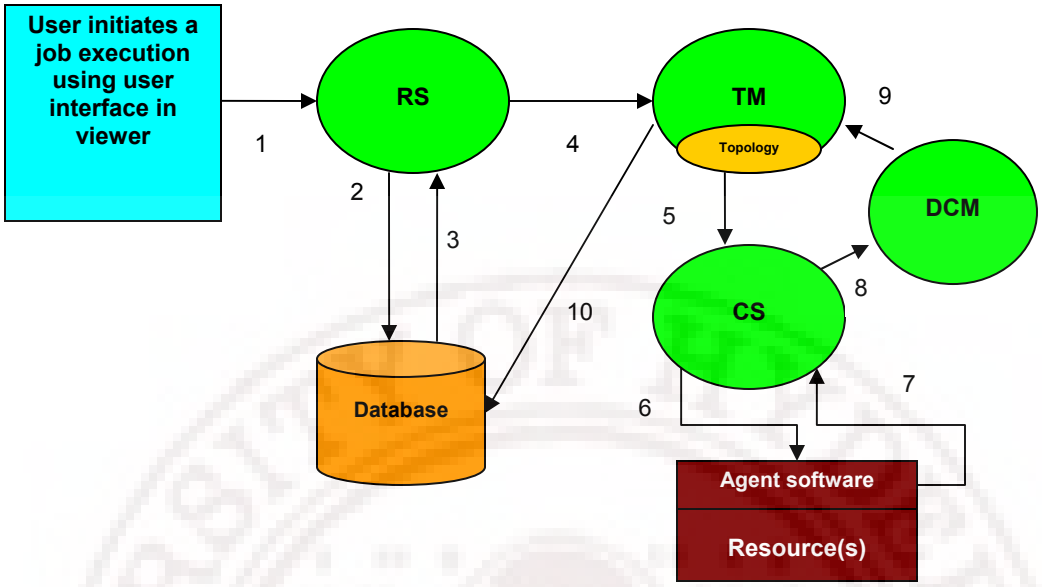

Figure 4.11: Data Flow Diagram for Job Execution Process

1. User initiates job execution using user interface on viewer. Viewer sends this request to Resource Selector (RS)
2. RS stores the job information in database.
3. RS retrieves resource information from the database.
4. Resource Selector sends a request to the Topology Manager(TM) with job details.
5. TM sends a request to the communicator server to submit the job on resource.
6. CS sends a request to Agent software to execute job.
7. Agent software on resources executes the job and sends the status of job to the CS.
8. CS forwards the status of job to the Data Collection Manager (DCM).
9. DCM sends update to the TM.
10. TM updates the database with the latest status of the job.

The above mentioned job execution process is a simple job execution process. For efficient utilization of resources, fuzzy based scheduling algorithm [Pramod, 2006] can be used in GMIS job execution process.

4.8 Communicator Server

Communicator Server (CS) provides GMIS the ability to communicate with resources in grid. CS listens to responses and events from the resources, converts them into internal GMIS messages and sends them to the requested GMIS components. Communicator server maintains all the transactions data in a hash memory. The functionalities of communicator server include:

- Provides interface to send a request message to a resource on behalf of GMIS
- Creates and initializes the transaction and session tables
- Creates UDP/IP sessions
- Handles all communicator related messages like request/responses from GMIS components and events originated from the resources.
- Acts as an communicator agent for GMIS

Communicator server gets requests from Discover Server, Data Collection Manager, Topology Manager and Resource Selector. Communicator server communicates with resource agent with the help of Management Information Base (MIB) generated by the Topology Manager. This process is shown in the following figure 4.12.

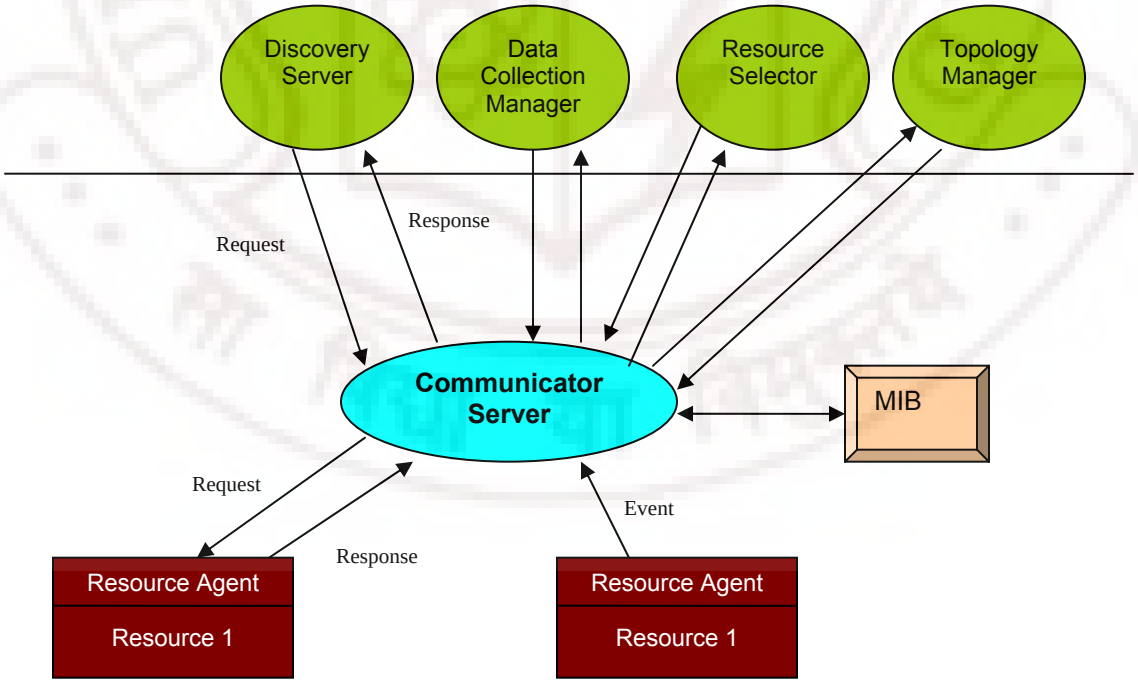

Figure 4.12: Communicator Server

4.8.1 Algorithm for Communicator Server

Algorithm: Communicator Server

1. *Initialize IPC environment and create two communication channels for sending and receiving messages to/from GMIS components.*
2. *Open two UDP/IP sessions one for listen and other for send messages from/to communicator agents.*
3. *Initialize the head and tail shared memory pointers to mib tree head and tail respectively.*
4. *For each message received do*
5. *BEGIN*
6. *Parse the message*
7. *if (messages from GMIS components)*
8. *BEGIN*
9. *Initialize the transaction data*
10. *Interpret the IP address of resource from the header of message*
11. *Add IP address to the transaction data*
12. *Get the MIB of requested resource*
13. *Update the appropriate hash table to accommodate this request*
14. *Create an communicator Packet Data Unit (PDU) for the request*
15. *Send message to the resource communicator agent using IP address and port 261.*
16. *Start timer*
17. *END*
18. *else if (messages from Resource agents)*
19. *BEGIN*
20. *Check whether transaction id exists in the message*
21. *If (transaction id exists)*
22. *BEGIN*
23. *Retrieve the transaction information based on transaction id from hash table*
24. *Create internal message*
25. *Send message to the GMIS components*
26. *Delete the transaction details from hash table*
27. *END*
28. *else*
29. *BEGIN*
30. *Validate the received Packet Data Unit*
31. *Retrieve the Managed object instance (MOI) from PDU*
32. *Retrieve the resource type and message type*
33. *Construct internal message*
34. *Mark the message as event message*
35. *Send message to the Data Collection Manager*
36. *END*
37. *END*
38. *else if (message is from remote GMIS)*
39. *BEGIN*
40. *if (message type is "JOB_EXECUTION")*
41. *BEGIN*

42. Initialize the transaction data
43. Retrieve the IP address and port of resource from the header of message
44. Add IP address to the transaction data
45. Get the MIB of requested resource
46. Update the appropriate hash table to accommodate this request
47. Create an communicator Packet Data Unit (PDU) for the request
48. Send message to the resource using IP address and default port 261.
49. END
50. END
51. else (ignore the message)
52. END
53. if timer expired
54. BEGIN
55. Retrieve the transaction from the hash table
56. Build transaction failed message
57. Send transaction failed message to requested GMIS component
58. END

4.9 Viewer

Grid users can use the viewer to access resource information, to discover resources on grid, for adding resources to grid and to submit jobs. Viewer is a web client which can be opened on any web browsers like Netscape or Mozilla firefox. Viewer uses Hyper Text Transfer Protocol (HTTP) to interact with the GMIS components through web server. Apache tomcat web server is used as a web server in GMIS. All classes related to interface between viewer and GMIS components are hosted by this web server. The following user interfaces are provided on the viewer.

1. User Management
2. Resources Discovery (Discover at node, cluster or grid level, add node, cluster or grid, add service, discover service and resource availability).
3. Job execution (Activation graph creation and execution, Job submission)
4. Resources monitoring at grid, cluster and node level.
5. Job execution monitoring

4.10 Agent Software

Agent software runs at each physical resource to provide resource information. Three types of agents are included as part of agent software.

- **Communicator Agent (CA)** – CA responds to communicator server requests includes job execution requests.
- **Resource Discover Agent (RDA)** – RDA collects resource specific details and generates events whenever there is a change in resource configuration.
- **Critical Information Agent (CIA)** – CIA sends any critical component change of the resource i.e., when CPU exceeds 80% CPU.

All agents use UDP/IP protocol to communicate with the GMIS communicator server as discussed in Chapter 3.

In operational setup these agents should be kept in the startup so that whenever the resource is booted these agents start automatically. In case the resource provider wants to use the system in a dedicated mode or wants to remove the resource from grid, for some reasons, then the resource provider could stop or kill these agents. If the resource provider stops the agents manually then before shut down RDA generates a resource delete event. Upon receiving this event GMIS updates its topology and database with this information. In case of killing agents, for the GMIS periodic request there will not be any response, so it will be treated as unavailable resource to the grid. Whenever resource provider wants to add the resource to grid, the resource owner can start this agent, by providing inputs that specifying which GMIS it needs to be added.

4.10.1 Algorithm for Communicator Agent

Algorithm: Communicator Agent

1. *Initialize IPC environment and create two communication channels for sending and receiving messages to/from RDA and CIA.*
2. *Open two UDP/IP sessions, one for listening and another for sending from/to GMIS communicator server.*
3. *For each message received do*
4. *BEGIN*
5. *if (message is from GMIS communicator server)*
6. *BEGIN*
7. *Parse the Packet Data Unit (PDU)*
8. *Extract the communicator server IP address and port number*
9. *if (request type is RESOURCE_DISCOVER)*
10. *BEGIN*
11. *Invoke Resource Discover Agent (RDA) to collect both static and dynamic resource information*
12. *END*

13. else if (request type is PHYSICAL_ RESOURCE_POLL)
14. BEGIN
15. Invoke Resource Discovery Agent (RDA) to collect only dynamic resource information
16. END
17. else if (request type is JOB_EXECUTION)
18. BEGIN
19. Create a new job id
20. Validate the job request details
21. If validation fails return appropriate error.
22. Create an entry in a hash table about the job execution with job id as a hash index
23. Invoke the corresponding application to execute
24. Send job execution information to CIA
25. Prepare a PDU with "JOB_EXECUTION_STARTED" with job id.
26. Send the PDU to GMIS communicator server
27. END
28. else if (request type is JOB_EXECUTION_POLL)
29. BEGIN
30. Retrieve the job id from PDU
31. Retrieve the job status from hash table based on job id
32. Prepare a PDU with job status
33. Send the PDU to GMIS communicator server
34. END
35. END
36. else if (message is from RDA or CIA)
37. BEGIN
38. if (message type is RESOURCE_DISOCOVER or PHYSICAL_RESOURCE_POLL or EVENT)
39. BEGIN
40. Pack the information into a PDU
41. Send the PDU to GMIS communicator server
42. END
43. else if (message type is JOB_EXECUTION)
44. BEGIN
45. Retrieve the job details based on job id
46. Update it's job status in hash table
47. Pack the information into a PDU
48. Send the PDU to GMIS communicator server
49. END
50. END

4.10.2 Algorithm for Resource Discovery Agent

Algorithm: Resource Discovery Agent

1. *Initialize IPC library and create two communication channels one for receiving and other for sending messages.*
2. *For each message received do*
3. *BEGIN*

```
4.      Parse the message.
5.      If (message is from Communicator Agent)
6.      BEGIN
7.      if (message type is “RESOURCE_DISCOVERY”)
8.          BEGIN
9.              Get static information like operating system, processor
                    family, number of processors using operating
                    system level commands.
10.             Get dynamic information like memory, disk space
                    available, and CPU idle time using system
                    commands.
11.             Pack both static and dynamic information into the internal
                    message format.
12.             Send this message to communicator agent.
13.         END
14.     else if (message type is “POLL_REQUEST”)
15.         BEGIN
16.             Get the component based on the object identifier (OID) of
                    the request
17.             Get the information of that particular component.
18.             Construct an internal message with the
                    resource/component  information.
19.             Send this message to communicator agent.
20.         END
21.     else
22.         Ignore the message.
23.     END
24.     else if (message is from Controller handler)
25.     BEGIN
26.         if (message type is “START” or “STOP”)
27.           BEGIN
28.             Prepare an internal message with message type as
                            RESOURCE_UPDATE event.
29.             Send this message to communicator agent.
30.           END
31.         else
32.             Ignore the message.
33.     END
34. END
```

4.10.3 Algorithm for Critical Information Agent

Algorithm: Critical Information Agent

1. *Initialize IPC library and create two communication channels to send and receive messages to/from the communicator agent.*
2. *Get the threshold values for different components from the threshold configuration file. Currently only two components are monitored, one is CPU usage and other is memory usage.*
3. *Get CPU and memory usage using the system commands.*

4. Start timer
5. If CPU or memory usage is exceeding the threshold values then
6. BEGIN
7. Prepare an internal message with event type as CRITICAL_EVENT.
8. Send this message to communicator agent.
9. END
10. If timer expires
11. BEGIN
12. Repeat steps 3 to 10.
13. END
14. For each received message
15. BEGIN
16. Parse the message
17. If (message is from Communicator Agent)
18. BEGIN
19. Store the job information details in internal memory.
20. Start execution of the job.
21. Start job timer.
22. END
23. END
24. If job timer expires
25. BEGIN
26. Get the status of job execution.
27. Prepare a message with the job execution status.
28. Send the message to Communicator Agent.
29. If (job execution status is not "JOB_EXECUTION_COMPLETED) then
30. BEGIN
31. Start job timer again.
32. Repeat steps 24 to 33.
33. END
34. END

4.11 Inter Process Communication

The components or processes of the GMIS communicate with each other through a set of library calls which contain the Inter Process Communication (IPC) library called message-queue IPC library. This IPC library is created to use message queues, which are resident as shared memory in the kernel of operating system. This new library handles communication among the GMIS components.

A socket-based IPC library has developed to communicate between GMIS communicator server and resource agent software. The following figure 4.13 shows the GMIS IPC mechanism.

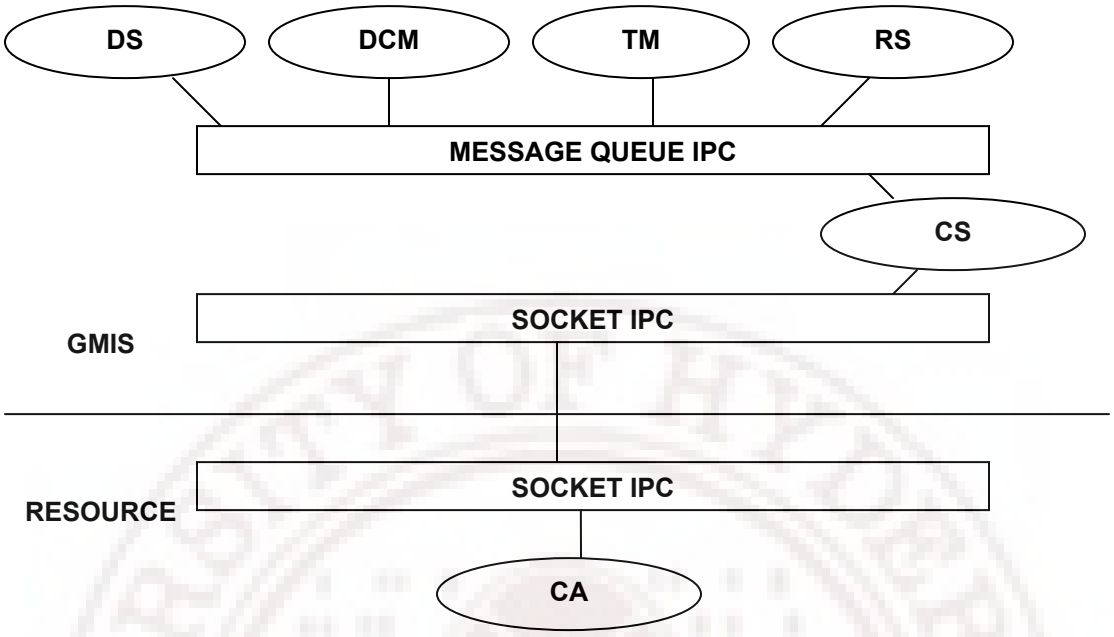

Figure 4.13: GMIS IPC Mechanism

4.12 GMIS-GMIS Communication

To manage multiple grid network resources and increase resource availability, multiple GMISs can be used across the grid network. In multiple GMIS architecture the advantages are:

- Network traffic and managing resources can be distributed among GMISs.
- When two or more GMISs are connected and need to transfer resource information, then one Communicator Server of GMISs acts as agent and other acts as manager.
- Grid users access resources and act on consistent management information, without regard to location.
- All information is gathered via management requests to GMIS.
- Any GMIS can locate requested managed information and execute requests, thereby distributing the processing load.

To achieve the GMIS – GMIS communication the following pre-conditions should meet:

1. Both running GMISs servers should have trusted host permissions.
2. GMISs users account should exist in both the machines.
3. Ensure that /etc/hosts of the GMIS contain the hostnames and IP addresses of all other GMISs to be connected.

The design of GMIS-GMIS is shown in the following figure 4.14. In this figure, GMIS-1 connected to GMIS-2 and all resource information of GMIS-2 is mounted to the Resource Management Information Tree (RMIT) of GMIS-1. An Application Programming Interface called GMIS-GMIS communicator is provided to connect the multiple GMIS-GMIS. This process is part of Communicator Server of GMIS.

