Jül-4263

Jülich Supercomputing Centre

A Reliable Grid Information Service Using a Unified Information Model

Mohammad Shahbaz Memon

Mitglied der Helmholtz-Gemeinschaft

JÜLICH
FORSCHUNGSZENTRUM

A Reliable Grid Information Service Using a Unified Information Model

Mohammad Shahbaz Memon

Berichte des Forschungszentrums Jülich ; 4263
ISSN 0944-2952
Jülich Supercomputing Centre Jül-4263

The complete volume is freely available on the Internet on the Jülicher Open Access Server (JUWEL) at http://www.fz-juelich.de/zb/juwel

Zu beziehen durch: Forschungszentrum Jülich GmbH · Zentralbibliothek, Verlag
D-52425 Jülich · Bundesrepublik Deutschland
☎ 02461 61-5220 · Telefax: 02461 61-6103 · e-mail: zb-publikation@fz-juelich.de

Abstract

An information system is an essential part of every Grid system since it provides information about the entities (services, resources, user, etc.) a Grid consists of. The information provided by such a system may serve a variety of needs. It can for example be used for brokering purposes, to schedule workflows, to orchestrate services, or to predict the performance of a Grid. Such use cases very much determine the information model and the design of the information service. In this thesis, I present an open standard, platform-agnostic, and web services based information service. Since a Grid information service consolidates disseminated information of diversified Grid entities, the modeling of this information has a large impact on the Grid information service while executing its business functions. In this thesis, I formalize a resource schema which models an elementary concepts required to represent significant Grid entities.

The question of reliability and fault tolerance is often not properly answered when it comes to the realization of distributed systems, although the deployment of any system in a distributed environment shows that those factors are of utmost importance. As the Grid information service developed during the diploma theses will be deployed in a highly-distributed Grid, and is one of the essential services for the Grid to operate properly, it is expected to satisfy certain reliability criteria. In this thesis, I present a reliability aspect of Grid information service through a multi-tier replication infrastructure by focusing on its architectural and implementation details.

Acknowledgements

I would like to offer my sincere gratitude to Dr. Felix Wolf for supervising my thesis. I am also grateful to Dr. Christian Bischof for providing important suggestions in the course of my thesis. I express my immense gratitude to Dr.Achim Streit for technical supervision and giving me an opportunity to accomplish my thesis in Forschungszentrum Jülich. My utmost thanks to Philipp Wieder and Bernd Schuller whose valuable suggestions, guidance, encouragement and continuous assistance made the completion of thesis.

Last but not least, I would like to admit that all my achievements are due to my family and parents, who are always a source of encouragement and motivation for me.

Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 Problem Statement

In recent years, Grid computing has been considered vital for evolving the distributed computing paradigm towards the next generation internet. As a matter of fact Grids complement traditional distributed computing infrastructures [1] and provide heterogeneous, distributed, coordinated and decentralized frameworks to manage Grid entities. The Grid encompasses the arrangement of resources, services, organizations, and users. In a typical Grid computing infrastructure, the entities are widely distributed. In order to manage the Grid-oriented processes, Grid middlewares are used to hide underlying complexities of process communication, resource and user management, and job scheduling. UNICORE [2], Globus [3] and GLite [4] are state-of-the-art middlewares serving in Grid environments. Hence they aim at providing seamless and transparent access to core Grid services e.g. resource management and job scheduling. Essentially, a Grid middleware serves the deployment of user applications such as computing and data clustering services.

As middleware is the focal point of any Grid environment, its core Grid services are intended to deal with the efficient utilization of resources. In order to achieve effective resource exploitation, the middleware should ideally have complete information of all hosted resources and interfaces to handle information access and storage mechanisms. With respect to information access consider an example of a job scheduler which is responsible for scheduling the execution of jobs based on available resources. To start a particular job, the scheduler needs the information about particular resources that fulfill certain criteria. Hence, the separate searching interface would facilitate the scheduler to find resources of its choice; hence without this middleware service it will be nearly impossible to execute jobs. , Furthermore, functions to advertise resource information are needed. As a first step a user provides sufficient details about the resource to the Grid middleware. The middleware needs to provide a registry interface to get resource information from user; then it registers and maintains this data in a location where it can

be further consumed by other components. The absence of a registry interface will make the middleware incapable of storing resource details, thereby paralyzing other middleware components to deal with resources.

This leads to the conclusion that without the management of resource information, Grid middlewares cannot deliver the desired level of quality of service. Thus, Grid middleware necessitates a separate registry service and associated search mechanisms. In the Grid community this component is called Grid information service, a service offering interfaces to search, insert, and update resource information. Furthermore, Grid information services

"[...] are concerned with the extraction and presentation of data with meaning by using the services of computational, data, and/or application services. The low-level details handled by this are the way that information is represented, stored, accessed, shared, and maintained. Given its key role in many scientific endeavors, the Web is the obvious point of departure for this level." [5]

Currently, there is no such information service perfectly managing versatile resource information and simultaneously offering programmatic APIs and usable front ends. Currently available information service tools like MDS [6] or R-GMA [7] do not contain all of the required features at one place.

The primary challenge is to exploit information service interfaces that allow the sharing and access of resources in a Grid environment. For this concern, the backend resource information has to be modeled in a correct and complete manner. Until the information is not modeled schematically and semantically correct, it is not possible for an information service to manage resource data. Without solving this problem, it will behave ambiguously while extracting and inserting information. For information modeling the challenge is to identify and characterize the resource concepts, and categorize the complex relationships among them. In this context the GLUE [8] information model solved this problem but lacking important parameters of extensibility, abstraction and completeness. This thesis evaluates other models and formulates a unified model using open source and open standard technologies.

In Grid computing, where resources are distributed all around the globe, hardware or software failure is highly probable. Therefore, in any distributed system, reliability and failover is considered to be a non-functional requirement and secondary compared to all other requirements. Other information services discussed above however do not provide a solution for the reliability and failover problems.

In summary, my thesis is primarily inclined towards the realization of the information service and resource modeling. Secondarily, the challenge is to design an architecture for a highly reliable information service through replication, along with a suitable implementation strategy.

1.2 Goals

The major goal of my thesis is to develop a Grid information service providing optimal resource discovery mechanisms based on a resource information model representing resource concepts in a consistent and structured manner. The information service can be used with any Grid middleware to manage information about resources.

Complementing Section 1.1, the proposed information service should be capable of

- registering resource information compliant with the information model,
- updating resource information from the service registry,
- providing an interface to perform convenient and template based search,
- exposing methods and business functions through web services, and
- communicating with remote clients via XML based protocols.

The preliminary requirements of information modeling are

- a unified model capturing all resource concepts from the perspective of type (software and hardware) and environment (computational and data),
- the schema based on the object-oriented paradigm,
- the model which clearly distinguishes between abstract and concrete classes,

- the refined hierarchy allowing extensibility, so that future concepts can easily be integrated,
- the model customized for UNICORE Grids, and
- an implementation based on interoperable and open standards such as XML schema.

Besides the functional requirements, the information service must comply with the non-functional requirement of reliability, and provide a framework:

- with a detailed replication architecture catering reliability on each functional layer,
- with a flexible and scalable architecture to allow future enhancements and extensibility, and
- guaranteeing high availability in order to achieve negligible downtime.

Primarily this thesis is influenced by the requirements of the research project NextGRID [9], an Integrated Project (IP) granted by the European Commission. The secondary requirements will be gathered from UNICORE [2], especially in terms of information modeling and persistence mechanisms. Both types of requirements will be represented by open standards tools and technologies.

1.3 Organization of this thesis

The remaining chapters are organized as follows

Chapter 2 (State-of-the-art) briefly describes the tools and technologies (web services, service oriented architecture) and state-of-the-art information models.
Chapter 3 (Application scenarios and requirements analysis) explains the requirement specification of Grid information services, specifically possible use cases, functional and non-functional requirements.
Chapter 4 (Design and implementation of Grid information service architecture) describes the detailed system design, architecture, and implementation details to realize service-oriented Grid information systems.

Chapter 5 (Replication infrastructure of Grid information service) explicates the architectural framework and the implementation strategy for the Grid information services.

Chapter 6 (Related works) explores related approaches and compares them with our approach.

Chapter 7 (Conclusion and future work) summarizes the accomplishment, key features, applications, and advantages of modeling an information service for Grid environments. The chapter concludes with possible future work to enhance the information schema and evolve the Grid information service.

Chapter 2
State-of-the-Art

This chapter contemplates the current research and development trends of Grid computing applications and tools. The chapter starts with an overview of Grid computing and then details the background of selected existing Grid middleware systems. Finally the emphasis will be put on the service-oriented architecture and its adaptability in architecting Grid computing infrastructures. The chapter concludes with details of state-of-the-art information models used in current Grid middlewares.

2.1 Grid Computing

2.1.1 Introduction

The term Grid computing cannot be defined easily. The definition depends upon the individual perception, the context, and research direction. There are many fields expanding in different dimensions of Grid computing. The basic concept behind Grid computing is the

"[...] coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations." [1]

The Grid's underlying concept is based on the seamless, secure, and independent access of shared resources. The Grid can be described as inter-institutional resource sharing analogous to the electric power Grid. It means the Grid environment provides economical and distributed access to inter-institutional shared resources with certain aspects of accountability and usage provenance. In order to exploit the need of seamless acquisition, resources are provided to get shared between and consumed by the entities of a Grid. As a result of this, one distinguishes between resource providers and resource consumers. Therefore Grid middlewares are there to cater this need by providing a level of transparency and ease of service environment; more details are discussed in the next section.