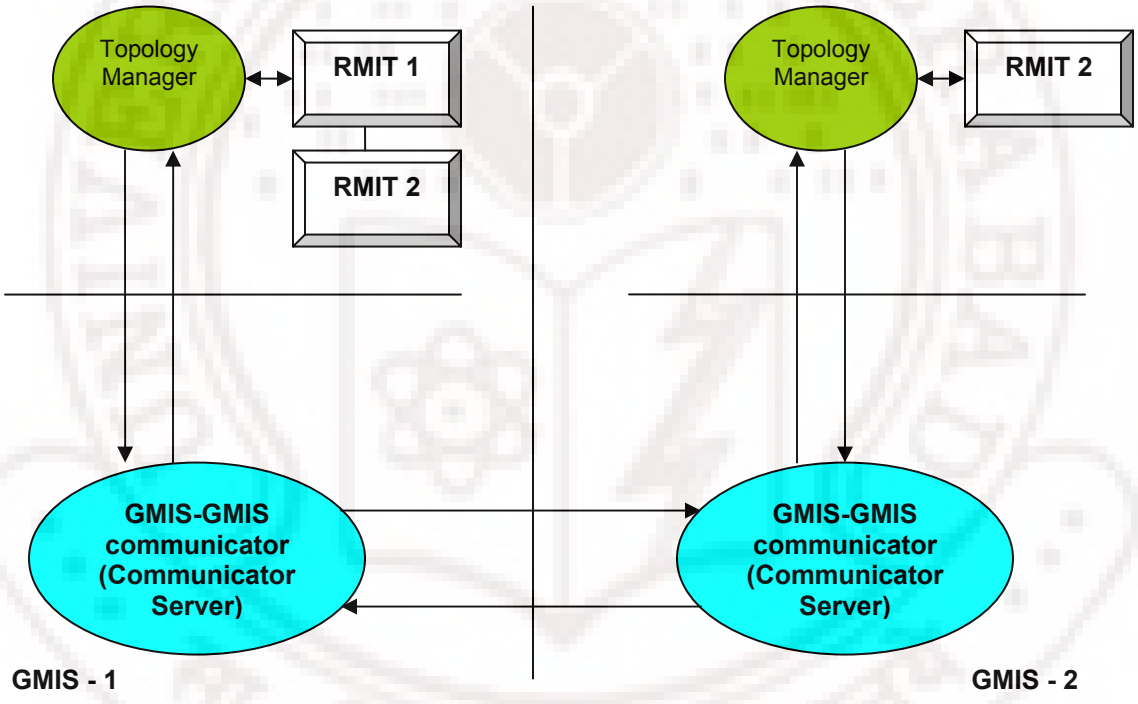

Figure 4.14: GMIS-GMIS Communication Design

4.13 Summary

This chapter discussed about the design of GMIS components with data flow diagrams and data models. Algorithms for various tasks are discussed in this chapter. This chapter also discussed about the communication methodologies between

Communicator Server of GMIS and grid resources. Finally it discussed about the GMIS-GMIS communication design.

Next chapter discusses about GMIS processes and interface modeling with experimentation results.

CHAPTER 5

GMIS Processes and Interface Modeling with Experimentation Results

This chapter discusses about the GMIS framework processes and interface modeling with the help of process flow charts. It presents the typical usage of Inter Process Communication (IPC) library for GMIS components communication. It presents the sample screen shots of GMIS viewer. It discusses about the testing environment and GMIS simulator called GMIS-SIM to test the GMIS services. Finally it presents some experiments and results.

5.1 Framework Components Processes

A set of processes are developed to achieve GMIS functionality. The following table 5.1 shows mapping between GMIS components and processes:

GMIS Component	Process name
Discovery Server	Resource Discovery Manager (RDM)
Data Collection Manager	Data Collection Manager (DCM)
Database	Data Handler (DH)
Topology Manager	Topology Information Process (TIP)
Monitor Service	Information Provider Service (IPS)
Resource Selector	Resource Broker Process (RBP)
Communicator Server	Communicator Server Process (CSP)
Agent Software	Communicator Agent (CA), Resource Discovery Agent (RDA), Critical Information Agent (CIA)
Viewer	Web Client
GMIS-GMIS Communication	Peer 2 Peer Communicator Service (P2PCS)

Table 5.1: Mapping of GMIS framework Components to GMIS Processes

To have control over these processes a process called “gmis_controller” process has developed. This process functionality is to start/stop GMIS processes and to monitor the GMIS processes. If any GMIS process is down then gmis_controller will restart that particular process. This process runs along with GMIS processes. Each GMIS process has a common functionality to handle “gmis_controller” messages. The following subsections show the development details of GMIS processes with process flow charts.

5.1.1 Resource Discovery Manager (RDM)

Resource Discovery Manager (RDM) process is developed for achieving Discover Server functionality of GMIS. RDM process receives resource discovery requests from the viewer and resource information from the communicator server. The following figure 5.1 shows the RDM process flow chart.

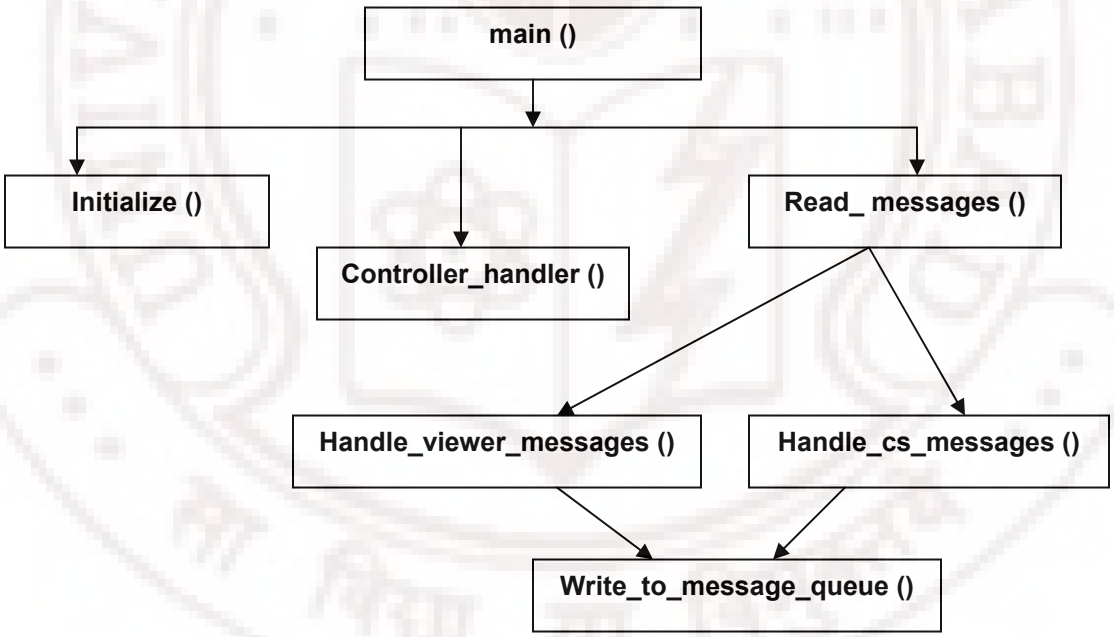

Figure 5.1: Resource Discovery Manager Process flow chart

5.1.2 Data Collection Manager (DCM)

Data Collection Manager process polls the resources and parses the responses. It also handles the events generated by the resources. This process flow chart is shown in the following figure 5.2.

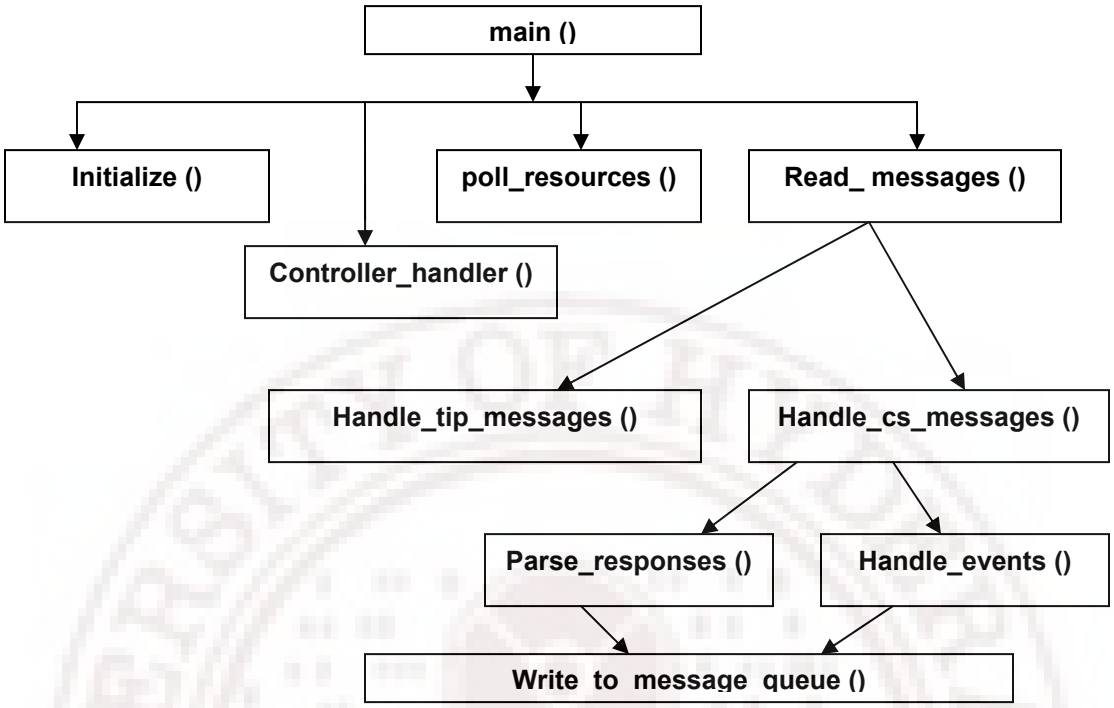

Figure 5.2: Data Collection Manager Process flow chart

5.1.3 Data Handler (DH)

Data Handler process is responsible to insert, update, delete and retrieve resource information from the GMIS database. The following figure 5.3 shows the data handler process flow chart.

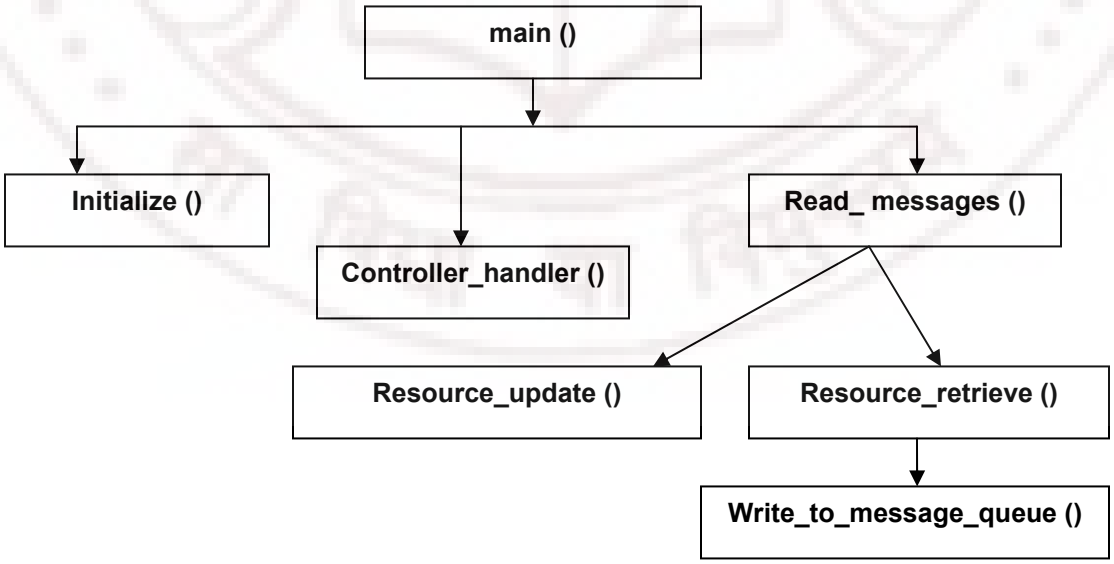

Figure 5.3: Data Handler Process flow chart

5.1.4 Topology Information Process (TIP)

Topology Information Process (TIP) creates, maintains resource object hierarchy. It creates MIB tree in a shared memory to be utilized by other GMIS processes like communicator server. It sends request to data handler to update resource information in database. TIP process flow chart is shown in the following figure 5.4.

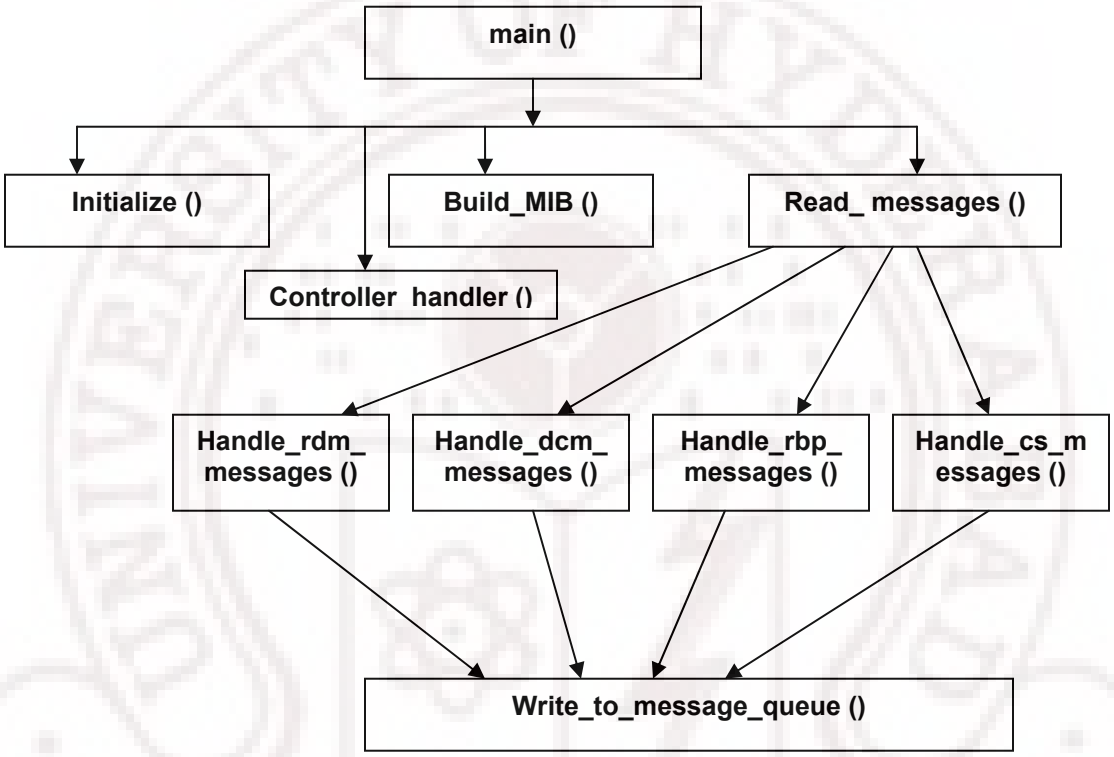

Figure 5.4: Topology Information Process flow chart

5.1.5 Information Provider Service (IPS)

Information Provider Service (IPS) is responsible for providing resource information. It gets the request from viewer and retrieves resource/job information from the database using data handler. IPS sends retrieved information to the viewer. The following figure 5.5 shows the IPS process flow diagram.

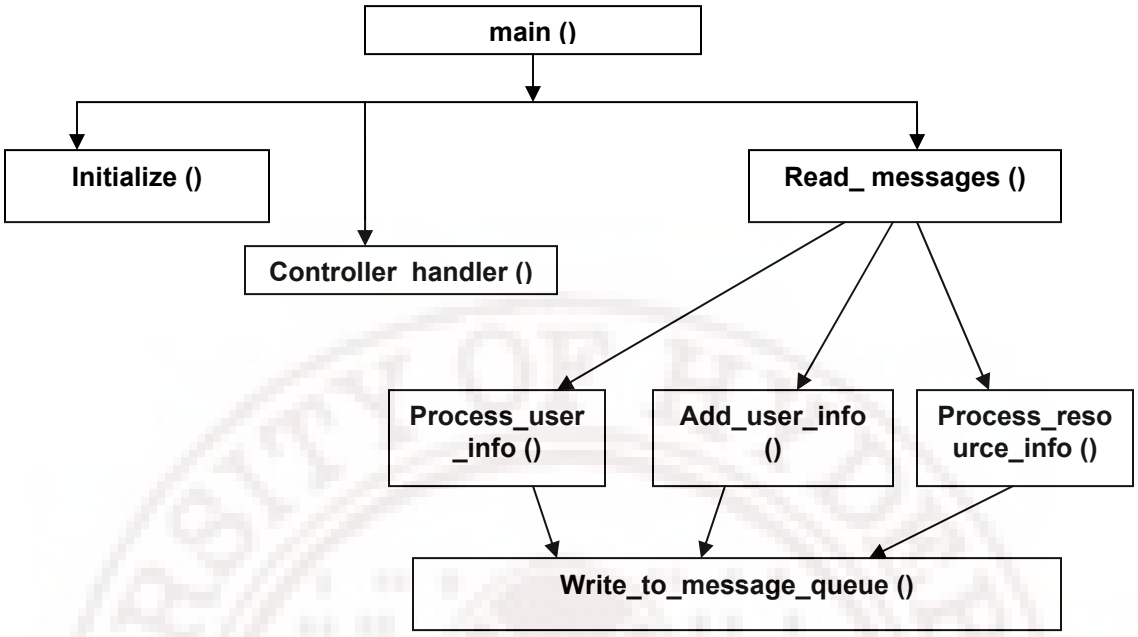

Figure 5.5: Information Provider Service Process flow chart

5.1.6 Resource Broker Process (RBP)

Resource Broker Process (RBP) is responsible for resource selection, generation of activation graphs and execution of jobs. The following figure 5.6 shows the RBP process flow chart.