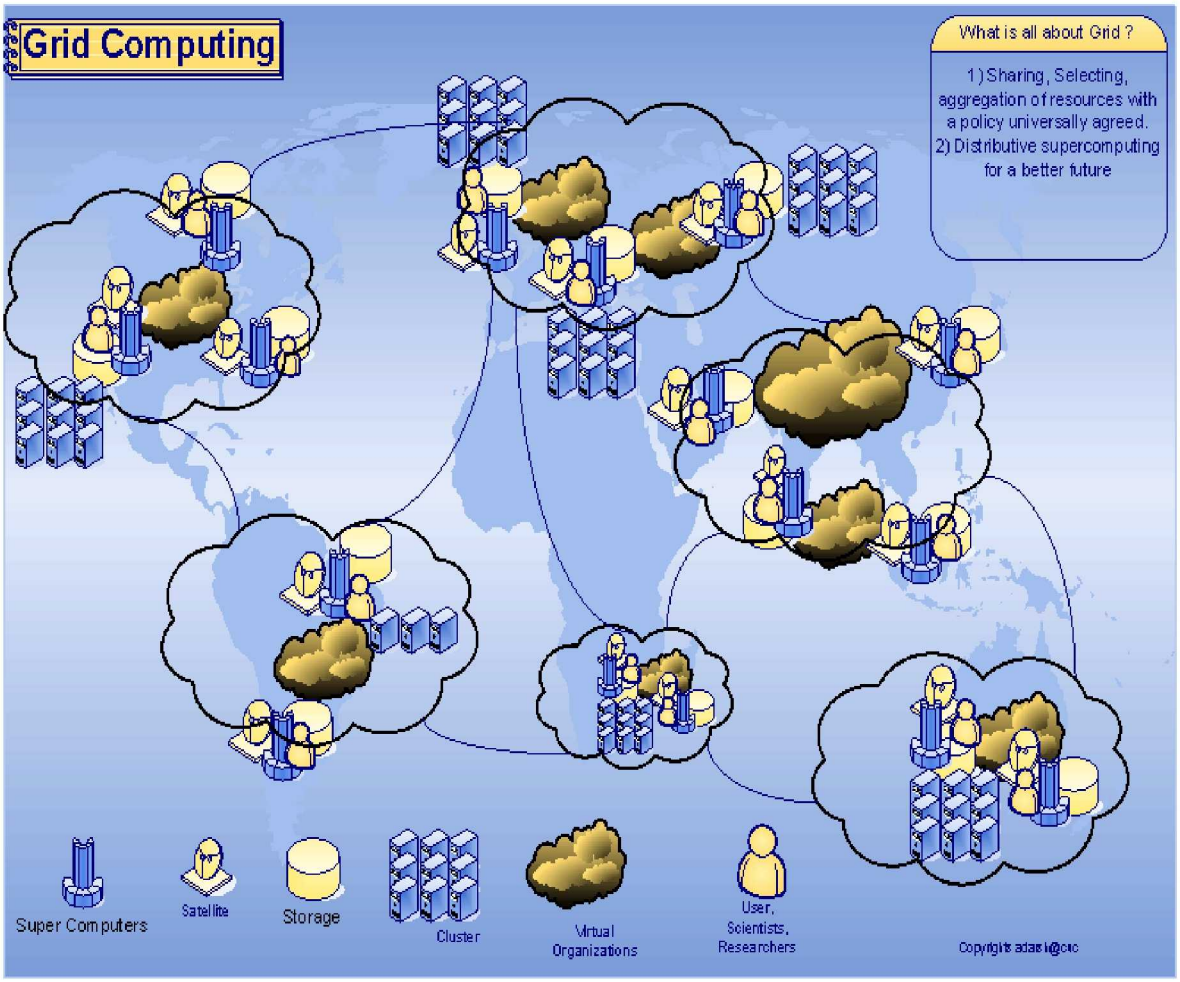

Figure 2.1: A Grid computing environment showing users in virtual organization accessing resources (like supercomputers, storage, …) [10].

The resource types are not only confined to only file sharing like in traditional Peer-To-Peer systems, moreover it can be hardware and software resources such as computers, applications, data bases, etc. The resources cannot exist in isolation; normally Grid resources are grouped and combined with respect to organization, type, and strength. In this way resources in a distributed environment can be managed guided by the concept of a *Virtual Organization (VO)*. The VOs can be structured in the form of hierarchies analogous to real organizational entities, e.g. a manager has sub-ordinates as sub-managers and they are leading various groups of employees. In the same way a VO aggregates nested VOs. This implies that, due to the hierarchical nature of the approach, there need to be certain measures of security and access policies by which appropriate resource access and management can be handled. The security aspect is one of the essential requirements to manage Grid environment in VOs. Additionally, the entities

within a Virtual Organization are dynamic, i.e. one cannot predict the lifetime or guarantee the presence of resources. Figure 2.1 depicts a real world Grid environment containing resources and users that are grouped in VOs. In summary, all the complexities of a Grid like handling resource access, sharing resources or security should be carefully handled, thus allowing commercial and public organizations to reap its benefits.

There is misconception between the terms Grid and cluster computing, and even people try to use them interchangeably. In cluster computing the resources are often merged and confined to a single organization, while a Grid is divided in VOs which are further aggregated in multiple organizations. In clusters the nodes formed in one location and they cannot become dynamic, which means they cannot leave and join the cluster at any time. In a Grid, VOs are formed by the individual participants who can freely join and leave at any point of time. This makes a Grid highly-dynamic in nature. There is no such restriction of their availability, until some rules are defined at the VO level. The section below explicates the different types of Grids with references to real projects.

2.1.2 Types of Grids

As already discussed, Grid computing allows access to distributed and hetrogeneous resources through one platform, and to cater application requirements, Grids can be used in numerous ways. IBM researchers described three types of Grids [11]. These types have no rigid boundaries, and can be described through application usage and user requirements.

Computational Grid

The computational Grid groups computational resources together. The shared resources are often high performance computing servers.

Scavenging Grid

This type of Grid focuses on the grouping of shared desktop machines. The scavenging Grid gathers desktop machines to provide computational and data resources. For example SETI@HOME is a Grid environment which allows users to share and utilize desktop machines around the world.

Data Grid

A Data Grid virtualizes the group of resources which are oriented towards the effective access and sharing of large pools of data. The article [11] describes its application in shared databases. A Data Grid is not only catering the use of databases but it can serve shared storage pools and data servers.

The types of Grids and their applications are paving the way towards the next generation of the Internet. The Internet constitutes the network of different networks that are interconnected through some channel and within which entities communicate through some well-defined protocols (HTTP, HTML). As far as this is concerned, the Grid computing infrastructure by perception must provide interoperability among the VOs which share and consume resources from diversified locations, platforms, and languages. The details related to the Grid interoperability protocols and their analyses are out of scope of this chapter and can be found in [1].

VOs and their participants are often geographically distributed, which introduces security issues like authorization and authentication. Indeed, security is one of the major requirements, without security a Grid is incapable for utilizing its own potential. Another essential requirement is the information service capability, which provides VOs with information about participants, resources that are shared, and the services provided. Further use cases and requirements related to the information services and security will be discussed in Chapter 3 and 4.

2.2 Grid Middlewares

The most significant component of the Grid infrastructure is the Grid middleware which hides Grid complexities from resource consumers and providers. The middleware layer encapsulates a transparent, secure, and interdependent mechanism to access resources across inter-institutional environment. This section discusses three Grid middlewares with academic background, namely UNICORE, Globus, and gLite.

2.2.1 UNICORE

The initial UNICORE (Uniform Interface to Computing Resources) project has been funded by the German Ministry of Research and Education (BMBF). UNICORE provides access to distributed computing resources by hiding the complexities behind the client, i.e. what servers are being accessed and where sub-jobs are delegated. From the security perspective UNICORE is based on PKI (Public Key Infrastructure) with X.509 certificates for user and data management.

UNICORE provides means to manage complex jobs that a user can construct within the client, i.e. make composite jobs (called job groups) which form a recursive structure of other jobs. The jobs consist of tasks, i.e. a single unit of action, that can be executed on a single resource or it can be distributed among other resources. The distribution can be e.g. one task that is executed on one resource and another one executed on a second resource, but the final result could be the combination of the output from both tasks which is sent back to job sender. UNICORE handles the temporary data required to execute particular job by managing caches. Users can also construct complex workflows which consist of a large number of tasks and, e.g., represent tasks related to physics which require many conditions or loops.

Among these features UNICORE possesses a variety of functionalities such as single sign-on, meta-computing, resource management and support for legacy jobs. More details can be found at [12].

The UNICORE architecture is composed of three tiers as shown in Figure 2.2.

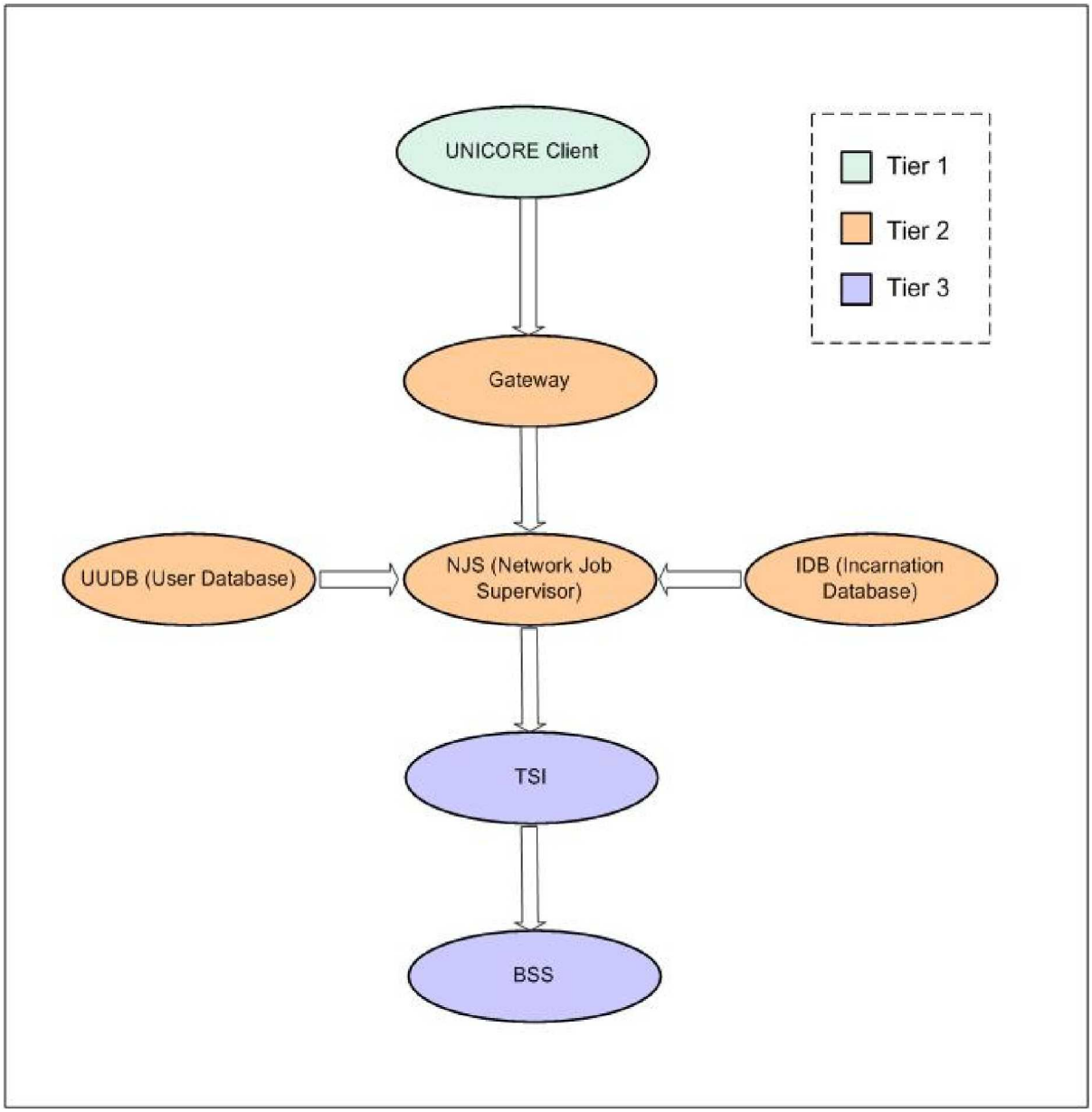

Figure 2.2: Three tier UNICORE Architecture.

From the end user point of view UNICORE has a client-server architecture. The UNICORE Client is a Java-based application used to send requests to a server. The Client prepares a job request in the form of an Abstract Job Object (AJO) which is a serialized Java object that contains the job specification, resource requirements and the task workflow. The AJO is received by the intended Gateway, an entry point to particular network. The Gateway authenticates the AJO and passes it to the targeted NJS (Network Job Supervisor), a server component also written in Java. The NJS manages jobs and user authentication data with the help of the UNICORE User Database (UUDB) and the

Information Database (IDB). The UUDB performs the mapping of X.509 user certificates onto the logins of a user. The IDB contains resource-specific job execution information. The IDB is the repository where a resource owner advertises the specification of resource capabilities and low-level properties. In the course of processing a job, the NJS sends the authenticated job request to the TSI (Target System Interface) layer which physically holds the resources itself. One Gateway contains a set of NJS and TSI servers, this one unit is called USite. And, the grouping which contains one NJS managing a TSI is termed as VSite (multiple TSIs per NJS are possible but not common). This means that a USite contain many VSites.

2.2.2 Globus Toolkit

The Globus Toolkit provides a software development toolkit for developing and managing Grid applications. The essential goal of this toolkit is to provide developers with a development environment and a set of APIs, so that they can exert all efforts on implementing the business logic. The Globus infrastructure is based upon the service-oriented architecture paradigm; it is one of the first working prototypes of the Open Grid Services Architecture (OGSA). In the upcoming section about web services, I will discuss briefly the components, applications, state and underlying specifications of OGSA. From an end user point of view, each Globus component can be accessed through a web service interface; these interfaces are also called *Grid Services*.

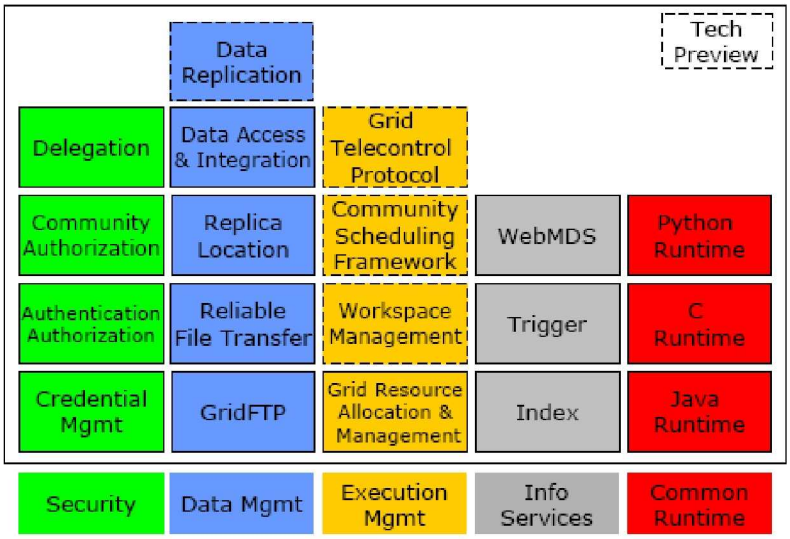

Figure 2.3: Core GT4 components [13].

As already outlined, the Globus architecture is service-oriented. There lies a significant portion of toolkit containing variety of distributed components and services. Globus is loaded with numerous multifaceted modules conforms to an infrastructure that provide every component's access through web services. This service oriented behavior is the most significant and community oriented aspect of Globus. The major functional components that are currently being part of Globus toolkit are Data Management, Security, and Monitoring and Discovery Services, as shown in Figure 2.3.

2.2.3 GLite

GLite is developed by the EGEE (Enabling Grids for E-Science Europe) project, which is funded by the European Commission. Its objective is to provide a robust, deployable, platform-.independent and functional system that offers interoperability with core Grid services like resource access, data management, information collection, authentication & authorization, resource matchmaking and brokering, monitoring and accounting [14]. For further information please consult [15].

2.3 Web Services

Web services offer software a layer that enables interoperability in machine-machine communication over a network [16]. In other words, web services are loosely-coupled distributed components allowing applications to interact through standardized interfaces. These interfaces are based on the XML wrappers by which application can send and receive messages in a standardized format. The major applications of web services are in business to business communication, e.g. in inventory control a system supplier application can check stock demand by calling a specific manufacturer's web service.

A web services exposes its functionality only via a specific format, the details of implementation and the platform on which it is hosted are hidden. In Grid computing this feature is indeed helpful to support communication between heterogeneous resources.

The standard of web services messaging is SOAP. SOAP is a protocol that describes what functionality web service offers, which type of messages it uses and which underlying protocol is employed for application communication. Specifically, SOAP is an XML

document, which has two mandatory parts: the envelope that encapsulates the data and inside that, a body which contains messages. The messages optionally can hold optional headers which contain additional processing information or description about messages. A SOAP document can be transported over many protocols such as FTP, POP3, and SMTP, but more commonly used is HTTP.

Web services are used to develop highly dynamic, heterogeneous and distributed applications. This feature of web services has its raison d'etre for Grid computing middlewares. Therefore, the majority of current Grid projects is based on this technology.