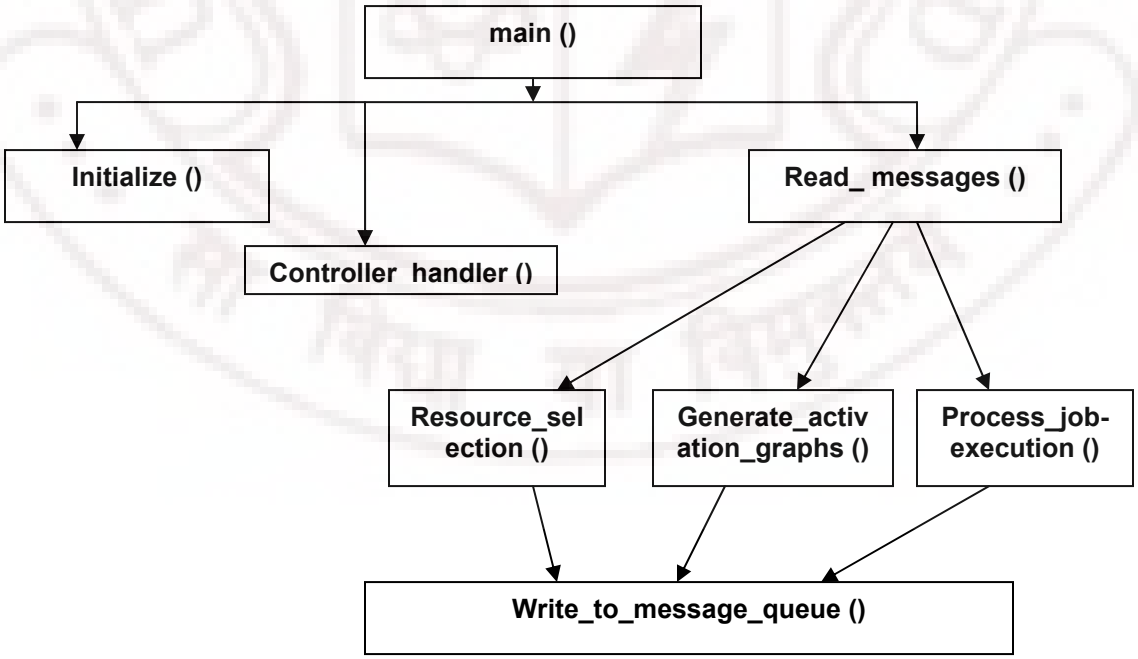

Figure 5.6: Resource Broker Process flow chart

5.1.7 Communicator Server Process (CSP)

Communicator Server Process (CSP) establishes communication between the GMIS components and resources. The following figure 5.7 shows the Communicator Server Process flow chart.

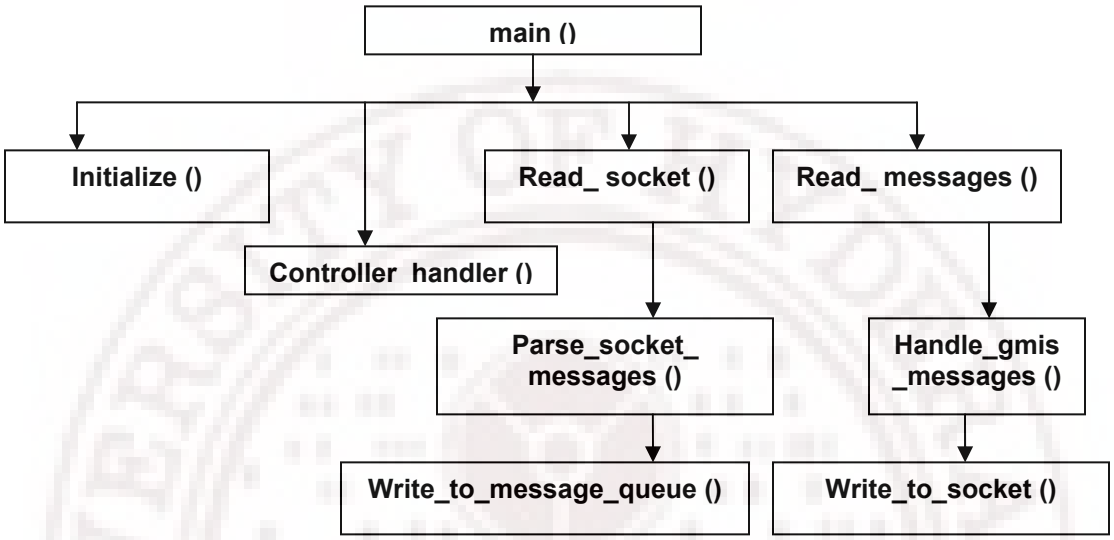

Figure 5.7: Communicator Server Process flow chart

5.1.8 Communicator Agent (CA)

Communicator Agent (CA) on behalf of resource sends updates to GMIS and responds to GMIS Communicator Server requests. The following figure 5.8 shows the communicator agent process flow chart.

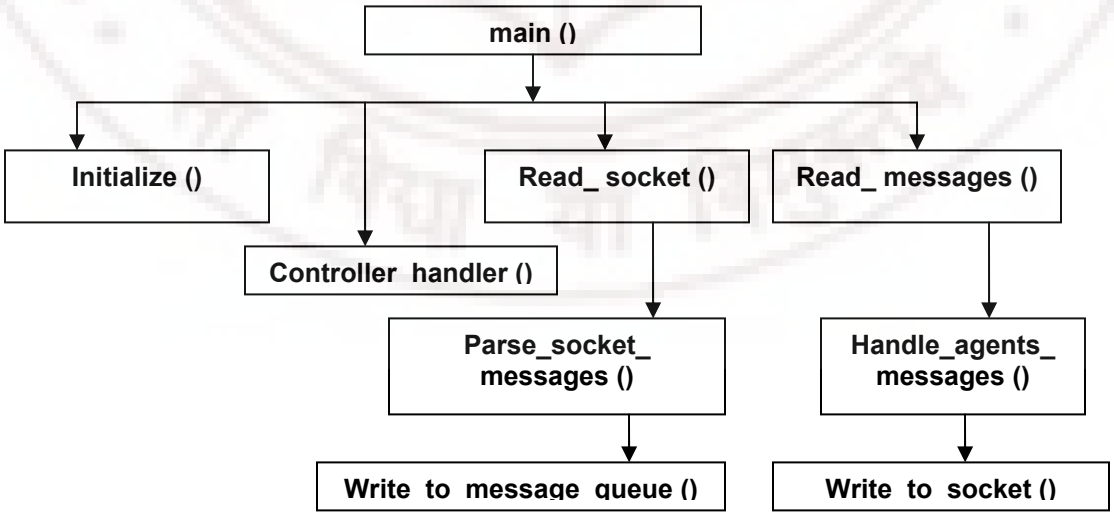

Figure 5.8: Communicator Agent Process flow chart

5.1.9 Resource Discovery Agent (RDA)

Resource Discovery Agent process responds to the requests generated by the communicator agent. This process collects resource information and sends to the communicator agent. Resource discovery agent process flow chart is shown in the following figure 5.9.

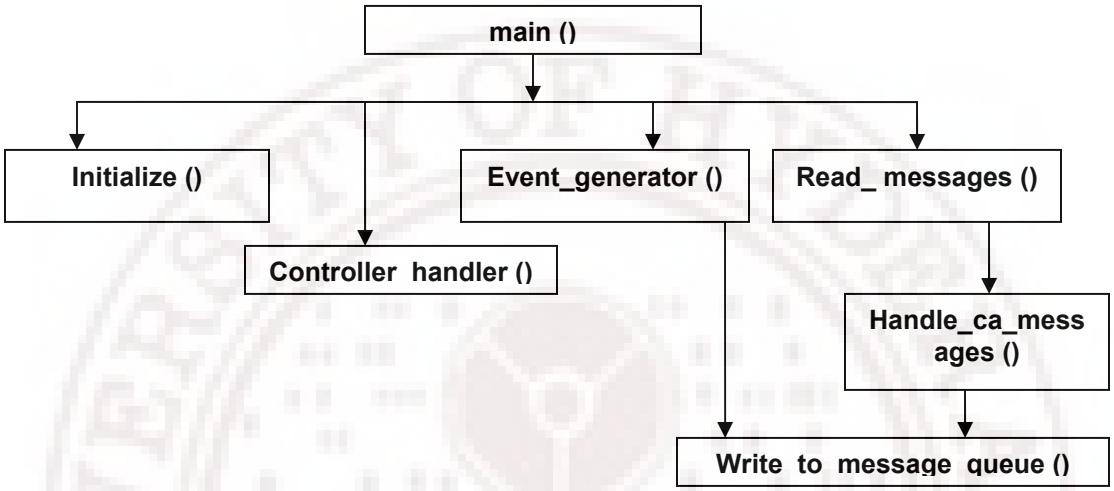

Figure 5.9: Resource Discovery Agent Process flow chart

5.1.10 Critical Information Agent (CIA)

Critical Information Agent (CIA) process periodically checks for the resource information. Whenever there is a critical change in the resource, then an event is generated and communicated to Communicator agent. Communicator agent sends this event to GMIS communicator server. The process flow chart of CIA process is shown in the following figure 5.10.

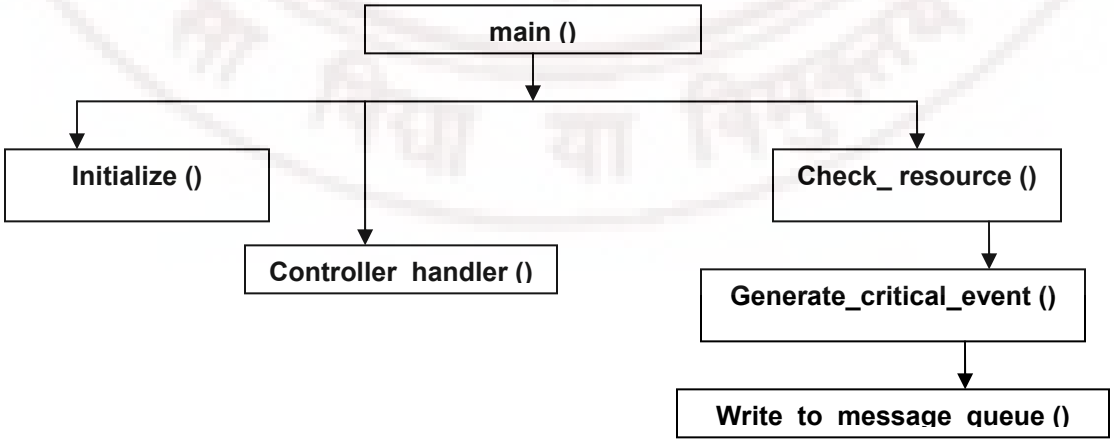

Figure 5.10: Critical Information Agent Process flow chart

5.1.11 Web Client

GMIS viewer is a web application which is implemented using Flex and Servlets. In web application, browser acts as a web client which displays the user interface and the core process logic is executed on the web server. The communication between the client and the server is over Hyper Text Transfer Protocol (HTTP).

To enable the invocation and functioning of the GMIS viewer application, the GMIS application directory on the web server will contain all the Servlets. Flex SWF and html files to support this integrated application. Servlets are java programs which run on the Tomcat web server receives an input request object and responds with a response object. Servlets are responsible for running the logic on the server. SWF and HTML files are required to support GMIS viewer functionality. This SWF file can be played by the Adobe Flash Player. When a grid user invokes GMIS viewer application, main GMIS SWF file is invoked. On the target GMIS which hosts the Tomcat Apache web server, there is an application directory for each web application. The directory structure of Tomcat web server is as follows:

1. /usr/apache-tomcat-5.5.20/webapps/ - This is a parent directory of Apache web server which has subdirectories for each web application.
2. /usr/apache-tomcat-5.5.20/webapps/gmis – This is the application directory specific to the GMIS application. This contains all the executables, configuration files and directory structure to support the GMIS application.
3. /usr/apache-tomcat-5.5.20/webapps/WEB-INF – This directory is a repository of servlets files like SWF files and the library files.
4. /usr/apache-tomcat-5.5.20/webapps/WEB-INF/classes – This directory contains the list of class files which are known as servlets. All GMIS servlets class files are in this directory.
5. /usr/apache-tomcat-5.5.20/webapps/WEB-INF/lib – This lib directory contains the jar files which are referred by the servlets classes.
6. /usr/apache-tomcat-5.5.20/webapps/gmis/css – This directory contains file gmis_style.css which contains the cascaded style sheet configuration settings.

5.1.12 Peer 2 Peer Communicator Service (P2PCS)

Peer 2 Peer Communicator Service (P2PCS) process maintains the connection and communication among multiple GMISs. It requests other GMIS for a service and processes other GMIS requests. The process flow chart of P2PCS is shown in the following figure 5.11.

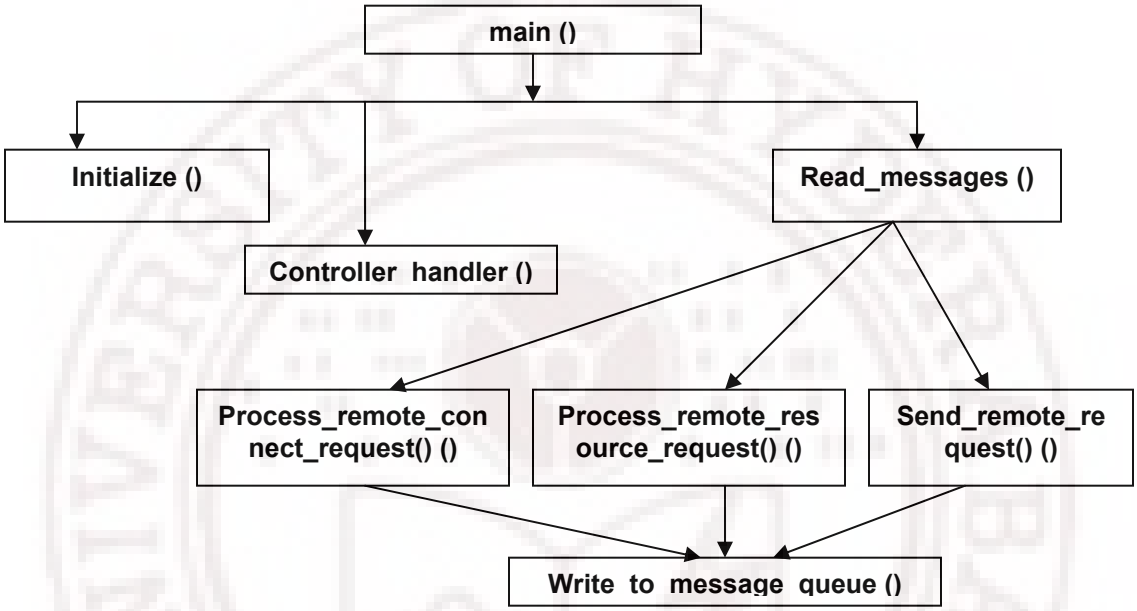

Figure 5.11: Peer 2 Peer Communicator Service Process flow chart

5.2 Communication among GMIS Components

To support communication among GMIS components a set of common Inter Process Communication library functions are developed. The same library is used for communication among the resource agents i.e., communicator agent, resource discovery agent and critical information agent.

The following figure 5.12 shows the typical order in which the IPC services are called. The time at which the un-connected functions are called is irrelevant, as longs as they are called in the right order.

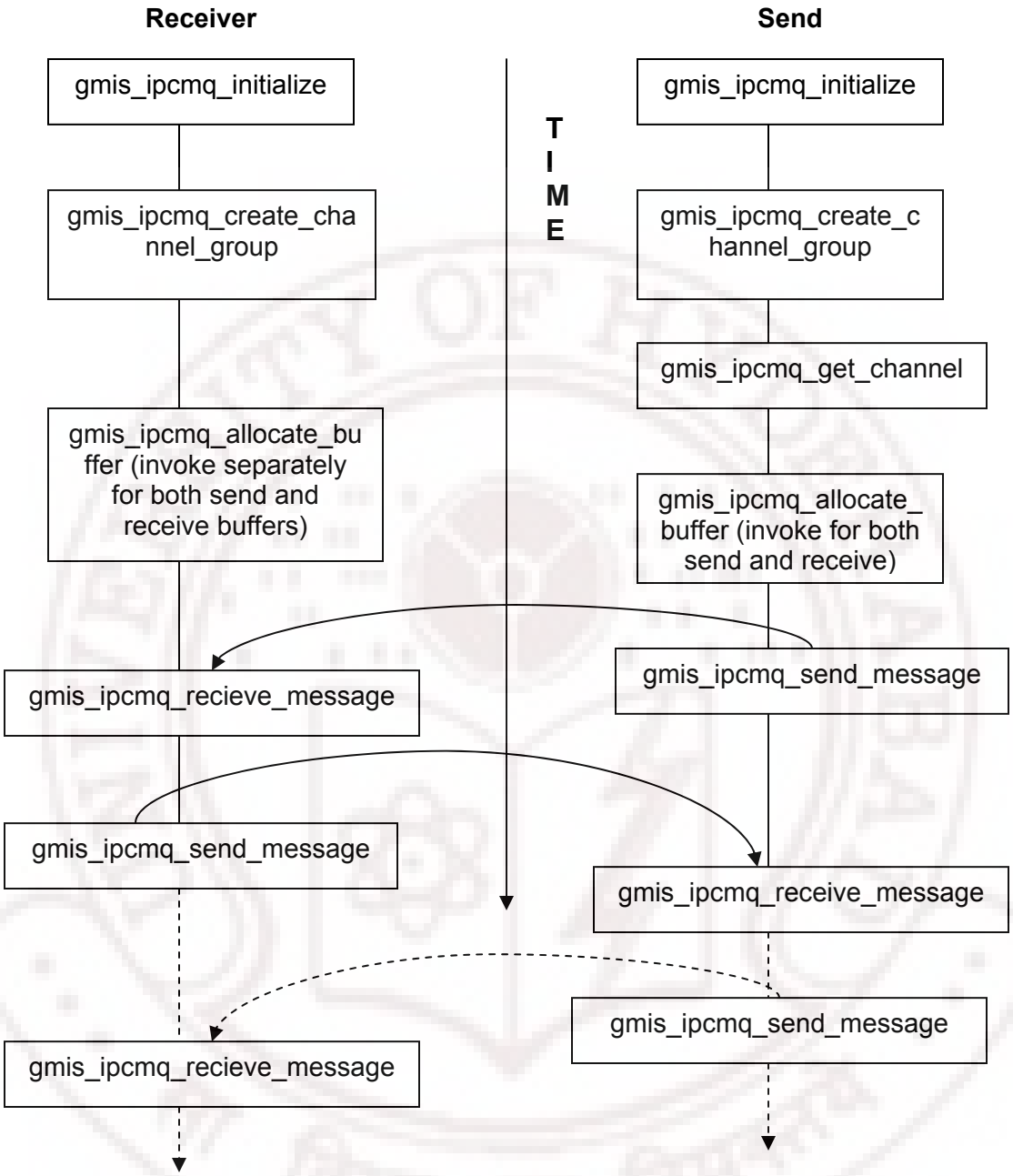

Figure 5.12: IPC Library Typical Usage

5.3 Viewer

Viewer provides a single system image view for all GMIS users. It is a web based application and is mainly responsible for capturing the inputs from the users to discover resources, add resources and to execute jobs. GMIS users can view the status of nodes

or status of jobs submitted and current jobs being executed in the entire GMIS managed nodes. The important user interfaces are discussed in this section.

The login or home page of the GMIS viewer is shown in the following figure 5.13. This home page is accessed through URL as http://<GMIS_IP_ADDRESS>:8080/gmis. For example, http://192.168.100.15:8080/gmis, in this example on 192.168.100.15 machine GMIS processes and Tomcat Server are running. In this page user requires to enter username and password to access GMIS services.

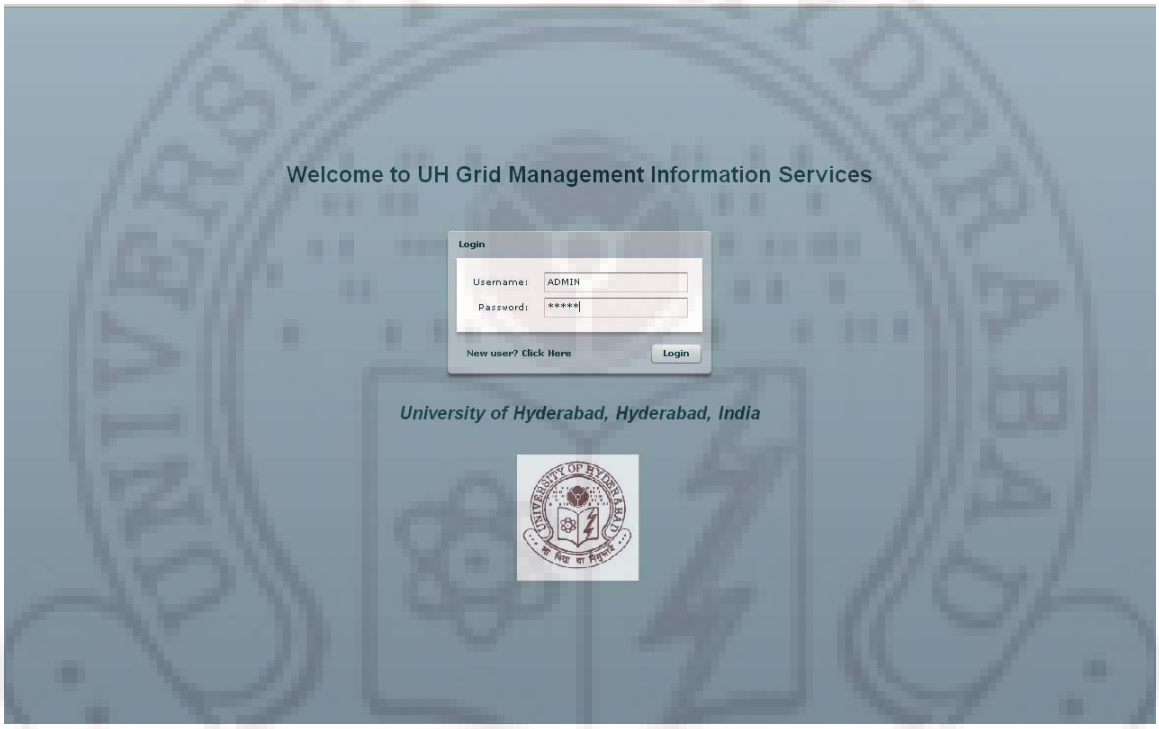

Figure 5.13: Viewer Home page of GMIS Services

If users don't have user credentials then users can register for the services, by clicking "New User" option on this home page as shown in the following figure 5.14. After providing user information, user verification will be done by the GMIS administrator. A pop up message shows a message saying that "Authentication will be done by GMIS administrator and confirmation email will be sent to your e-mail address". GMIS administrator gets a notification about the user registration. GMIS administrator validates the user information. If the registration information is valid information then this user details will be updated into the GMIS database and a confirmation email will sent to the user. After receiving confirmation the user can access GMIS services.

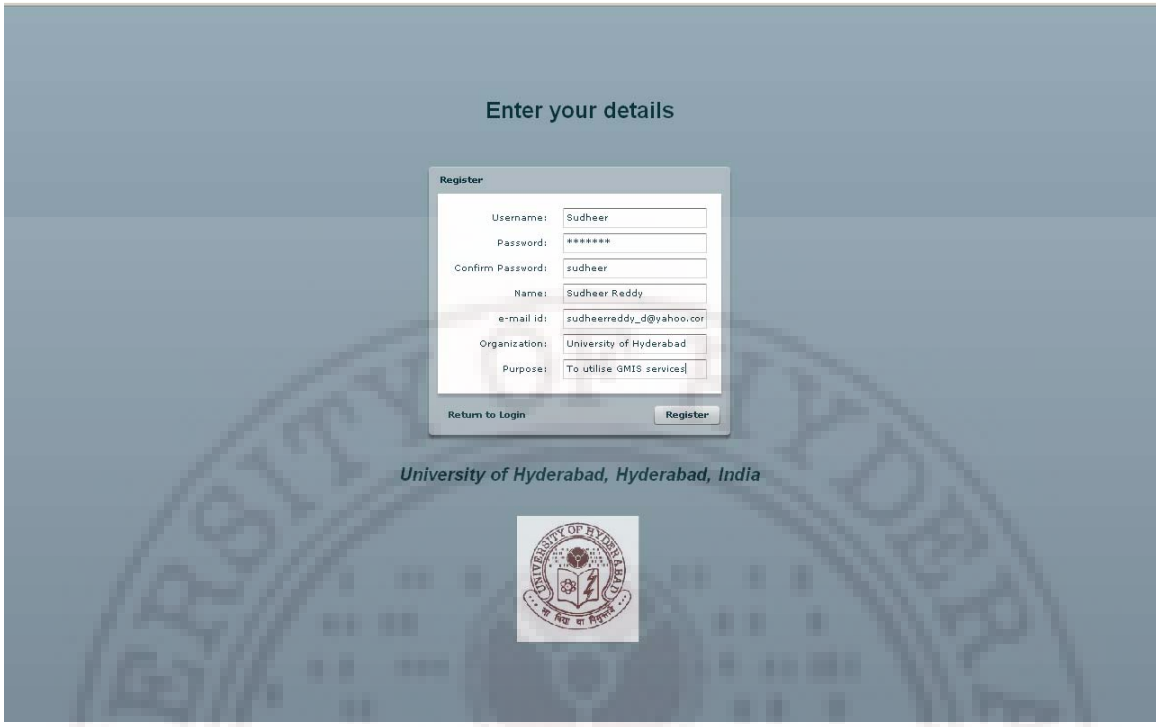

Figure 5.14: New User Registration page on Viewer

After successful login, the GMIS services page will be as shown in the following figure 5.15.

Figure 5.15: GMIS Monitoring View

This page layout designed into two frames. The first frame contains the resources tree hierarchy of resources and jobs. Upon selection it displays the status of resources and jobs.

Second frame contains menu options like File, User Management, Resources, Jobs and Help. File menu will have two options, one is logout and second one is exit. Upon selection of logout, the current home page returns to login page of GMIS. Upon selection of exit the viewer will be closed. User Management menu contains options for changing user password and viewer settings. Resources menu contains various options like Add Node, Add Cluster, Add Grid, Add Service, Discover Node, Discovery Cluster, Discovery Grid, Discovery Service and Resource Availability. These options provided to achieve the user based resource discovery and dynamic grid network discovery services. These options are shown in the figure 5.16.

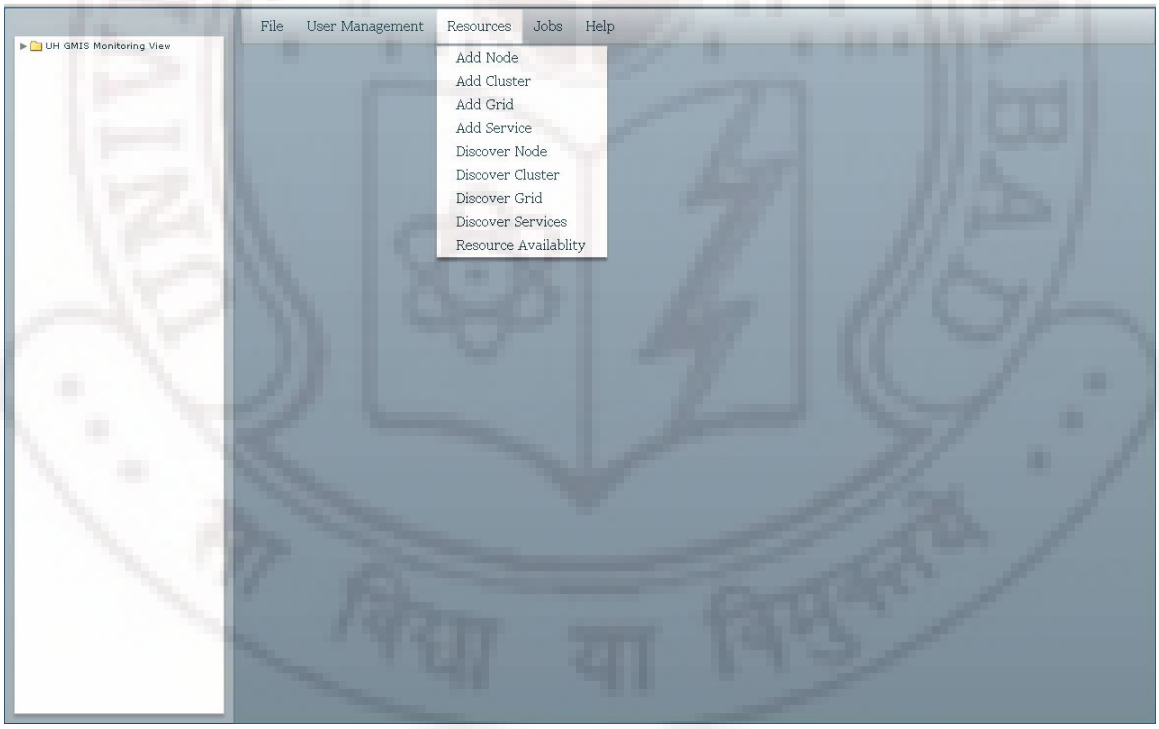

Figure 5.16: Resources Discovery Services View

Jobs file menu contains the options of activation graph creation, execution, job submission and Auto Job execution options as shown in the figure 5.17. The last menu

option Help contains the help contents of GMIS services and how to use the GMIS viewer.

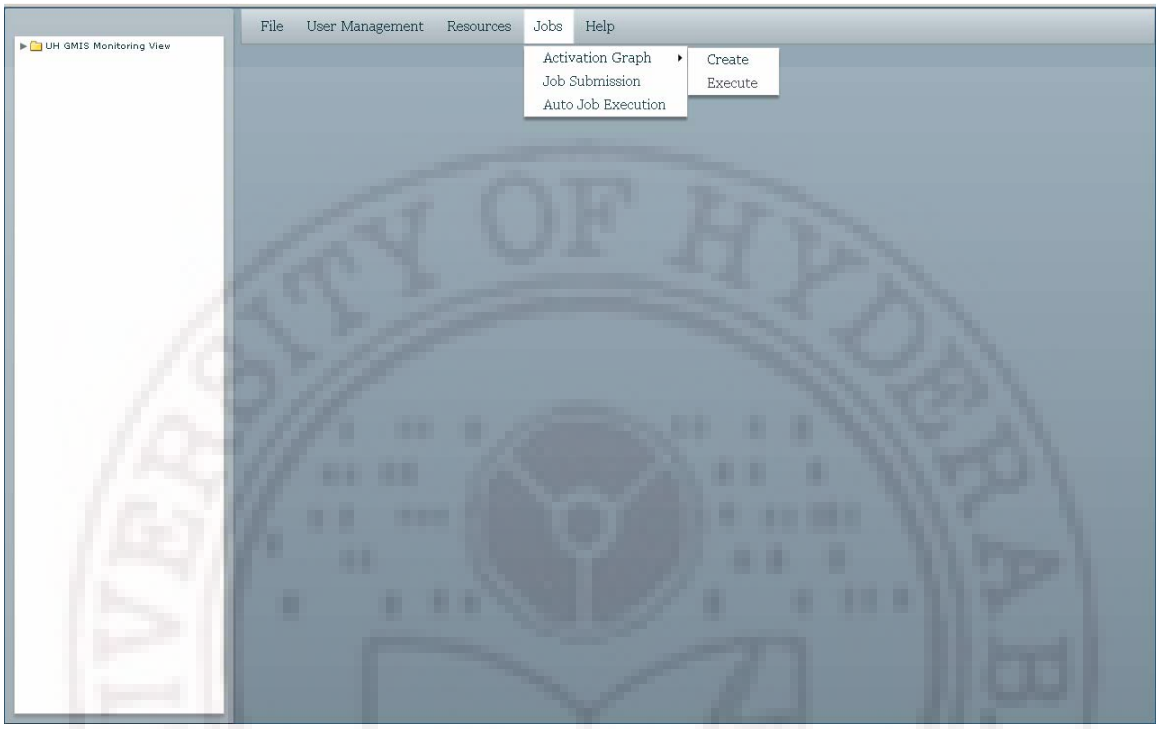

Figure 5.17: Job Execution Services View

5.4 Testing Environment

A simulator called GMIS-SIM has developed to test the GMIS framework. It functions as a set of resources that are configured to interface to the GMIS. The GMIS-SIM is developed for the testing of GMIS services for a variety of reasons. Real applications may run for very long time and it would not be feasible to perform statistically significant number of experiments. Using real resources make it difficult to explore wide variety of resource configurations. The dynamic nature of resources makes it difficult to repeat the experiments. Other reasons include difficulty to create all situations, for example, to create or test on a large cluster or grid, say 100 nodes with heterogeneous computing resources, or test multiple jobs arrived at a single instance is very difficult and some times not feasible. A common practice followed by the researchers is to create a simulator, and simulator is used to create any number of nodes/clusters/grids artificially.

GMIS-SIM simulator is a simple application, which creates a homogeneous and heterogeneous nodes, clusters and grids using response configuration files which consist of sample resource information. There are four types of response configuration files.

1. **Resource Response Configuration file:** This contains basic resource information in the following format.

   Resource_type|FQDN|Host_name|IPaddress|Status|Parent_host_name

   Where,

- Resource_type indicates the type of resource. "1" indicates grid, "2" indicates cluster and "3" indicates node.
- FQDN is Fully Qualified Domain Name.
- Host_name is host name of the resource.
- IP address is physical IP address of the resource. For nodes IP address is mandatory. If any node is acting as a head node for a cluster or grid then that IP address is used for cluster or grid. Other wise it will be marked as "0.0.0.0".
- Status indicates the status of resource. It indicates three values. "1" indicates the resource is available for GMIS users. "2" indicates the resource is busy in performing some job executions. "3" indicates not available to the GMIS users.
- Parent_host_name indicates the parent host of the resource. In case of node, parent host name should be the cluster host name. For cluster parent host name should be a grid host name. For grid it will be the GMIS server host name, which will be mentioned as "0",

   e.g., for a Grid the entry is as

   1|GRID_1@uohyd.gridproj1.ernet.in|GRID_1|0.0.0.0|1|0

   Cluster entry is as

   2|CLUSTER_1_GRID_1@uohyd.gridproj1.ernet.in|CLUSTER_1_GRID_1|0.0.0.0|1|GRID_1

Node entry is as

3|NODE_1_CLUSTER_1_GRID_1@uohyd.gridproj1.ernet.in|NODE_1_CLUSTER_1_GRID_1|192.168.100.16|1|CLUSTER_1_GRID_1

2. **Node Response Configuration File:** This file contains detailed node information in the following format.

FQDN|IP_Address|Operating_System|Processor_family|Number_of_Processors|MEMORY|SWAP|CPU_usage|Disk_space|Dedicated_flag

e.g., for a node an entry is as

NODE_1_CLUSTER_1_GRID_1@uohyd.gridproj1.ernet.in | 192.168.100.16 | Linux8.3|XEON|1|512|1024|28|12000|0

3. **Service Response Configuration File:** This file contains services available in a node in the following format.

FQDN|IP_Address|Service_name|Service_location|Input_file_location|Input_file_IP_Address|Output_file_location|Description

e.g., for a service an entry is as

NODE_1_CLUSTER_1_GRID_1@uohyd.gridproj1.ernet.in|192.168.100.16|Service_1|/home/applications|/usr/applications/input|192.168.100.16|/usr/applications/output| his is an example service

4. **Job Response Configuration File:** This file contains a set of jobs in the following format. Status attribute takes various values. “1” indicates new job submitted. “2” indicates job execution started. “3” indicates job execution completed. “4” indicates job execution failed. Priority of Job takes integer values range from 1 to 9. “1” stands for highest priority where as “9” stands for least priority.

FQDN|Service_name|Status|Input_FQDN|Input_file_location|Input_file_name|Priority_of_job|Output_FQDN|Output_file_location|Output_file_name

e.g., whenever a job is submitted, then this job file gets updated. An entry will look like,

NODE_1_CLUSTER_1_GRID_1@uohyd.gridproj1.ernet.in|Service_1|1|NODE_1_CLUSTER_1_GRID_1@uohyd.gridproj1.ernet.in|/home/applications|input|1|NODE_1_CLUSTER_1_GRID_1@uohyd.gridproj1.ernet.in|/home/applications|Service_1_output.

These configuration files are manually editable. Whenever a status change is required then the configuration files need to be edited manually. For example, after submitting a job, the status will be “job submitted”, this status can be changed to “job execution started”, “job execution completed” or “job execution failed” manually.

For each GMIS there will be one GMIS-SIM process that runs to simulate a grid environment. An environment variable called “GMIS_SIM_ENABLE” is used to differentiate communication between simulator usage and normal network IP addresses usage. If this environment value is set to “1”, then all requests from GMIS communicator server will reach to the GMIS-SIM process. GMIS-SIM process responds to all the requests that come from the GMIS communicator server. GMIS-SIM reads resource information from the configuration files and sends the resource information to the communicator server. For instance if a request comes from a GMIS communicator server to discover a cluster resources then GMIS-SIM looks into the resource configuration files and if there are any resources available with those IP addresses then those resource information will be sent to the GMIS communicator server. In case of job execution job information will get updated in the job configuration file and by default the status is set to “Execution is in progress”. To change the status of a job or resource, the configuration files need to be edited manually, and an event message need to be sent to GMIS communicator server. Other event messages like addition, modification and deletion of resources also can be sent through the GMIS-SIM. For sending messages, GMIS-SIM sends messages to the GMIS communicator server message queue.

5.5 Some Experiments and Results

5.5.1 Cluster Level Dynamic Network Discovery

To discover resources at cluster level a range of IP addresses need to be given on the viewer. For this select “Resources -> Discovery Cluster” option on menu bar. A viewer appears to enter “Start IP address” and “End IP address”. After entering IP addresses click on “Discover” button. A pop up message comes to the user saying that “Select the resource view to see the discovered resource information”. If no resource information is available with the given IP addresses range, an alert message will display on viewer as shown in the following figure 5.18.