2.4 Service Oriented Architecture

The service-oriented architecture (SOA) is an instance of the composite computing model that eradicates all the barriers of interoperability between heterogeneous applications.
It is further defined as:

> *"The composite computing model is an architecture that uses distributed, discovery-based execution to expose and manage a collection of service-oriented software assets."* [17]

The SOA is a collection of services that are interacting inside a particular application domain. The major aim of SOA is to provide a loose coupling between the service-oriented software agents. Service-orientation here means that the application software is exposing its functionality and its way of accessing them through platform agnostic interface. For applications that are required to communicate with external business applications, there are two options to achieve this goal. The first and unrealistic solution is to develop the required component, which indeed leads at the expense of time and money. The second option is to leverage SOA benefits by using the desired service component that is developed by other vendors through standardized interfaces, e.g. SOAP.

In abstract terms loose coupling between entities is used to reduce the artificial dependencies without altering the real one. A real dependency is a functional dependency of one application on other. An artificial dependency can be considered as a layer on a

real dependency, in principle it helps fulfilling the dependency by using some interface. For example, if you are carrying a charger of mobile phone, what you need to charge your phone is electric power. But to charge, you should have some interface to consume it, i.e. the power plug should be compatible to the electric switch. In this case, the dependency between charger and electric power is the real dependency and the interfacing is the artificial one. We should not alter the real dependency, because it is a service provided and they are specialized in it, but we can alter the interface in such a way that we can use the charger around the world and use the electric power service. So, in software there should be a mechanism so that we can use other applications by means of any standardized interface. In terms of SOA, the interface should expose minimal information about software components. In a web services context, SOA has been used and there are certain standards of exposing interfaces for other applications.

Web services can be elegantly combined with SOA, where web service providers can easily share their services and a consumer can remotely access it. On the other hand service consumers can search and use the desired service. Figure 2.4 depicts the SOA view in web services context, the major components being Service Provider, Service Requestor and Service Broker. The three major operations of SOA are publishing, find or locate, and bind or invoke. In one scenario, the Service Provider publishes the web service information in the WSDL format, a standard define by W3C to expose web services information in XML format. The Service Broker can be realized as UDDI that stores and manages the deployed web services information in its repository. In the event of search it handles complex querying and searching operations. The Service Requestor uses the Service Broker to search for desired services. After having the service information, the Service Requestor directly invokes the service by calling the provider interface methods, a mechanism is called binding. The interaction of the Service Provider's service with the Service Requestor's application is done through SOAP messages.

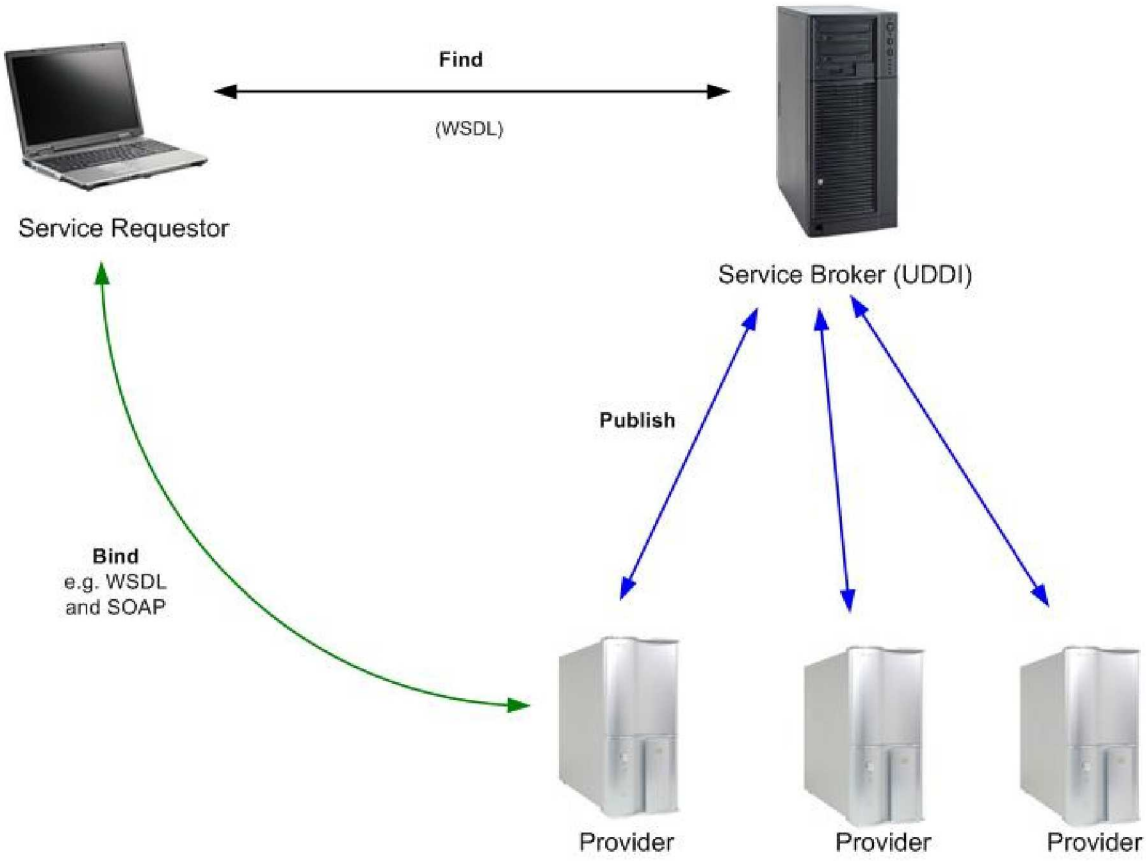

Figure 2.4: An overview of service-oriented architecture [16].

It is believed that SOA can only be realized through web services. Infact, there can be numerous design settings by which SOA can be realized, though web services is adapted by majority of the vendors in SOA context. It can be used in other applications like distributed objects, but for its use there constraints and rules should be defined which accomplish loose coupling and satisfy other primary requirements. The convergence of SOA has attracted Grid computing researchers. Many of the Grid middlewares are either transformed to or based on the SOA environment.

2.5 Service-Oriented Grids

This section focuses on the Grid community advancement towards the standardization of Grid middlewares on service-oriented architecture and web services. This section elaborates the OGSA architecture and Grid services.

2.5.1 Open Grid Services Architecture (OGSA)

The Open Grid Services Architecture is a joint effort of Globus and IBM towards the convergence of Web services and Grid computing. Since the submission of first proposal of OGSA by Foster et al, OGSA has been considered as a de facto standard in the research and development of Grid middlewares.

"A basic premise of OGSA is that everything is represented by a service: a network enabled entity that provides some capability through the exchange of messages. Computational resources, storage resources, networks, programs, databases, and so forth are all services. This adoption of a uniform service oriented model means that all components of the environment are virtual." [18]

The emergence of this architecture appears to be a paradigm shift in Grid computing. The most significant concept behind OGSA is Service Oriented Architecture. There are middlewares that provide implementation of service oriented applications, but there exists some deficiencies in those middlewares. OGSA aims to remove these deficiencies and expected to provide features that are essential to develop and deploy Grid oriented services. Grid functionalities can be exposed via collection of Grid oriented web services. The requirements catering distributed Grid services have evolved this architecture in terms of low level functionality. In this context, OGSA researchers described the involvement of Grid services in distributed environment "we view a Grid as an extensible set of Grid services that may be aggregated in various ways to meet the need of VOs, which themselves can be defined in part by the services that they operate and share" [18].

Grid service is one of the building blocks of OGSA specification and implementation. A Grid service is a web service interface that follows a specific set of conventions, specifically defined as "a (potentially transient) stateful service instance supporting reliable and secure invocation (when required), lifetime management, notification, policy management, credential management, and virtualization" [16]. The most important property of Grid service is statefulness, the property by which these specialized web services can save state during intermediate invocation. OGSA is composed of four tier architecture [19], in which Grid service is placed on the second tier, shown in Figure 2.5.

Physical and Logical Resources - include server, storage, file systems and database managers etc.

Web services plus OGSI extension - In Grid environment every resource is modeled as Grid service. This component encapsulates OGSI extensions which are still being considered as potential services.

OGSA architected services – They are based upon OGSI services and provide functionalities e.g. program execution, data services and core services etc.

Grid Applications - The applications that build upon Grid architected services.

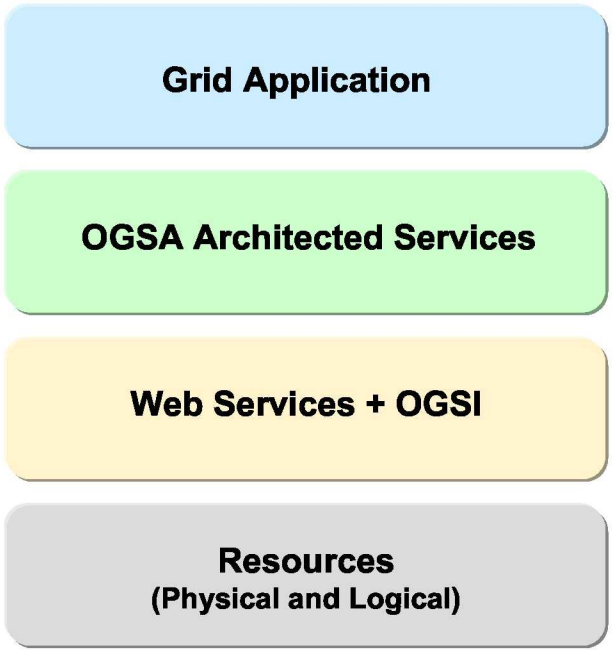

Figure 2.5: Four tiered OGSA architecture [11].

2.5.2 WSRF

The Web Services Resource Framework (WSRF) is an effort initiated by the Globus Alliance and IBM and later contributed by GGF for standardization. Recently WSRF has been accepted by OASIS as a web services standard for modeling and accessing stateful resources. Since the inception of OGSI, it was taken as specification that was dependent on other technologies (web services and XML). As these technologies evolved then it was requirement to reassess and enhance OGSI standard.

In OGSA, web services' core specification (WSRF) is separated from Web Services Notification (WSN). Following are the brief details WSRF and WSN specifications,

WS-ResourceProperties describe a WS-Resource and how service properties can be managed (retrieved and changed).

WS-ResourceLifetime enables a web service client to embed resource lifetime within associated WS-Resource instance.

WS-ServiceGroup describes the aggregation mechanism of services that are desired to be part of a particular group.

WS-BaseFault represents a specification of service faults and errors.

WS-Notification allows services to publish/subscribe for particular event, e.g. service data being changed or updated. WS-Notification provides an asynchronous model of notification.

2.6 Grid Information Schemas

2.6.1 Introduction

An information schema is a template for structuring entities into one model. In the context of Grids, the information schema presents a viable solution for the management of resources and other entities by representing them in correct and unambiguous hierarchy, as described in Section 4.2. Such a schema, e.g., can support a resource broker in choosing the best resource on the behalf of user request or it provides information to a job scheduler for efficient scheduling and monitoring. Other scenario could be for example some middle ware that stores the meta-information of user authorization for a particular resource or how long a user will have access to particular resource(s).

This document discusses CIM and GLUE schemas in general and will emphasize on each model with respect to Grid-oriented concepts.

2.6.2 GLUE

GLUE, which stands for Grid Laboratory Uniform Environment, is an information model that aims to provide interoperability among different Grid vendors. GLUE is a collaborative effort that was initiated by EU-DataTAG [21], US-iVDGL [22] projects and later on EGEE [23], Globus [3], NorduGrid [24] and LCG [25] joined the process of development and standardization. Its objective is to define a common information model to represent Grid resources so that Grid middlewares can use it in their information system's implementation. GLUE is currently used by GT4's MDS information service; therefore some attributes are specialized to this particular implementation. In order to standardize GLUE, other vendors are in the process of adaptation but it leads to model requiring extensive concepts. The following section describes the GLUE v1.2 schema's base entities that represent hardware and software resources.

Core Entities

The core entities of this schema contain classes that are the basis of this model. These entities are abstract classes by which further specialization orchestrates in the form of concrete classes. In the current specification the core entity is Site, which stores all the service and resources information. As shown in Figure 2.6, site is aggregated by service, computing and storage element. By principle of resource concept and SOA, the computing and storage element should not directly be the aggregation of Site; logically they have to be encapsulated within a service class because they are both concrete services within a Grid environment. Concerning this, the specification will revise change in the upcoming major version. Figure 2.6 below shows the core classes of the GLUE schema, their details are given in the following sections.

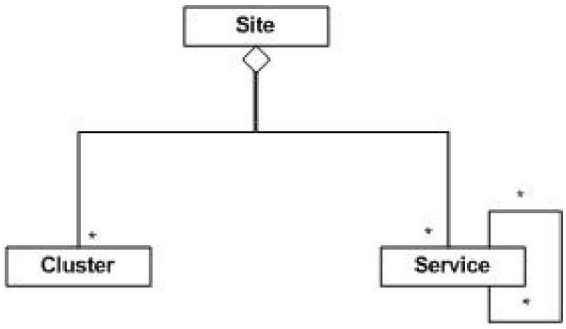

Figure 2.6: Glue schema's core classes [8].

Site

The Site represents an administrative view which encapsulates data regarding a group of people and resources related to any part or the whole of an organization. This class is a template which holds information with respect to user administration and geographical location. In terms of UNICORE its instance could store the information about a USite.

Service

Indeed the growing demand of web services in the area of Grid computing introduces the need for managing information related to the Grid services. The **Service** class is responsible for storing services information offered by the Grid. The service could be either WSRF or WS-I complaint. This class provides a fine grained view of the web service structure such as meta-data, service status and functional description.

Cluster

GLUE provides the abstraction of computing resources through the **Cluster** class. Additional to meta-information and resource description this class' aggregated entities manage information related to access, control and reservation of computing resources. A Cluster is a template of heterogeneous (computers may exist in same cluster but they can have different configuration) physical resources (hosts or nodes or computers) within a computing element. The diagram below shows the hierarchy of cluster classes.

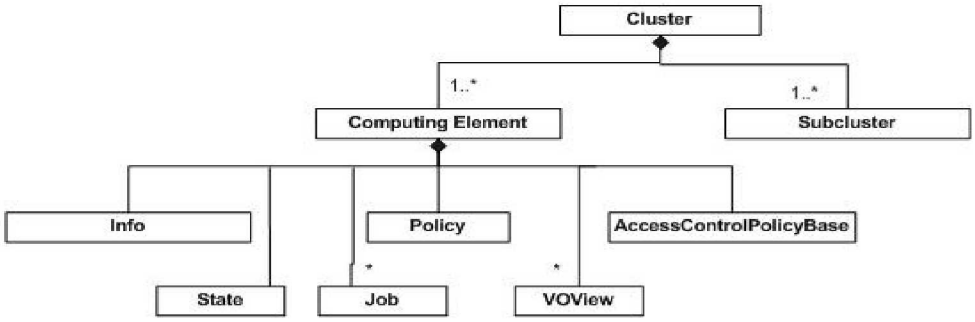

Figure 2.7: Cluster directly aggregates ComputingElement and Subcluster [8].

Computing Element

A **ComputingElement (CE)** models a queue which manages computing resources. Specifically, it manages information related to the current state, access policy, the group of authorized users and their accounting policies, and the status of running jobs.

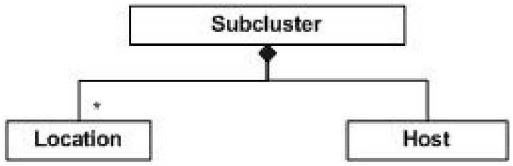

Figure 2.8: Subcluster aggregates Location and Host entities [8].

2.6.2.6 Subcluster

The Subcluster entity models homogeneous physical resources. It contains the structure of machines that offer an execution environment for jobs. The two major aggregated classes are Host and Location, as shown in Figure 2.8. Host stores data related to the machine configuration and Location models the location of software application.

Storage Element

GLUE represents the abstraction of storage resources. It is capable to accumulate storage resource data from simple disk servers to massive storage systems. The schema also focuses on management issues such as data access, space reservation and quota management. In this model the **StorageElement (SE)** is an abstract class which stores all the information related to storage services and provides a flexible way to handle access and control policies information. The section below details the access and control protocol.

Access Protocol

The Access Protocol allows modeling the type of file transfer mechanisms. The mechanism could be Gridftp, rfio, and file (NFS or AFS) which follows POSIX standards. Moreover, the Access Protocol also describes the capability of each of the file transfer protocols.

Control Protocol

The Control Protocol describes the storage control and management functions. It models the elements of locking mechanism like pinning and space reservation.

2.6.3 CIM

The Common Information Model (CIM) is an information model presented by the DMTF consortium to model business, logical and physical computing entities for application and system-oriented standardization. It is further defined as;

"CIM is an information model, a conceptual view of the managed environment, that attempts to unify and extend the existing instrumentation and management standards using object-oriented constructs and design." [26]

The model relies on object-oriented methodologies. The main objective of CIM is to unify other models (e.g. SNMP, DMI, CMIP, etc.) rather than devising a new model. Therefore it is a source of opportunity for other application and system development vendors that not only they can use this model but apply extension and customization according to their own needs. The extension capability comes through object-oriented design and the use of tools which smoothly suppresses the "flat" or non object-oriented nature of data models. Following are the motivations of using CIM as an information model in the light of object-oriented concepts.

Abstraction and classification: Abstraction is however a solution to deal with application complexity [27]. In CIM classes are modeled from abstract to concrete classes. Though it seems to be a long hierarchy but at the end it reaps a fruit of extensibility and customization.

Object inheritance: It is an important concern while designing an object oriented model. The inherited classes (subclasses) can possess additional behavior with respect to their parent classes (super classes). In this account great care has to be taken to prevent the fallacies and hazards of inheritance. The significant inheritance problems and solutions are described in [28].

Ability to depict dependencies: In information model different relationship between objects introduces complex dependencies, and the complexity becomes more visible after its realization. By using an object-oriented approach the dependencies can be handled in a simplistic manner, moreover, this also depends upon the schema's meta-model. CIM provides mechanisms to conceptualize complex dependencies with its consistent, well-d defined hierarchy and the structure of inherited meta-model. Thus CIM handles the complex concept associations with object-oriented principles, thereby overcoming design ambiguities.