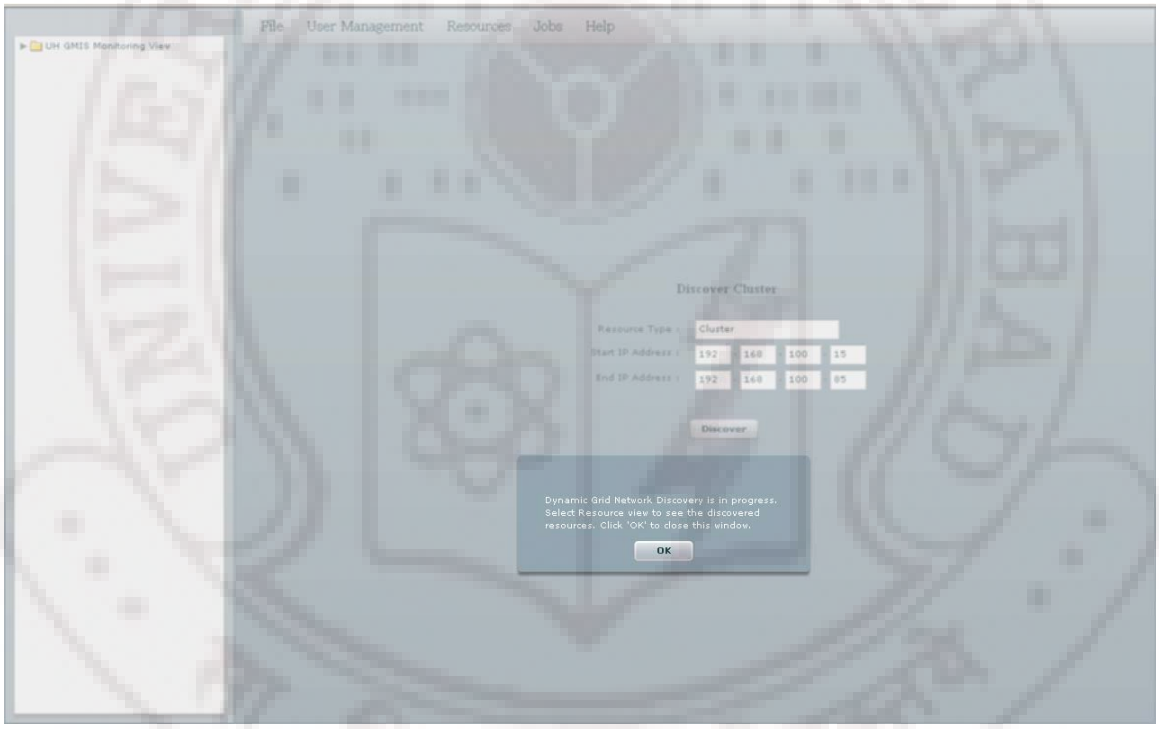

Figure 5.18: Cluster Discovery View

5.5.2 Monitoring of Resources

To check the status of resources, users have to select the resource view option on left side hierarchical tree. Selection of a cluster shows the nodes and their status on right side frame. Nodes are represented as colored icons. A green icon represents that node is available for providing service. A red icons represents that resource node is busy in performing some job execution. Resource Monitor service decides which node is

available and which is not, based on the node's CPU and Memory usage. Any of these two values exceeds to a certain threshold values then icon will be represented as a red icon. Otherwise it will be shown as Green color. These threshold values are configurable. Currently, two resource attributes are considered. This can be extended to other parameters like how many jobs it is executing, how much swap is available or instead of values, usage percentages. A blue icon represents that node information is not available due to network problem or some other problem. 48 node cluster status information on viewer is shown in the following figure 5.19.

Figure 5.19: Node Status View

By clicking on any node, complete node configuration information window will be displayed as shown in the following figure 5.20. This information includes node host name, parent cluster name, parent grid name, FQDN, IP address, operating system version, processor family, number of processors, RAM, SWAP, CPU Idle, Disk space available and whether it is a dedicated node or not.

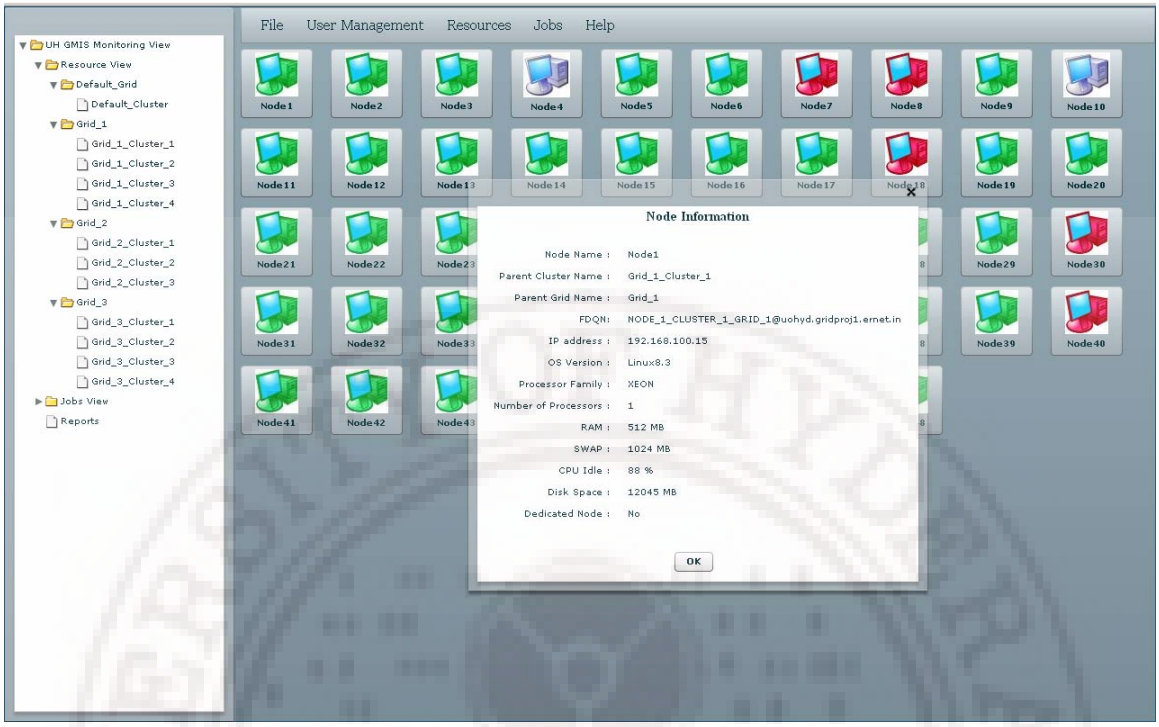

Figure 5.20: Detailed Resource Information of a Node

5.5.3 Job submission

GMIS users can submit the jobs using interface provided on viewer. Select "Jobs-> Job Submission" option to submit the jobs. Enter the Job submission information like FQDN where job need to be executed, Input file location, output file location, priority of job. Upon filling all the information related to job submission, click submit button on the viewer. The information entered by the GMIS user is validated and if the information is valid information, a pop up message shows up saying that "Go to the Jobs view to see the job status" as shown in the following figure 5.21.

There is another option called "Auto Job Execution" provided on the viewer. Here user mentions node requirements, service location, service input file location and output file location desired. GMIS service resource selector selects a best suitable node for the user requirements and submits the job execution on that node using GMIS communicator server. Resource communicator agent gets the service and input files from the user mentioned locations and start execution of the job. After completion of job execution output file sends to the user mentioned location. Other job submission option "Activation Graph" creation and execution is discussed in the next chapter.

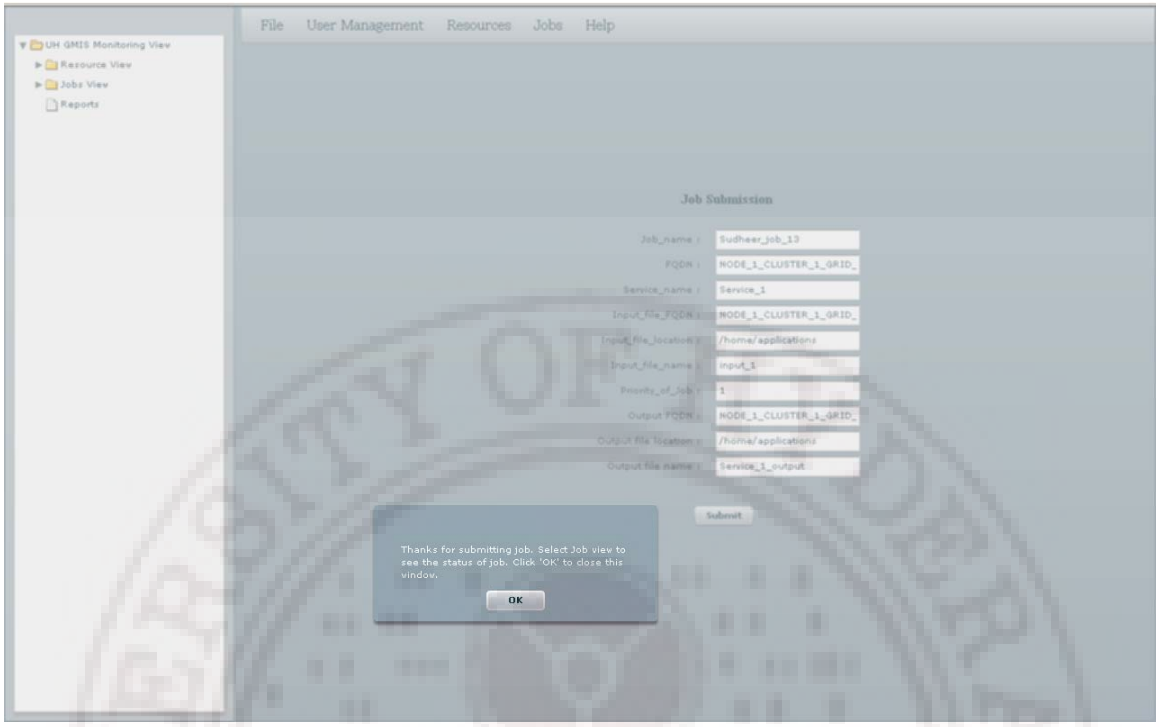

Figure 5.21: Job Submission on GMIS Viewer

Select the “Jobs view” and click on the job icon to see the status. The status of job execution will be in the popup message as shown in the following figure 5.22.

Figure 5.22: Job Execution Status

5.5.4 Multiple GMISs' view

Multiple GMISs can be connected for achieving GMIS-GMIS communication. Multiple GMISs resource information will be shown on the left side frame with the under different GMIS names. For instance, if "CS GMIS" is another GMIS which maintains different set of resources, connected to the UH GMIS, then the left side resource monitoring view of UH GMIS shows the CS GMIS information also as shown in the following figure 5.23. At the same time "CS GMIS" monitoring view contains only "CS GMIS" managed resources. User of "UH GMIS" can view the resource status of "CS GMIS" resources and can submit the jobs on "CS GMIS" managed resources.

Figure 5.23: Multiple GMISs Connected View

5.6 Summary

This chapter discussed about the GMIS processes development and interface modeling using process flow charts. It also discussed about the IPC library services typical usage. It shows the sample screen shots of GMIS viewer like login screen, resources discovery services option, monitoring resources and job submission options. It

discussed about the GMIS-SIM which is a simulator to test GMIS services. Finally it has shown the multiple GMISs view on a single GMIS viewer.

Next chapter will discuss about the Multidisciplinary Design Optimization (MDO) and UH MDO framework which is chosen as a case study for the GMIS. Virtual cluster formation is another application of GMIS is also discussed in the next chapter.

CHAPTER 6

Multidisciplinary Design Optimization (MDO) – A Case Study

Multi-disciplinary Design Optimization (MDO) application is being looked at as a case study for the GMIS framework. This MDO application requires integration of disciplinary analysis modules. These disciplinary analyses are typically distributed over Internet. Integrating such analysis modules is usually quite complex and addressed in UH MDO framework [Saran 2005, Nilakanta 2006]. This framework would require a platform which provides distributed computing environment over a WAN. Globus was used as a distributed environment for the framework. How ever GMIS can be used as a distributed environment for the UH MDO framework. GMIS provides all services required for UH MDO framework.

This chapter gives an introduction to Multidisciplinary Design Optimization, MDO requirements and then discusses about some of the frameworks available. It discusses about the UH MDO framework architecture and how GMIS components support UH MDO framework functionality. It shows how activation graphs can be created in GMIS viewer with an example. Finally another application called “Virtual Cluster” formation using GMIS is discussed.

6.1 Introduction to Multidisciplinary Design Optimization (MDO)

Multi-Disciplinary Design Optimization (MDO) is gaining popularity and is increasingly being used to design complex engineering systems. This requires integration of disciplinary analysis modules. Disciplinary analysis typically evolves without any centralized coordination and is often distributed over network.

System level analysis can be performed using MDO framework. Such a system level analysis includes each sub-system analysis and various sub-system interactions. For the purpose of performing system analysis various analysis modules are to execute in some sequence. For large number of modules it becomes humanly difficult to iterate, the generated input files and parsing output files to extract useful information. Therefore

a software integration system which helps user to tie in all disciplinary analysis modules is required.

The UH MDO framework [Saran 2005, Nilkanta 2006] supports process execution in distributed heterogeneous computers. Globus was used as a distributed environment for the framework when it is developed. The frame work provides intuitive GUI and is extensible. It handles large size problems, by dividing it into small modules, automates the execution and movement of data between different modules over grid. The framework provides the database management features for storing and provides data exchange between analysis modules.

Multi-disciplinary analysis process contains:

1. Build the data dictionary.
2. Collect all the analysis modules
    i. Build wrappers for all analysis modules
    ii. Register in the framework
3. Design the system problem
    i. Specify data dependency
    ii. Providing execution sequence
4. Specify optimization problem
5. Solve the problem
6. Visualize results

6.2 MDO Requirements

The previous MDO framework designs are based on distributed architectures like Common Object Request Broker Architecture (CORBA). CORBA based programs are interoperable irrespective of the type of computer, network, operating system or programming language. In addition to remote execution, CORBA provides services for naming, transaction monitoring, error handling, etc. This makes CORBA ideally suited for distributed computing. Most of the frameworks that are developed so far and are being developed are using CORBA for network communication.

However, all these activities are concentrated to make frameworks work on Local Area Networks (LANs). To make these activities over a Wide Area Network (WAN) require a distributed architecture that is suitable for the requirement. Grid computing is a technology where distributed computing over WAN is possible. So grid computing has used to build a MDO framework called UH MDO framework, so that it will be suitable for computing over WAN.

The purpose of UH MDO framework is to provide support for multidisciplinary design optimization application development and execution. This section lists a set of requirements for an ideal framework. The requirements are presented from the following points of view:

- Architectural design
- Problem formulation and construction
- Problem execution
- Information access

Architectural Design:

- Provide a intuitive GUI
- Adapt object-oriented principles
- Extensible and support for developing interfaces for adding new programs
- No unreasonable amount of overheads
- Handle large size of problems
- Support collaborative design

Problem Formulation and Construction:

- Configure complex branching and iterative problem formulations easily.
- Easily reconfigure existing problems
- Support multiple optimization methods including multilevel schemes.
- Provide debugging support for multiple processes executing across computers on the network.

Problem Execution:

- Automate execution of processes and movement of data
- Execute multiple processes in parallel
- Support execution distributed across network of heterogeneous computers
- User interaction (steering) during design cycle
- Operate in batch mode

Information Access:

- Provide database management features
- Capability to visualize intermediate and final results of the analysis
- Monitoring capability for viewing the status of an execution and system status
- Mechanism for fault tolerance

6.3 Related work

6.3.1 Frameworks at Research Labs

There are various MDO frameworks developed at research labs. Some of the important frameworks are discussed in the following subsections.

6.3.1.1 FIDO

FIDO stands for Framework for Interdisciplinary Design Optimization was developed by NASA Langley Research center [Townsend, 1993]

Architectural Design: The FIDO architecture is modular. The framework is organized into distributed computational and service modules, which communicate through a communications library. There is a computational module for each discipline contributing to the application. The service modules, such as the GUI, Executive (control), Data Manager, Setup, and Spy, are intended to be application independent.

The communications library contains functions designed to facilitate communications among a general system of computer codes executed in a heterogeneous, distributed network of computers. This library allows FIDO to be programmed without directly accessing the underlying, message-passing primitives and minimizes the impact on FIDO due to any changes in them. Currently, the PVM (Parallel Virtual Machine) primitives from the Oak Ridge National Laboratory are used.

The GUI is limited to displaying the status of the execution. The Spy tool promotes collaboration among researchers by allowing access to Spy from multiple remote computers. Although object-oriented principles were not applied to FIDO, there was a strong emphasis on producing a modular system.

Problem Formulation and Construction: A major limitation of FIDO is that it lacks support for building and reconfiguring MDO problem formulations at a higher level of abstraction than coding in the currently available programming languages, such as FORTRAN and C. Some discipline codes used in FIDO are decades old and originally contained deeply embedded *print* and *stop* statements. These codes were modified to behave as library subroutines and are invoked from the appropriate discipline driver. As a result, the discipline driver and the associated discipline codes are linked into one executable program. Overall, this is not a desirable approach because it involves extra work, duplicates maintenance tasks, and does not promote code reuse.

Coordination of discipline analyses is provided by a problem-dependent Master module. The code for this module must be rewritten for each specific MDO application.

Problem Execution: The user designs the FIDO 'master' module so that the optimization and analysis processes are invoked and synchronized appropriately. The user provides the synchronization logic within the discipline drivers in the form of calls to the communications libraries send and receive routines.

The FIDO 'setup' module allows the user to choose the system configuration, initial conditions and constraints of the optimization process from a range of previously defined possibilities. These are contained in configuration files that define the data in standardized formats.

All major data elements (individual items or file pointers) that are shared between modules are passed to, stored in, and retrieved from the central Data Manager. Using file pointers, data files are passed directly on request from the generating computer to the requesting computer with in a LAN. The discipline drivers and their corresponding analyses are assigned to execute in parallel on different computers defined to be part of the PVM network.

From the beginning of its development, FIDO was designed to allow some interactivity during in design cycle. This feature is accomplished using the Spy tool, which allows the user to steer the process while the application is executing. By means of the Spy tool, the user may change current values of design variables, constraints, and parameters. On the other hand, FIDO lacks convenient way of setting up multiple problems that can execute one after the other. In particular, changing the initial conditions of a problem requires manually editing the input and configuration files.