The CIM model is encapsulated in a hierarchical fashion and is categorized into a set of specifications and unified schemata. CIM model is structured upon the meta-model (model about model) which defines the syntax and set of rules to describe a unified schema. The meta-model is based on the UML notation so it can represent common elements of a data model in an open and standardized format. The common elements are classes, properties, methods, indications and associations etc

The CIM schema is an instance of the meta-model which holds the set of models to represent managed elements. There are three main classes of schemata which cleanly separate responsibilities in a hierarchical manner keeping consistent abstractions. These three abstract models are: Core, Common and Extension.

Core Model

The Core model contains a group of abstract classes from all domains. This model is ground to all common models where abstract classes, associations, properties and methods are structured. This is the model which originates the notion of model scalability and extension. It means the common models are integrated such that if any modification or update is applied to core classes then there is a high probability the related common models will change. The CIM hierarchy of classes is rooting from the Core model to Common models and Extension models.

Common Models

The Common models are based on the Core model which contains the classes from multiple domains independent of any particular technology or implementation. The Common model contains Systems, Applications, Devices, etc.

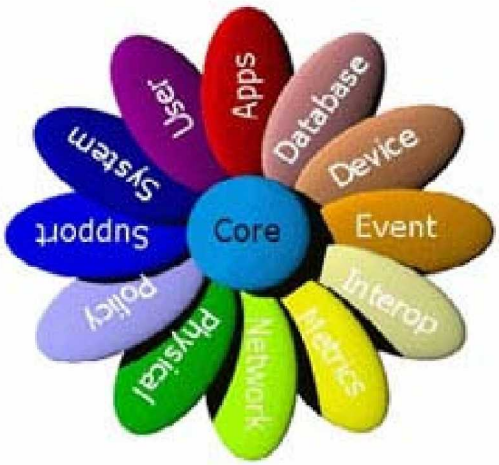

Figure 2.9: Relationship between CIM's Core and Common models [29].

As shown in Figure 2.9, there are 12 common models in total. The section below will discuss only those Common models which are significant to the Grid information model.

System Model

The System model is an important group of classes to model computer systems consisting of parts forming a whole unit. The major concept of system is coming from the Core model and the abstractions are being transformed to concrete classes. The main class of the System model is the computer system in which all parts are accumulating. These parts reside under the boundaries of the System model and they are file systems and files, operating systems, and services. A short description of each part is given below:

System Element: Computer system and its subclasses.
File Elements: Describe File Systems, Files, and Directories etc.
Operating System: Define operating system's relationship and classes.
Services and Service Access Points: Represent computer system services and access points.
Time: Represent timing information (time-zone) for computer systems.

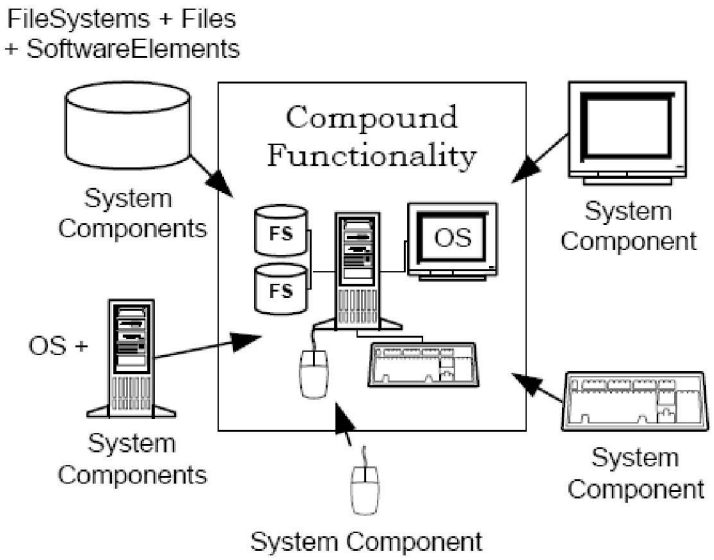

Figure 2.10: Server containing sub components [29].

An example could be a server as a computer system that contains many hardware and software components. These components could have different operating systems, local and remote file systems, and system and application services with access points, as shown in Figure 2.10.

Device Model

CIM's Device model formalizes the functionality of hardware components, their configuration and state. The functionalities capture the behavior and any of their operation that affects the state based upon the configuration. In the Device model every hardware component is a logical device. The logical device can aggregate other logical devices through class association. For example, a Printer device contains a Memory device. CIM integrates logical devices with the System model to form a computer system with the representation of hardware devices. The term logical device comes from CIM_LogicalDevice which represents an abstract class of the Device model.

Application Model

The CIM Application model describes necessary information to represent software and entities related to application management. This model caters the diversified information which ranges from basic software characteristics to the environmental properties where it will be deployed and executed. Additionally, it provides flexibility for application

vendors to extend the model from every dimension of concept with respect to association and inheritance.

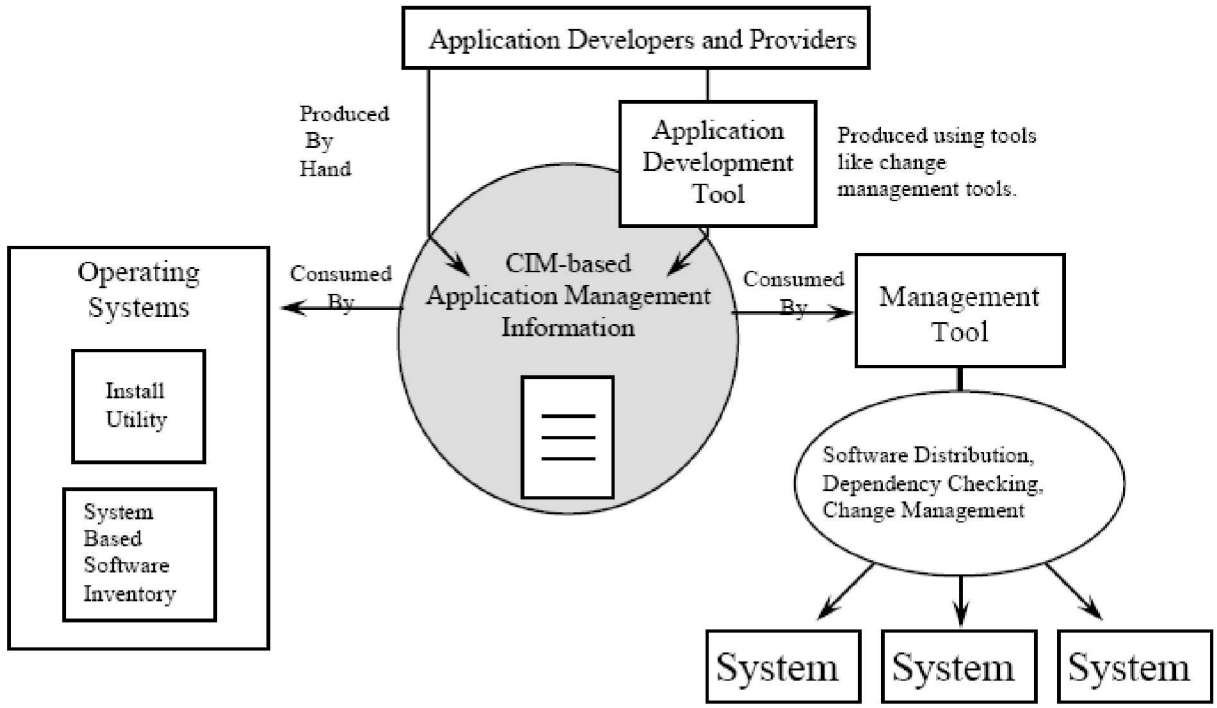

Figure 2.11: Managed Application example [30].

Imagine an application being installed under an operating system. The software vendor packages software by indirectly creating an instance of the Application model containing information pertaining to the configuration, management, required components and directories, and target operating system. Eventually, at the time of installation, the software information will be consumed by the operating system and it will probe all the software conditions provided by the vendor. A more vivid picture can be seen in Figure 2.11. The presence of this model provides interoperability support between applications and operating systems, and from the user point of view there will be less interaction required while configuring and installing the software. In Grid applications, CIM based information services are greatly influenced by its application and system model. For instance a resource provider shares a computing resource with some software application installed. In this scenario the information service maps the computing and software information by creating the instances of the System and Application models.

In contrast to the System, the Application model focuses on the details related to the software product, software element, and application system. These are further described as follows;

A Software Product represents a collection of software features that form one unit. This unit is formed according to certain agreements between supplier and consumer, which may be significant with respect to licensing, support and warrantee issues.

A Software Feature encapsulates a collection of software elements that exhibit some functionality or role which is important to the user or consumer of software rather than applications that needs to know what parameters constituted the product.

A Software Element contains the collection of files and other details related to particular platform. Normally software features manage this information.

An Application System holds the collection of software features that manage an independent unit containing particular business functions.