Information Access: The FIDO Data Manager allows storage and retrieval of data during problem execution, so that no additional coding is required for new problems. The user must define the data to be handled prior to execution.

The FIDO Spy module allows the user to access and plot data from previous design cycles. The accessible data includes information on the cycle status and selected scalar and array data from each cycle. The data can be displayed as text or graphics. However, the database is not persistent, so FIDO must be running for data to be accessed.

The FIDO GUI displays the state of the problem execution at all times. The GUI displays the problem formulation and uses color to indicate those processes that are starting up, executing, inactive, or shutting down. Although FIDO allows restart from a completed optimization cycle, it provides no other fault-tolerance capability. However, a restart requires some data file preparation.

6.3.1.2 DAKOTA

DAKOTA which stands for Design Analysis Kit for OpTimizAtion, was developed by Sandia National Laboratories [DAKOTA]. The DAKOTA design is based on object-

oriented principles and is implemented with the C++ language. The definition of generic interfaces between optimization methods and analysis codes hides the specifics of each. Use of these interfaces and object oriented language features promotes the “plug and play” capability.

Problem Formulation and Construction: To define the MDO application, the user must create a file that specifies information about interfaces, variables, responses, strategies, and methods. In DAKOTA, “strategies” manage methods and “interfaces” provide access to the discipline code, which map the variables to the responds.

Several types of interfaces are defined in DAKOTA, the primary being the application interface. The application interface allows discipline codes to be accessed through either system calls or direct function calls. The direct function call interface requires converting main programs to function calls and linking the functions into the DAKOTA executable. The system call interface allows access to external programs; communication between the external program and DAKOTA is accomplished via files.

The interface section of the specification file must include the name of the analysis (or analysis driver), and if required, the names for the input and output filters (i.e., pre and post processors). Note that only one analysis driver may be specified; however, an entire MDO application, developed outside of DAKOTA, along with an input and an output filter. The input filter must use the design parameters list provided by DAKOTA to prepare the input for the analysis driver. Also, the output filter must retrieve data from the analysis driver and prepare the response and sensitivity data in the format required for use by DAKOTA.

A variety of optimization methods are provided, including Non-Linear Programming (NLP) and genetic algorithms. The DAKOTA strategies manage multiple methods, disciplines, and approximations. The strategies include single, multilevel hybrid, and sequential approximate optimization. The single strategy allows a single method to be used with a single discipline. The multilevel hybrid strategy allows multiple methods to be used in succession with a discipline. This strategy uses the best solution from one method as the starting point for the next method. The switching criteria used can either be based on an individual method’s convergence criteria or an adaptive

technique that employs method performance metrics. The sequential approximate optimization strategy uses both a discipline and an approximation of the discipline. The approximation model is optimized, and the discipline model is evaluated at the approximate optimal solution. These results are used to update the approximation.

Problem Execution: Both the execution of the analysis driver and input/output filters and the transfer of data between these and the optimization methods are automated by DAKOTA. Distributed computing is supported using Message Passing Interface (MPI) message passing on workstation clusters and on massively parallel supercomputers. The asynchronous function evaluation command option allows concurrent analysis calculations and is available with both system call and direct function interfaces. This feature can be used when calculating derivatives using finite differences or when using the parallel algorithms provided in DAKOTA.

Information Access: There is an option for the user to specify creation of a restart log. Also, several options are available for handling application failure recovery.

6.3.1.3 CASDE

CASDE is Center for Aerospace Design and Engineering department in IIT Bombay [Amitay, 2003]. They have developed a software integration system, which helps user tie in all the disciplinary analysis software, and automate the execution of those modules in the sequence provided by the user. The design of their framework is based on object oriented principles and was implemented in Python.

Architectural Design: The framework developed by CASDE contains the GUI, which provides features for constructing the analysis module sequence, design problem and for analyzing the results. The GUI is written using the wxPython language. The user can include any analysis code as part of an MDO application as long as the design input and output can be identified in the input/output files. Database engine used in this framework is MySQL.

The concept of Data Server is introduced which serves the purpose of exchanging data between different analysis modules in the distributed environment. Data can be specified as scalars, vectors, multidimensional arrays or complex

structures. CORBA (Common Object Request Broker Architecture) is used as the distributed computing feature in the framework.

Problem Formulation and Construction: The user employs the framework GUI to define the analysis sequence. Through the GUI, the user identifies the data flow and the execution sequence analyses modules along with their corresponding input and output files. In addition, the files associated with the design data input and output are identified.

MDO framework uses multithreading approach to perform the parallel execution that is identified in the execution sequence. Corresponding to each analysis module, one thread of execution is started. Directed edges in the sequence diagram denote the events, which each thread waits on.

Problem Execution: The framework automates the execution of the various disciplinary modules included in the analysis, manages the input and output data, and adjusts the design variables. The processes defined in the analysis are executed sequentially. For distributed computing support, CORBA features are used which could provide a remote process invocation methods.

Information Access: The framework's data server allows storage and retrieval of data during problem execution and is designed in such a way that no additional coding is required for new problems. The user must define the data to be handled prior to execution. The framework GUI is helpful in viewing the results of all the problems which were already executed and saved.

6.3.2 Frameworks at Commercial Organizations

There are some commercial organizations developed MDO frameworks. Some of such commercial frameworks are discussed in the following sub sections.

6.3.2.1 iSIGHT

The iSIGHT [iSIGHT, 1998] framework is a generic shell environment for supporting multidisciplinary optimization developed by Engineous software. A key feature of iSIGHT is the ability to combine numeric, exploratory, and heuristic methods during an optimization.

Architectural Design: The iSIGHT environment consists of several modules including an interpreter, toolkits, and GUIs. The Tcl language is the interpreter that provides the "glue" for integrating various processes. GUI services are provided for connecting processes, defining the optimization plan, and monitoring results. GUI services are provided for wrapping discipline codes. In addition, the user may integrate additional optimization techniques into iSIGHT. However, the iSIGHT Application Programming Interface (API) must be used to create the appropriate interface between the optimizer and the framework. In addition, a Tcl command must be created for the optimization technique.

Problem Formulation and Construction: The iSIGHT framework provides a GUI, in which icons represent modules, and the Multidisciplinary Optimization Language (MDOL) for constructing MDO problems. Use of the GUI to define the problem generates the appropriate MDOL file, referred to as a description file. MDOL has a block structure style and English like language constructs.

The GUI provides the user with building blocks representing discipline codes and calculation blocks that may be needed in addition to the discipline codes. The user may define the input, output, and execution invocation of the discipline code blocks, as well as the arithmetic expressions for the calculations blocks. However, within the GUI, the user is limited to define a sequential order for the disciplines and calculations.

The iSIGHT framework allows users to construct MDO applications using existing discipline codes without modifications by interactively generating a code wrapper. Parsing utilities create the appropriate input files and extract the appropriate data from discipline output files. Using the GUI and the input and output file templates for a discipline code, the user can generate the appropriate file parsing commands. As a result, the user is able to integrate legacy and proprietary codes into the MDO problem.

Problem Execution: The iSIGHT framework automates the execution of the various discipline codes and calculation blocks, the handling of the data, and the adjustment of design variables during optimization. The computational processes defined in the problem formulation are executed sequentially, because iSIGHT provides no support for parallelism. There is very limited support for distributed computation. For example, a

discipline code may initiate a remote process; however, all description codes for a problem must reside in the same directory on a single computer. Interactive features in iSIGHT provide the capability to pause the execution and continue it later. The user can stop the execution to modify the optimization methods, design variables, constraints, and objective function. Upon restart, the execution resumes from the best design point of the previous optimization. Discipline codes can be switched only if the appropriate logic is present in the description file.

Information Access: The iSIGHT framework lacks a database capability other than the data management toolkit that keeps a history of the design states. Therefore, data sharing among several discipline codes has to be accomplished by writing and parsing files. A monitoring capability is provided that can be applied at any time during execution. Input and output values can be monitored in tabular or graphical form. In addition, the user can review the data from a completed optimization and can restart the optimization process from a design point previously computed. The GUI also indicates which process in a task is running by changing the appearance of the module icon.

6.3.2.2 LMS Optimus

LMS Optimus [LMS, 1997] is a framework for multidisciplinary optimization, developed by LMS International Inc. LMS Optimus, allows a user to setup a problem, select a method to be used with the problem, and analyze the results. Features of LMS Optimus provide nonlinear programming (NLP), DOE and RSM techniques for optimization.

Architectural Design: The Optimus Kernel module contains the GUI, which provides features for constructing the analysis sequence and design problem and for analyzing the results. The GUI is written using the C++ language and Motif. The user can include any analysis code as part of an MDO application as long as the design input and output can be identified in the input/output files.

Two optimization modules are provided in LMS Optimus: the DOE/RSM module and the NLP module. A recently available feature allows the integration of an external optimizer. All that is required for integration of an optimizer is that it writes the adjusted design variables to a file and reads the analysis results from a file. Due to internal array

sizes, the maximum number of design variables allowed is 50; the maximum number of design outputs allowed is 200.

Problem Formulation and Construction: The user employs the LMS Optimus GUI to define the analysis sequence. Through the GUI, the user identifies the analyses and their corresponding input and output files. In addition, the files associated with the design data input and output are identified. Command file will be created based on the actions taken through the GUI. The command file contains sections for defining design inputs, design outputs, discipline input and output file parsing commands, analysis sequencing, and optimization method selection. The sequencing commands include if/then/else and for control statements. The GUI generates only a subset of commands that can be included in the command file; the user can edit the command file to include additional commands.

The LMS Optimus user can include legacy and proprietary codes in the analysis sequence without making any modifications. The GUI can be used to identify the design data in the input files; before each analysis is executed. The framework will include appropriate input automatically from the input files. Similarly, the user identifies, through the GUI, the output which needs to be extracted from the output file; after the analysis completes, the data is automatically extracted.

Problem Execution: The Optimus Kernel automates the execution of the various discipline codes included in the analysis, manages the input and output data, and adjusts the design variables. The processes defined in the analysis are executed sequentially. For distributed computing support, the command language includes a command for executing a remote process.

Information Access: The results from a completed NLP or DOE method can be loaded by the GUI and post processed. The results of an optimization can be displayed in a tabular format. Several options exist for visually analyzing as RSM.

6.4 UH MDO Framework Over Grid

6.4.1 Architecture

The architecture of the MDO Framework is shown in following figure 6.1 [Saran 2005, Nilakanta 2006].

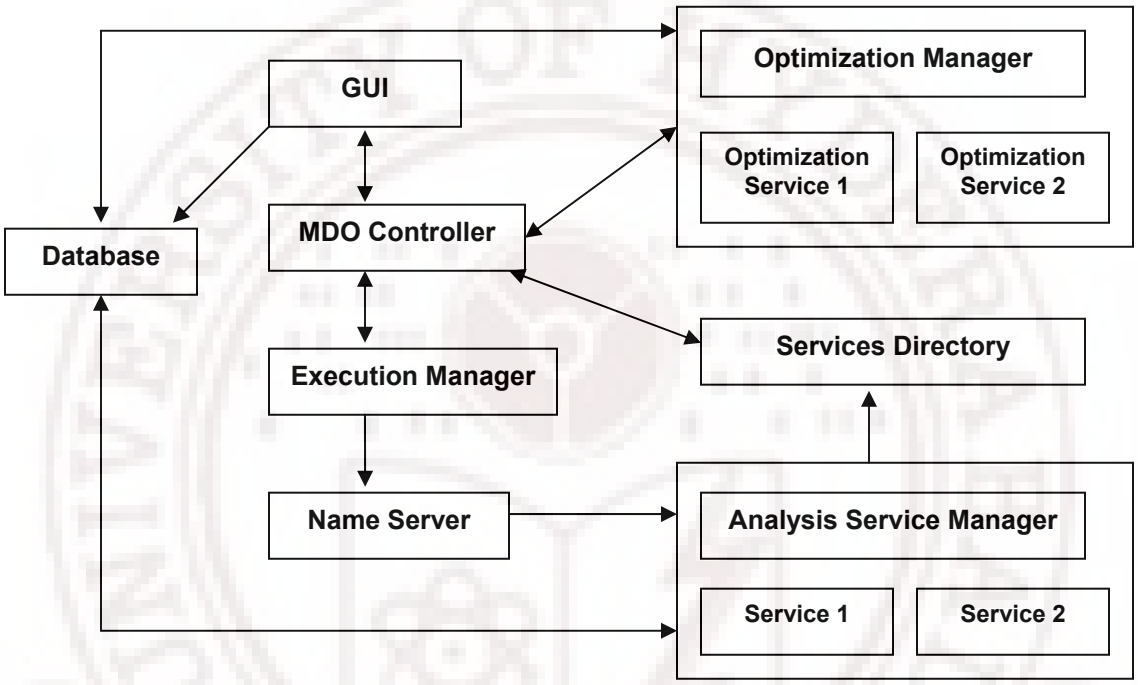

Figure 6.1: UH MDO Framework Architecture [Nilakanta, 2006]

UH MDO framework has the following main components:

1. **Graphical User Interface (GUI) -** is the interface for interaction between the user and framework.

2. **MDO Controller -** is the interface between optimizer, execution manager and service directory. This provides the user what services are available, how to execute the activation graph i.e., either sequential or parallel execution of analysis services.

3. **Database -** is Relational Database Management System (RDBMS). PostgreSQL is used as a database engine. Database stores the data dictionary information about various analysis software and wrappers, analysis and optimization variables and their solutions.

4. **Optimization manager -** manages the execution of various optimizers. There can be a multiple optimizers available in the framework.

5. **Optimization Services -** are grid services in the grid based environment. Available optimizer services can be integrated with the framework.

6. **Service Directory -** contains the list of the services available on the grid and their corresponding FQDN (Fully Qualified Domain Name) to tell where the service is located. The nodes on the grid must register with this service directory to know what services that grid can contribute to the framework.

7. **Execution Manager -** does the job of distributed execution of analysis modules. Given multiple analysis modules located on the network and a sequence of execution of these analysis services, the execution manager contact corresponding analysis managers to execute required analysis. If possible execution manager executes some of the analysis services in parallel.

8. **Name Server -** Any distributed system requires a mechanism to locate resources on the network. Either it can be done statically through configuration files or dynamically through name resolution. UH MDO framework supports locating the services dynamically. Name server keeps track of all components running in the system.

9. **Analysis Service Manager -** analysis service manager is a process running on each computer where analysis services are located. The analysis manager controls execution of all analysis modules.

10. **Analysis Services -** Analysis service is wrapped disciplinary analysis software. This is the only component, which is created by service provider. Analysis services are the grid services that can be accessed across the network.

6.4.2 Problem Formulation and Construction in the UH MDO Framework

A Java based Graphical User Interface is developed to create activation graphs and to execute the activation graphs. The user employs the framework GUI to define the analysis sequence. Through the GUI, the user identifies the execution sequence of different services along with their corresponding resource properties.

MDO Framework uses multithreading approach to perform the parallel execution is identified in the execution sequence. Directed edges in the sequence diagram denote the events, for which each thread waits on.

GMIS viewer supports creation and execution of activation graphs. It also provides the status of job execution. So instead of UH MDO Framework GUI, GMIS viewer can be used to formulate the problem.

6.4.3 Problem Execution in the UH MDO Framework

The framework automates the execution of the various services drawn in activation graph. For distributed computing support, Globus Toolkit 4(GT4) features were used which could provide a remote service invocation. The activation graph can be executed in parallel. The flow of the execution depends upon the notification from one service to other service.

GMIS supports the distributing computing support over WAN so GMIS services can be used for UH MDO Framework problem execution.

6.4.4 Information access in the UH MDO framework

The information regarding different domain names, services, their resource properties, and the input variables are stored in the database. The retrieval of the information is done while executing the framework. The transferring of files from one

system to other is transferred using *gridftp,* which is provided by Globus toolkit. These transferred files are displayed dynamically based upon the arrival of data.

GMIS services provide all resource information including applications information. In GMIS framework input/output files transfer happens through File Transfer Protocol (FTP). Using GMIS resource selector and monitor services required information can be accessed for UH MDO framework.

6.4.5 Mapping of UH MDO Framework Components to GMIS components

UH MDO framework has developed based on Globus, which can be migrated to GMIS as all the required components of UH MDO framework will be provided by GMIS framework. The following table gives the mapping of UH MDO Framework components to GMIS components.

UH MDO Framework component	GMIS component
GUI	Viewer
MDO Controller	Topology Manager
Database	Database
Optimization Manager	Resource Selector
Service Directory	Monitor Service
Execution Manager	Resource Selector
Name Server	Data Collection Manager
Analysis Service Manager	Resource Selector

Table 6.1: Mapping of MDO Framework Components to GMIS Components

6.5 UH MDO Application Design on GMIS viewer

UH MDO application design on GMIS viewer will be done in two levels, one is creation of activation graph and second one is execute activation graph. These two steps are discussed in detail in the following sub sections.

6.5.1 Creation of Activation Graph

The following steps are used to create activation graph on GMIS viewer.

1. Select a drawing Space for Activation graph, which can be done by selecting “Jobs-> Activation Graph -> Create” option in the menu bar of viewer. The drawing space created is shown in the following figure 6.2.

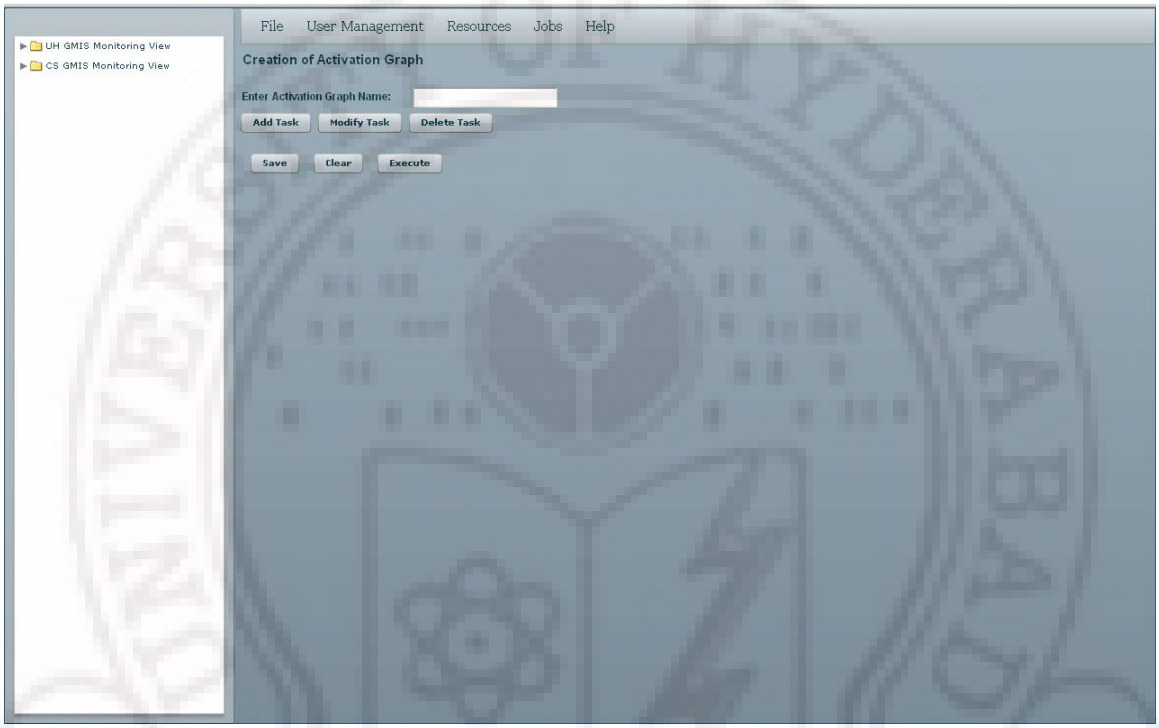

Figure 6.2: Creation of Activation Graph on Viewer

2. First give activation graph name for the activation graph which is going to be created. Then click on “Add Task” button to add a task. This will be the first task for an activation graph. There will be list options to select the node using FQDN names present in GMIS. After selecting the system, next list shows the services available in that particular node. Select service which needs to be executed. After that need to mention the input file location. Finally need to mention the dependent task(s) which needs to be executed before executing this task. If it is ‘0’ means this task is either the first task or can be executed in parallel. This is shown in the following figure 6.3.

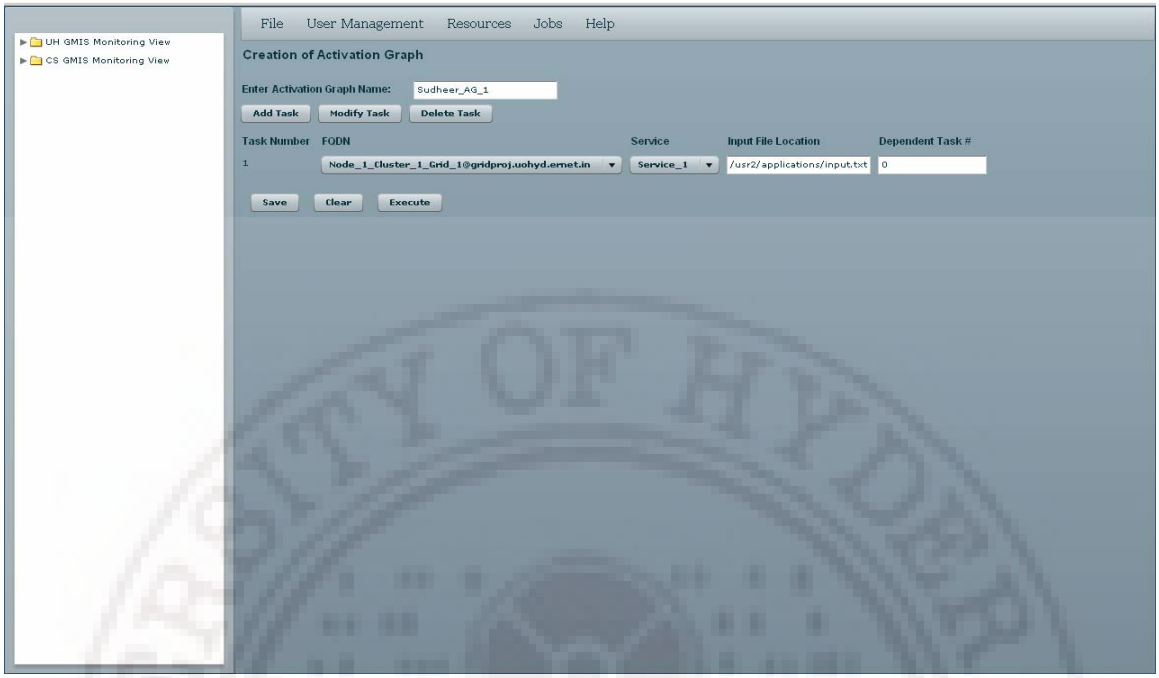

Figure 6.3: Add Task to Activation Graph

3. Next select "Save" button on the viewer. It shows a rectangular text box with task number as shown in the following figure 6.4.

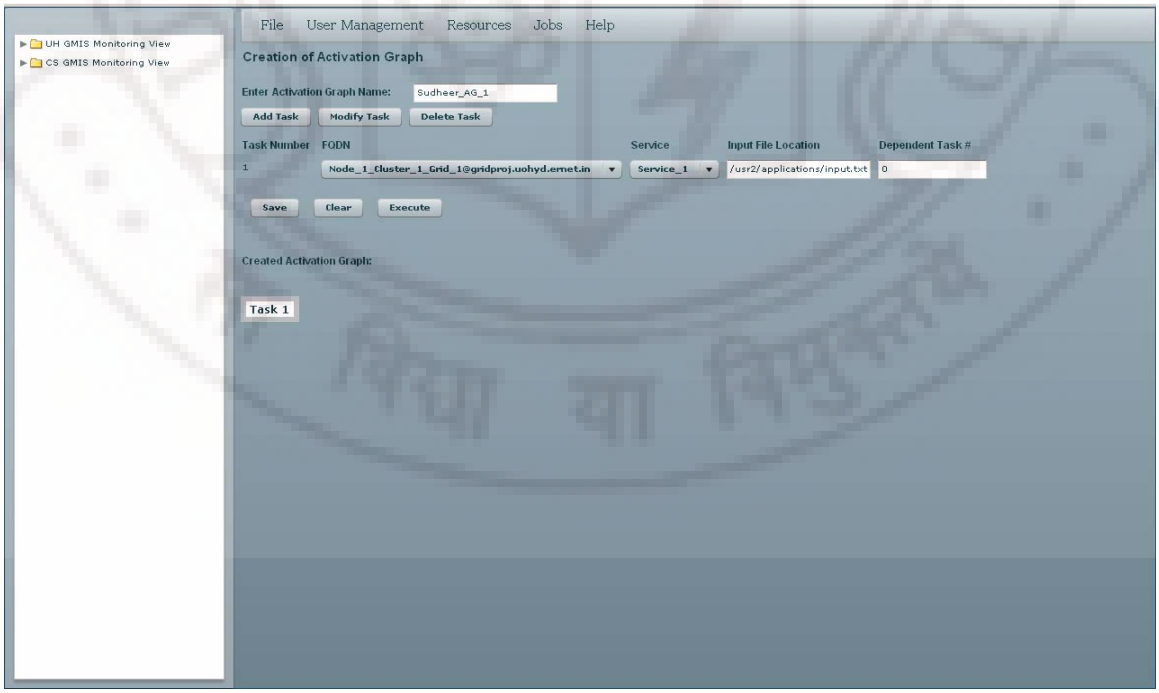

Figure 6.4: After Selection of "Save" Button in Creation of Activation Graph

4. Repeat steps 2 – 3 to add multiple tasks. Following figure 6.5 shows after adding three tasks to the activation graph.

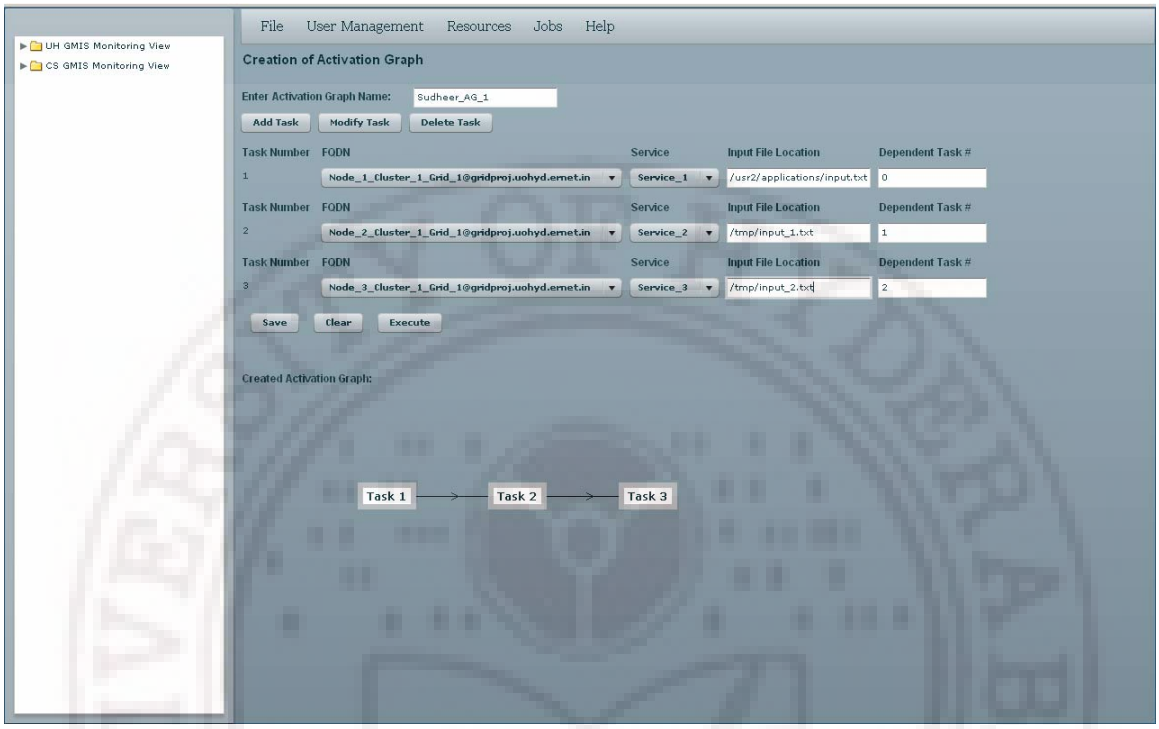

Figure 6.5: After Creation of Activation Graph with 3 Tasks

6.5.2 Execution

Two options are provided to execute the Activation Graph. First option is after creation of activation graph click on “Execute” button to execute the tasks according to the order mentioned. Second option is select “Execute” option from Jobs-> Activation Graph -> Execute option from the file menu. After selecting this menu it populates all the activation graphs created so far in a list. Select an activation graph which needs to be executed. Selected activation graph will be shown and option is provided to execute the activation graph. The screen shot of this option is shown in the following figure 6.6.

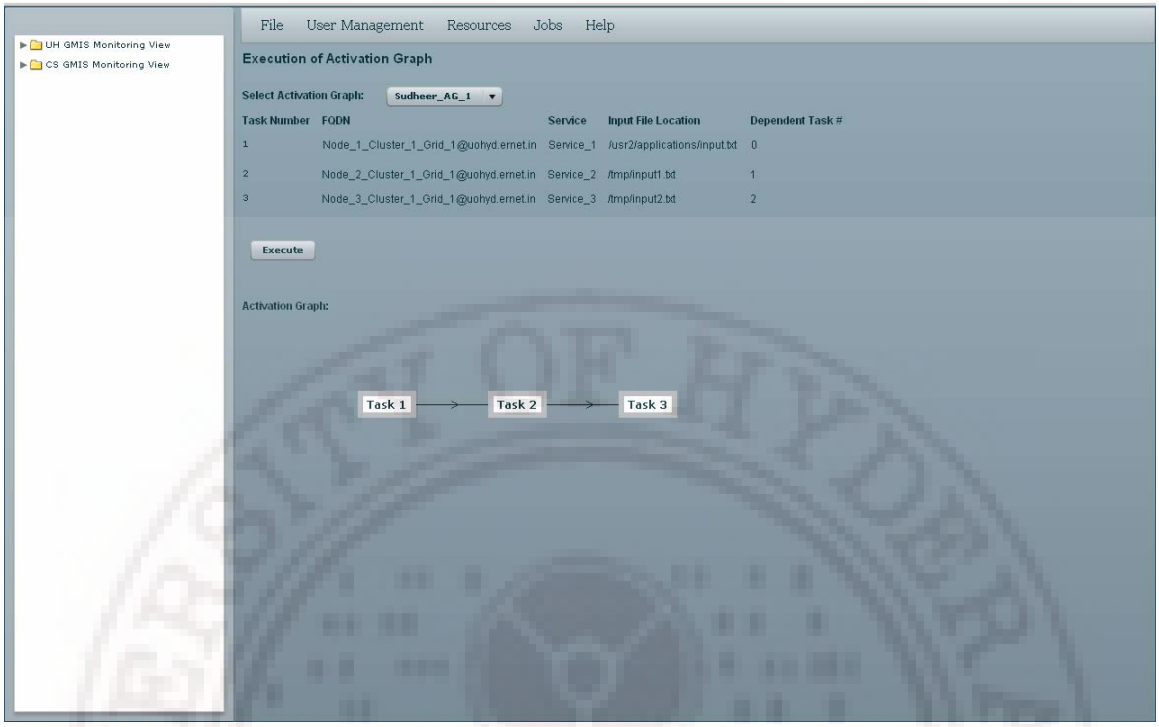

Figure 6.6: Execution of Activation Graph

6.6 Virtual Cluster

A virtual cluster is defined as a set of virtual machines configured to behave as a cluster and intended to be scheduled on a physical resource at the same time [Xuehai Zhang, 2005]. In other words, virtual cluster is a group of physical or virtual machines configured for a common purpose.

Virtual cluster is another application of GMIS framework. As all the resources information is stored in database and topology manager maintains the resources hierarchy, a user defined virtual cluster can be formed as shown the following figure 6.7.

In this figure 6.7 there are two physical clusters Cluster I and Cluster II. Cluster I contains three nodes and Cluster II contains two nodes. The requirement is to have a new cluster which contains node 1 of Cluster I and Node 5 of Cluster II. In reality forming such a cluster is expensive. However a virtual cluster can be constructed with these two nodes, since all the node information is available in the GMIS's database and nodes are maintaining by the topology manager. Virtual cluster formation can be done by using the user based discovery resource option of GMIS viewer. To create a virtual

cluster first create a cluster manually using the option provided on viewer and then add required nodes to this cluster manually using the interface provided on viewer. This cluster will be created under default grid. If any particular grid name required then first create a grid manually followed by adding clusters and nodes.

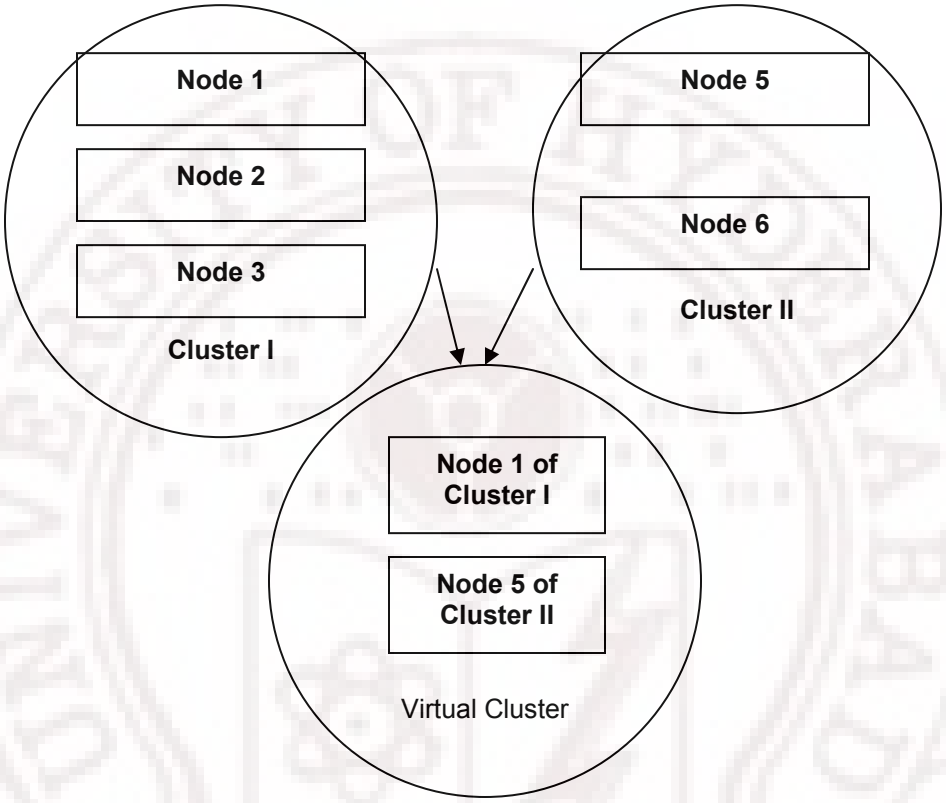

Figure 6.7: Formation of Virtual Cluster

6.7 Summary

UH MDO framework has been developed as a general programming environment for automating the distribution of complex computing tasks on nodes connected over a grid network. The framework system facilitates communications between computational tasks distributed over a grid network and provides the automatic interactions in multidisciplinary problems. Globus was selected as a distributed computing environment for UH MDO framework. This chapter discussed how GMIS services can be used for UH MDO framework, as all services required for UH MDO framework are provided by the GMIS. Hence UH MDO framework is chosen as a case study for GMIS framework. Virtual cluster formation using GMIS is also discussed in this

chapter. These two applications demonstrate how GMIS services can be used as distributed computing environment over WAN.

Next chapter concludes the thesis and discusses the enhancements of GMIS framework.

CHAPTER 7

Conclusions and Future Enhancements

7.1 Summary

In Grid environment, the resources are heterogeneous and geographically distributed with varying capacities, usage and constraints. The management of resources and job scheduling in such large and distributed environment is a complex task. We have developed an architectural resource management framework called Grid Management Information Server (GMIS) to address this complex task. To support the thesis, that an efficient grid resource management system can deliver a significant value to users as compared to traditional approaches, we have:

- Identified key requirements that an efficient resource management system that needs to support.
- Identified important design considerations for efficient resource management system.
- Compared with the traditional resource management frameworks.
- Identified and included several features not available in the other popular resource management frameworks.
- Developed a distributed resource management framework called Grid Management Information Server (GMIS).
- Designed resource discovery algorithms with three different strategies i.e., dynamic, event based and user based discovery.
- Developed monitoring and data collection manager services to provide users with wide range of options to select resources based on their capabilities and historical performance.
- Developed push and pull based algorithms for gathering resource information, to ensure that the grid users will have latest status about the resources as well as submitted jobs.
- GMIS provides support for activation graph creation based job submission and execution i.e., submitted jobs can be divided into multiple tasks and results can be analyzed at each task level.

- Shown how GMIS can be used as distributed computing environment over a WAN for UH Multidisciplinary Design Optimization framework.
- Demonstrated how a virtual cluster can be formed using GMIS.