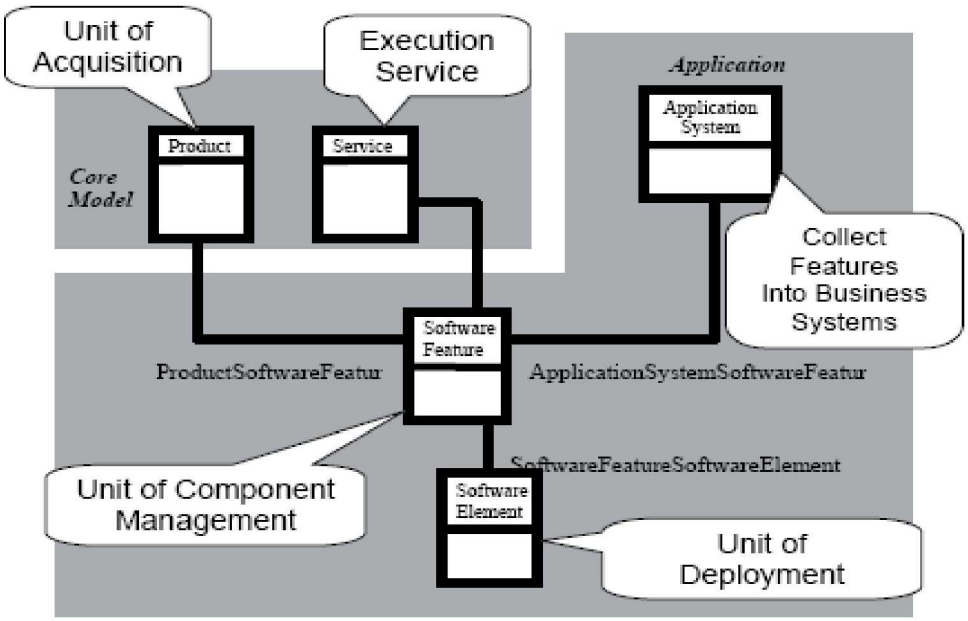

Figure 2.12: Abstract classes in Application Model [30].

Figure 2.12 depicts the summary of responsibilities and relationships of the four discussed concepts. It shows how the classes from different domain models are

interlinked and dependent, i.e. the Application model's Software Feature concept depends on the Service and Product class of the Core model.

Extension Model

The Extension model provides extension points to an application or vendor-specific set of classes. For example, operating system vendors may define particular classes and associations with respect to their customized application and infrastructure. Specifically, this model is targeted towards user roles such as system analysts and architects who design enterprise and Grid applications and may frequently apply extensions to achieve system-oriented customization and application requirements. In the architecture chapter, more emphasis will be given upon the Common and Extension models with respect to a Grid information service.

Chapter 3

Application Scenarios and Requirements Analysis

In the design and implementation of any system, we always need well analyzed requirements which should foster the system in all development phases. Before presenting the architecture, this chapter exploits information service requirements as there are application scenarios, functional and non-functional requirements.

One of the main responsibilities of a Grid environment is to manage and consolidate resources in one infrastructure. The management of resources in one network requires a centralized controlled mechanism to track resources and curb the state of Grid that is to integrate various resources virtually at one place. In order to analyze these issues, the proposed information service will play a vital role in a dynamic Grid environment.

In order to analyze uses cases an exemplary situation has been taken. For instance an organization manages the cluster resources. These clusters are such that they can leave and enter the Grid dynamically. Even individual workstations can interleave and rejoin at any moment. The core functionality of system is therefore dynamic resource discovery. The next section explains the functionality including use case model diagram and description.

3.1 Dynamic Resource Discovery

An information service is an independent component which handles resource (data and service) information. Two essential sub-components are registration and lookup.

A registration module allows resource providers to seamlessly register resource data and meta-data. The information is published to a location where certain interfaces provide access to resource consumers and brokers. This module allows resource providers to register resource in an interoperable and standardized format. This standardization of the

resource data in the repository holds certain significance in terms of application integration with other open standard technologies.

Once the registry assimilates the resource data and meta-information supplied by the resource provider, that data will be available for resource consumers via lookup interfaces. The dynamic resource discovery module facilitates resource consumer to search for legitimate resources serving inside Grid. The returned resource data can be for example used by resource managers or schedulers who send requests to particular resources in order to achieve desired tasks.

In summary, performing searches for information is an essential step in the course of resource acquisition.

3.2 System Actors

In use case modeling, actors are external entities that interact with the system. The information service has three actors to which it exposes functionality.

The *Resource Provider* is an entity which advertises information about the resources it wants to share within the Grid environment. As shown in Figure 3.1, the Resource Provider is interacting with different use cases in the process of resource registration and update events. In our scenario, the resource provider can be information sensors and clusters e.g. Ganglia and Hawkeye.

The *Resource Consumer* is an entity which aims at using a resource. For resource consumption, the actor performs functions related to lookup and notification. For instance, a Resource Consumer can be job scheduler or a resource manager.

The *Administrator* can have access to all the functions of the system and has the authority to change and update any resource data at any time. As depicted in Figure 3.2, the Administrator inherits the privileges of Resource Provider and Resource Consumer. The rationale behind introducing this actor is that the application will always be less susceptible to ambiguities, as there is a centralized user control. Actually, the

Administrator will have seldom communication with system (until any failure occurs) once the information service is functioning. The majority of his responsibilities are exploited during administrative actions like maintenance and upgrade. Usually administration use cases are targeted to this actor.

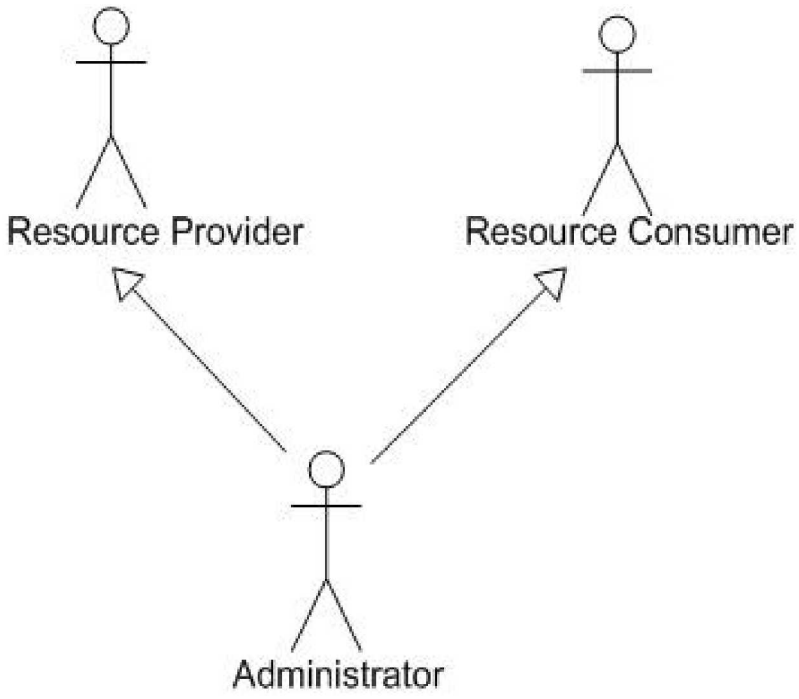

Figure 3.1: Administrator inherits the role of Resource Provider and Resource Consumer