7.2 Conclusions

Grids have evolved rapidly and are becoming a good alternative for the otherwise costly supercomputers. The tools and number of applications being built around these technologies are also increasing day by day. The resource management in grid environment is a complex task, because resources are distributed over wide area network and dynamic nature of resource availability.

The resource management systems used in grid environments need to be adaptive so that they can handle dynamic changes in availability of resources and user requirements. At the same time, they need to provide scalable, controllable, measurable, and easily enforceable policies for management of the resources. To address these requirements, a resource management framework called Grid Management Information Server (GMIS) has been developed. The GMIS provides dynamic discovery of resources, monitoring of resources and brokerage service. Since the GMIS framework is a component based architecture implementation can be easily done for different middleware technologies.

The information services used in grid environments need to be highly flexible in nature to handle the dynamic behavior of the grid network resources. Additionally, they need to be scalable and easily implement able in grid based environments. The GMIS framework addresses these requirements. Several features and design considerations in GMIS framework are not available in the other popular frameworks.

The GMIS framework describes the resource management for grid environment that supports resource discovery, monitoring and brokerage with integrated topology information. This frame work is an open and extensible resource monitoring system, based on the Client - Server Architecture. The GMIS prototype is developed and efforts are currently underway to develop complete services.

7.3 Future Enhancements

This thesis formulated a comprehensive distributed resource management architectural framework. Resource management algorithms have been developed to meet grid users' requirements. This work is laid as a foundation for the efficient resource management and it opens up several avenues for future work in efficient resource management in grid environment.

As the complete implementation requires more effort, only a prototype has been developed. Complete implementation will be done in future as part of UH Grid development projects. As part of this, there is a need to focus on synchronization issues, since this framework depends on message communications among the GMIS components and as well as with resources agents communications. The scope of future enhancements includes, a feature displaying the current status of job and its details, as currently it shows only a set of states like job execution started, completed or failed. Some of the important enhancements are listed below:

- **Authentication and Authorization:** Currently the authentication and authorization of GMIS users is manual process. User requests for GMIS services access permissions and GMIS administrator manually verifies the user request. This process can be automated similar to generation of digital certificates of Globus.

- **Middleware support:** GMIS framework needs to be integrated with other frameworks like Globus, GridBus for which a set of APIs development is required. Current developed APIs is useful for scheduler level frameworks.

- **Enhance GMIS-SIM:** Current GMIS-SIM is a simple process which simulates resources from response configuration files. This needs to be enhanced to simulate the network model which supports various types of networks with different static and dynamic configurations. A lot of real time examples should be tested on the framework, so that the shortcomings of the framework could be seen. For testing real time examples the GMIS-SIM need to be enhanced.

- **Enhancements to Brokerage Service:** Currently GMIS brokerage service supports only dynamic resource allocation. There is no provision of reserving resources in advance. This need to be incorporated in GMIS brokerage service. Economic models also need to be adapted to the brokerage service.

- **Crash Recovery:** Crash recovery is an important area, which needs to be addressed in GMIS framework. If the node executing a task which is responsible for supplying input data to the other remote node crashes, then the job or activation graph execution could fail. Hence a mechanism for crash recovery also needs to be evolved in GMIS framework, so that even if the node fails in supplying input data, then the job execution should restore to a stable state. This can be achieved by having check points in job execution process.

- **Poll Package Optimization:** While packing Object Identifiers (OIDs) for polling a set of managed resource objects, the number of OIDs that need to be polled in a single poll request packet can be optimized based on the OID lengths and message buffer limits. Packing related object OIDs into smaller number of poll packets reduces the traffic between the GMIS communicator server and the resources.

BIBLIOGRAPHY

1. [Arun Agarwal, 2004] Development of Grid Middleware for Integration of Dispersed Resources By Prof. Arun Agarwal, Department Of Computer and Information Sciences, School of Mathematics & Computer/Information Sciences, University of Hyderabad, Hyderabad 500046, INDIA, September 2004.
2. [ASN.1,1987] Information processing systems - Open Systems Interconnection, "Specification of Abstract Syntax Notation One (ASN.1)", International Organization for Standardization, International Standard 8824, December 1987.
3. [Bester J, Foster I, 1999] J. Bester, I. Foster, C. Kesselman, J. Tedesco, S. Tuecke, *GASS: A Data Movement and Access Service for Wide Area Computing Systems,* Proceedings of the Sixth Workshop on Input/Output in Parallel and Distributed Systems, Atlanta, GA, May 1999.
4. [Buyya, VenuGopal, 2004], Rajkumar Buyya and Srikumar Venugopal, "The Gridbus Toolkit for Service Oriented Grid and Utility Computing: an Overview and Status Report", 2004.
5. [Chidambaram, 2008] J. Chidambaram, "An algorithm for high availability of data in distributed database using partial replication in Grids", M.Tech Thesis, University of Hyderabad, Hyderabad, 2008.
6. [Condor]: http://www.cs.wisc.edu/condor/
7. [Czajkowski, K., 2001] Czajkowski, K., S. Fitzgerald, I. Foster, and C. Kesselman, "Grid Information Services for Distributed Resource Sharing". In Proc. 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10), IEEE Press, 2001.
8. [Dan Kusnetzky, 2004] Dan Kusnetzky, Carl W. Olafson,"Oracle 10g: Putting Grids to Work", White paper, April 2004.
9. [DAKOTA]: DAKOTA - http://endo.sandia.gov/DAKOTA/software.html
10. [DataGrid, 2002] "DataGrid Information and Monitoring Services Architecture: Design, Requirements and Evaluation Criteria", Technical Report, DataGrid, 2002.
11. [DataGrid] : DataGrid: http://eu-datagrid.web.cern.ch/eu-datagrid/
12. [Distributed Computing] http://e-comm.webopedia.com/TERM/D/distributed_computing.html

13. [FAFNER]: Factoring via network enabled recursion. http://cswww.bu.edu/cgi-bin/FAFNER/factor.pl
14. [Foster, I, C. Kesselman, 1997] Foster, I., and C. Kesselman, “Globus: A Metacomputing Infrastructure Toolkit”. Supercomputer Applications, 11(2):115–128, 1997.
15. [Foster, I, C. Kesselman, 2001] Foster, I., Kesselman, C. and Tuecke, S. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International Journal of High Performance Computing Applications, 15 (3). 200-222. 2001.
16. [Foster, I, C. Kesselman, 2002] I. Foster, C. Kesselman, J.M. Nick and S. Tuecke, “The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration”, *Open Grid Services Infrastructure WG*, Global Grid Forum, June 22, 2002, also appears as “Grid Services for Distributed System Integration”, *IEEE Computer*, 35(6), 2002.
17. [Fisher, S., 2001] Fisher, S., “Relational Model for Information and Monitoring”. Technical Report GWD-Perf-7-1, GGF, 2001.
18. [GGF] : http://www.ggf.org/
19. [Globus] The Globus Alliance. http://www.globus.org
20. [GridbusBroker] : http://www.gridbus.org/broker/
21. [Grid Computing] : http://www.jayeckles.com/research/grid.html
22. [Grimshaw, 1994] A. S. Grimshaw, William A. Wulf, James C. French, Alfred C. Weaver, Paul F. Reynolds Jr, "A Synopsis of the Legion Project", Technical Report No. CS-94-20, June, 1994.
23. [Grimshaw, 1997] A. S. Grimshaw, Wm. A. Wulf, and the Legion Team. The legion vision of a worldwide virtual computer. Communications of the ACM, 40(1), January 1997.
24. [Hawkeye]: http://www.cs.wisc.edu/condor/hawkeye
25. [Hrudayanath, 2008], “Adding the Meta-scheduling Feature to Portable Batch Scheduler (PBS) through High Performance Computing Basic Profile”, M.Tech Thesis, University of Hyderabad, Hyderabad, 2008.
26. [IWAY, 1996] I. Foster, J Geisler, W Nickles, W. Smith and S. Tuecke: Software infrastructure for the I-WAY high performance distributed computing experiment. In Proc. 5th IEEE Symposium on High Performance Distributed Computing, 1996.

27. [Ian Foster, Carl Kesselman, 1999] Foster, Ian and Carl Kesselman ed.: The Grid: Blueprint for a New Computing Infrastructure; Morgan Kaufmann; San Francisco; 1999.
28. [IBM developerWorks], New to Grid Computing : http://www.ibm.com/developerworks/grid/newto/
29. [IBM Red Book, 2003], Introduction to Grid Computing with Globus, September, 2003.
30. [IBM Red Book, 2004], Grid Computing with the IBM Grid Toolbox, May, 2004.
31. [Internetworking, 2001], Internetworking Technologies Handbook, 4th Edition, Cisco Press. 2001.
32. [iSIGHT, 1998] Engineous Software, Inc., iSIGHT Designer's Guide, *Version 3.1*, Morrisville, NC, Apr. 1998.
33. [Java Servlet Technology]: http://java.sun.com/products/servlet
34. [Jennifer, Bill, 2002], Jennifer M. Schopf and Bill Nitzberg, "Grids: The Top Ten Questions", 2002.
35. [Keahey, 2005], Keahey, K., I. Foster, T. Freeman, X.Zhang, and D.Garlon, "Virtual Workspaces in the Grid", TR-ANL/MCS-P1246-0405, April 2005.
36. [Klaus, Rajkumar, 2002] Klaus Krauter, Rajkumar Buyya, and Muthucumaru Maheswaran, "A Taxonomy and Survey of Grid Resource Management Systems for Distributed Computing, Software: Practice and Experience (SPE)", ISSN: 0038-0644, Volume 32, Issue 2, Pages: 135-164, Wiley Press, New York, USA, February 2002.
37. [LIVNY M, 1995] LIVNY M, "The Condor Distributed Processing System", Dr Dobbs Journal, February 1995.
38. [LMS, 1997] LMS Numerical Technologies, LMS Optimus Revision 2.0 Users Manual, Leuven, Belgium, Oct. 1997.
39. [Maheswaran, 2001] Muthucumaru Maheswaran, "Data Dissemination Approaches for Performance Discovery in Grid Computing Systems", IEEE 2001.
40. [Mark Baker, 2003] Mark Baker and Garry Smith, "GridRM: An Extensible Resource Monitoring System", Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER'03), 2003.
41. [Mathijs, Thilo, Henri, 2002], Mathijs den Burger, Thilo Kielmann, Henri E.Bal, "TOPOMON: A Monitoring Tool for Grid Network Topology", International

Conference on Computational Science (ICCS 2002), April 21 – 24, 2002 Amsterdam, The Netherlands.

42. [Amitay, 2003] Amitay Isaacs, K Sudhakar, P M Mujumdar "Design and Development of MDO Framework" Center for Aerospace Design and Engineering, Department of Aerospace Engineering, IIT Bombay, 2003.
43. [MDS] : http://www.globus.org/mds/
44. [Meta computing] : http://en.wikipedia.org/wiki/Metacomputing
45. [Microsoft .NET] : http://www.microsoft.com/NET/
46. [Multics] : www.multicians.org
47. [Mustafa, 2008], "Design and Implementation of a structured fine-grained access control mechanism for authorizing Grid resources", M.Tech Thesis, University of Hyderabad, Hyderabad, 2008.
48. [Naveen, 2008], "Dynamic Ants Resource Discovery System for a Grid", M.Tech Thesis, University of Hyderabad, Hyderabad, 2008.
49. [Narsimha Rao, 2008], P.A. Narsimha Rao, "Extending OGSA DQP framework for high availability", M.Tech Thesis, University of Hyderabad, Hyderabad, 2008.
50. [OpenLdap] OpenLdap: http://www.openldap.org/
51. [O. Salas, 1998] O. Salas, J. C. Townsend, "Framework Requirements for MDO Applications Development," AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, 7th, St. Louis, MO, Sept. 2-4, 1998.
52. [Parallel Computing] http://www.llnl.gov/computing/tutorials/parrel_comp/
53. [Pramod Kumar, 2006], "Dynamic Scheduling and Resource Management for High Throughput Clusters", PhD thesis, University of Hyderabad, Hyderabad, India, 2006.
54. [Peggy Lindner, 2003] Peggy Lindner, Thomas Beisel, Michael M. Resch, Toshiyuki Imamura, Roger Menday, Philip Wieder, Dietmar Erwin, „GRIDWELTEN: User Requirements and Environments for GRID-Computing", DFN-Project report, 2003.
55. [PRAGMA]: www.pragma-grid.net
56. [Rajkumar Buyya, 2002] Rajkumar Buyya, "Economic-based Distributed Resource Management and Scheduling for Grid Computing", Ph.D. Thesis, Monash University, Melbourne, Australia, April 12, 2002.
57. [Rajkumar Buyya, Manzur Murshed, 2002] Rajkumar Buyya and Manzur Murshed, GridSim: A Toolkit for the Modeling and Simulation of Distributed

Resource Management and Scheduling for Grid Computing, Concurrency and Computation: Practice and Experience (CCPE), Volume 14, Issue 13-15, Pages: 1175-1220, ISSN: 1532-0626, Wiley Press, New York, USA, November - December 2002.

58. [Rajkumar Buyya, David Abramson, 2002] Rajkumar Buyya, David Abramson, Jonathan Giddy, and Heinz Stockinger, Economic Models for Resource Management and Scheduling in Grid Computing, Concurrency and Computation: Practice and Experience (CCPE), Volume 14, Issue 13-15, Pages: 1507-1542, ISSN: 1532-0626, Wiley Press, New York, USA, November - December 2002.
59. [Rajkumar Buyya, Steve Chapin, 2000] Rajkumar Buyya, Steve Chapin, and David DiNucci, Architectural Models for Resource Management in the Grid, The First IEEE/ACM International Workshop on Grid Computing (GRID 2000), Springer Verlag LNCS Series, Germany, Dec. 17, 2000.
60. [Raman, R., 2001] Raman, R., “Matchmaking Frameworks for Distributed Resource Management”. PhD thesis, Department of Computer Science, University of Wisconsin, 2001.
61. [RFC 1065, 1988] Rose, M., and K. McCloghrie, "Structure and Identification of Management Information for TCP/IP-based internets", RFC 1065, TWG, August 1988.
62. [RFC 1066, 1988] McCloghrie, K., and M. Rose, "Management Information Base for Network Management of TCP/IP-based internets", RFC 1066, TWG, August 1988.
63. [RFC 1155, 1990] Rose, M., and K. McCloghrie. Structure and Identification of Management Information for TCP/IP-based Internets. May 1990.
64. [RFC 2790, 2000] Waldbusser S., Grillo P. “Host Resources MIB”, Network Working Group, the Internet Society, 2000.
65. [Saran, 2005], A Framework for Multidisciplinary Design Optimization Over Grid, M.Tech Thesis, University of Hyderabad, Hyderabad, 2005.
66. [S. Baburao, 2008], “Design of Grid Framework for Scientific Applications”, M.Tech Thesis, University of Hyderabad, Hyderabad, 2008.
67. [SCMS, 2003] Putchong Uthayopas, Jullawadee Maneesilp, Paricha Ingongnam, “SCMS: An Integrate Cluster Management Tool for Beowulf Cluster System”, Parallel Research Group, Kasetart University, Bangkok, Thailand, 2003.

68. [Schopf, 2002] J. Schopf. A General Architecture for Scheduling on the Grid. Journal of Parallel and Distributed Computing, 2002.
69. [SETI] SETI @ Home Internet Computing, http://setiathome.ssl.berkeley.edu/
70. [Sotomayor, 2006] Sotomayor, B., K. Keahey, I. Foster, “Overhead Matters: A Model for Virtual Resource Management”, VTDC 2006, Tampa, FL, November 2006.
71. [Srikumar, Rajkumar 2004] Srikumar Venugopal, Rajkumar Buyya and Lyle Winton, A Grid Service Broker for Scheduling Distributed Data-Oriented Applications on Global Grids, Technical Report, GRIDS-TR-2004-1, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia, February 2004
72. [Srinivas, 2008] Ch. Srinivas, “Building and Integrating Web Grid Services for Ocean Science Applications”, M.Tech Thesis, University of Hyderabad, Hyderabad, 2008.
73. [Srivastav Aditya, 2005], Grid Computing to Grid Services: An investigative study, M.Tech Thesis, University of Hyderabad, Hyderabad, 2005.
74. [SUN Grid] : http://www.sun.com/software/grid/
75. [SGE Features] : http://www.sun.com/software/gridware/features.xml
76. [Steven Fitzgerald, 2000] Steven Fitzgerald, Ian Foster, Carl Kesselman, Gregor von Laszewski, Warren Smith, Steven Tuecke, “A Directory Service for Configuring High-Performance Distributed Computations”, 2000.
77. [T.Hey, A.E. Trefethen, 2002] T. Hey and A. E. Trefethen, The UK e-Science Core Programme and the Grid, *Future Generation Computer Systems*, Volume 18, Issue 8, Pages 1017-1031, October 2002.
78. [Tierney, B., R. Avdt, 2002] Tierney, B., R. Aydt, D. Gunter, W. Smith, V. Taylor, R. Wolski, and M. Swany, “A Grid Monitoring Architecture”. The Global Grid Forum GWD-GP-16-2, January 2002.
79. [Townsend, 1993] Townsend, J. C., Weston, R. P., and Eidson, T. M., “A Programming Environment for Distributed Complex Computing - An Overview of the Framework for Interdisciplinary Design Optimization (FIDO) Project,” NASA TM 109058, Dec. 1993.
80. UDDI: Universal Description, Discovery and Integration. www.uddi.org.
81. [Utla Raju, 2008], “An efficient design method for searching resources in multiple grid portal nodes”, M.Tech Thesis, University of Hyderabad, Hyderabad, 2008.

82. [Wolski, R, 1999] R. Wolski, N. Spring, and J. Hayes, “The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing”, Journal of Future Generation Computing Systems, Volume 15, Numbers 5-6, pp. 757-768, 1999.
83. [Xuehai, Jeffrey, Jennifer, 2003] Xuehai Zhang, Jeffrey L. Freschl, and Jennifer M. Schopf, “A Performance Study of Monitoring and Information Services for Distributed Systems”, Proceedings of HPDC-12, 2003.
84. [Xuehai Zhang, 2005] Xuehai Zhang, Katarzyna Keahey, Ian Foster, Timothy Freeman, “Virtual Cluster Workspaces for Grid Applications”, ANL Tech Report: ANL/MCS-P1246-0405, 2005.
85. [Yulan Yin, 2007] Yulan Yin, Huanquing Cui and Xin Chen, "The Grid Resource Discovery Method Based on Hierarchical Model", Information Technology Journal 6(7): 1090-1094, 2007.
86. [William E. Johnston, 2002] William E. Johnston, “The Grid Architecture”, Distributed Systems Department, Computational Research Division, Lawrence Berkeley National Laboratory, 2002.