3.3 Use Case Model

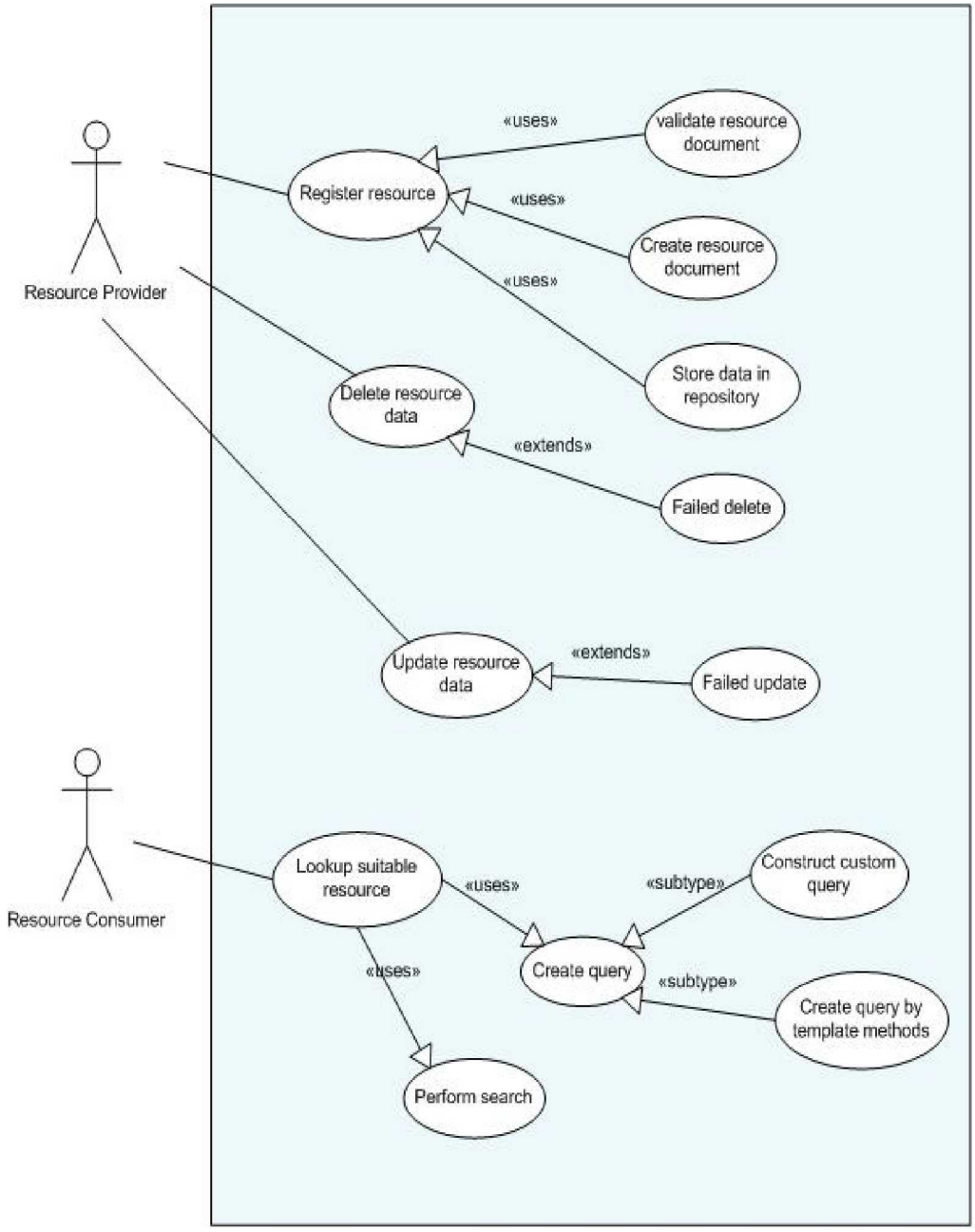

Figure 3.2: Application use cases for resource registration and lookup.

In the Figure 3.2 four main use cases are identified for dynamic discovery. These use cases are directly invoked by users/actors and front use case calls to other use cases that are transparent to the user.

1) *Register resource* is the main interface for a Resource Provider through which he can register his resource to the information registry. This use case further aggregates sub-

steps before the final registration is accomplished. The very first step is that the Resource Provider prepares resource information that is compliant with the system information schema. Once it is prepared and validated by the registry, it is stored in the registry.

2) *Lookup resource* provides an interface to the Resource Consumer to search for resources in the registry. This use case is the generalization of two use cases: template query and custom query. The template query use case allows a consumer to use the queries provided by the information service and specify parameters for the query. The second use case for creating custom queries gives enough freedom to the user such that he can construct his own query for a search. As a result of this use case, the registry should output consistent and non-ambiguous results to the consumer.

3) The *update resource* interface is only invoked by the Resource Provider. It allows him to update resource information. Before the execution of this use case, the Resource Provider should provide legitimate information that he owns the resource. As a result of this use case the registry contains updated resource information. Presumably, the system validates and authorizes resource data for secure modification.

4) *Delete resource* is another interface to the Resource Provider. The system allows the resource provider, who shares the resource, to remove the resource from the registry. A t this level the requirement of removing a resource is abstract and will be fine grained through analysis and design phases. After the successful execution of this use case the resource entry is deleted from the registry.

3.4 Functional and Non-Functional Requirements

The previous section focused on use cases of an information service. On the basis of these scenarios the requirements are identified which then support one step towards the realization of the information service. The requirements which are used to capture core functionality of the system are called functional requirements. Functional requirements are of most significant concern to the implementation of every system and ensure stake holders' satisfaction. There are other requirements, which however have not necessarily

to be fulfilled and thus enhance the system with extra features like extensibility, performance and reliability. These are termed as non-functional requirements. The section below details the functional and non-functional requirements in the Grid information service context.

Functional Requirement 1: Information schema

In Grids, heterogeneous and distributed resources are shared and consumed by different entities. For efficient resource management, a Grid environment needs a complete, structured and extensible data model to structure resource data. Indeed it is expected that any resources belong to different categories and may possess numerous characteristics. This data model should be consistent and capable of handling complex resource data. Moreover the data model must provide room for extensibility for novel hardware and software resources.

Functional Requirement 2: Dynamic discovery of volatile resource information

Grid environments change dynamically with resources joining and leaving at any point in time. This behavior leads to a requirement of a dynamic registry which shall be capable of managing the addition and deletion of resource data and services dynamically. Other motivation to offer this functionality would be that the state of the resource may change frequently, e.g. a resource may go offline or resume after temporary failure. Hence this requirement caters the volatile behavior of resources.

Functional Requirement 3: User defined resource lookup

A Grid system is expected to contain resources which possess state and lifetime. For any Grid environment it is essential to offer functionality by which a user can share and consume resources. In order to consume resources for any task, a Grid must offer an efficient query mechanism by which users can freely search for particular resources. There may be hundreds or thousands of resources which results in a huge amount of resource-related meta-data. Since the consumer does not need all information of every resource at one time, the lookup or query service provides the user with the freedom to fetch the right and correct information based on his search.

Functional Requirement 4: User can track his own shared resource

An information service allows resource providers to share resources through exposing their resource data. It is a normal case that a resource provider often desires to track his own resources to check the correctness of information. This requirement will expose an interface to the provider through which his resources can be tracked.

Non-functional Requirement 1: Interoperability

In a Grid environment, there are multiple sub-systems inter-related to each other, hence in the execution of Grid-enabled tasks these systems are highly dependent. In this case when systems are highly dependent the interoperability appears to be a significant factor. In order to leverage this environment, the subsystems need to contact information service for discovering resources. Due to this fact, the information service is expected to be linked with every middleware component and to formulate this structure, a careful design strategy should be implemented which should be capable of resolving future integration problems. For instance Grid middleware entities or subsystems that may be a part of a middleware are: scheduler, resource broker, and execution framework. Further details pertaining to related components will be given in the next chapter.

Non-functional Requirement 2: Performance

As the information service will be handling extensive requests, the user expects to receive quick response for any query. It may annoy users if the system takes too long to reply and lead to negative user feedback. In a dynamic Grid environment, an information service shall provide accurate information with high throughput and low response time.

Non-functional Requirement 3: Extensibility

The release of every major milestone of an application should not undermine the further development and future enhancements. And, it is not like that an application has fulfilled all user requirements and will not undergo any further changes. In order to prevent this situation, an application should provide a set of interfaces that allow extensions and further development. The information service shall expose programmatic interfaces in every level of the architecture in such a way that it can be extended for future enhancement. This requirement is very much prominent in the case of customizability and application integration.

Non-functional Requirement 4: Reliability

In order to provide the user with a reliable application, there has to be a mechanism which handles fault tolerance and high availability. As the resource information is precious and the system depends on it, the information service should be highly available. Replication is also one of the important requirements of this thesis. A separate chapter is dedicated to the architecture of a replicated information service.

Non-functional Requirement 6: Transparency

As a Grid environment is composed of inherently complex systems with extensive architectures, users cannot be privileged with it until and unless there is simple entry point of access. End users are not Grid experts who can tolerate the complexity and setup configuration by their own, but they rather need a simple interface to achieve registration and lookup. An information service is also a complex unit of Grid application. It should provide transparent access such that system complexity could be minimized. Additionally, transparency will be beneficial while service testing and application integration.

Chapter 4

Service-Oriented Grid Information Service Architecture

The basis of any complex application is a well defined set of requirements and the analysis which leads to the design process. In the previous chapter, the resource discovery is identified as core functionality for which functional and non-functional requirements were identified. The next step is to take the requirements one step further to the architecture level and put more emphasis on the fine grained details of the implementation. Formulating an effective design and an architectural model is one of the primary and significant building blocks in the realization of any system. This chapter emphasizes upon the architectural design of the Grid information services. With this outline, the chapter starts with a description of the development environment and the frameworks required to implement the architecture. Following that, the Grid information model section will describe the resource model for the information service. In the last section the information service's architecture and system decomposition will be discussed.

4.1 Development Environments

To realize the technical aspects a set of tools/technologies are identified for the development and deployment of Grid information service. This section focuses on the brief description of these tools, and the rationale behind their adaptation.

4.1.1 Apache Axis 2

Axis2 is an Open Source SOAP framework maintained by the Apache Software Foundation (ASF). ASF serves as one of the portals of tools and technologies with motivation to uplift the awareness and promotion of open source frameworks and application programming interfaces. For example, commonly used tools for development like Log4J, Ant, Rampart, or Xerces are maintained by ASF.

Axis2 provides a framework for the development and deployment of web services. Application developers can easily develop applications by leveraging the Axis2's simple and adaptable framework. Not only there is a room for web services developers but for the developers who are implementing web services specifications. It would require pages to elaborate the Axis2 model, therefore a concise description of important features [31] is stated below.

Programming Model: This model facilitates the framework with an open standard client API based on XML, and provides support for all WSDL constructs. It can support the synchronous and asynchronous type of service communication with respect to web services' clients. This model also provides flexibility to specify message exchange patterns while coding server-side logic.

Support for WS-Specifications: Besides web services core specification, Axis2 provides room for other WS specifications by which web services developers can integrate their implementation with the framework. In this way Axis2 can be customized for multifarious applications. Currently, among the supported specifications are WS-Addressing, WS-Policy, SAAJ, and WS-Security. This feature of Axis2 however fulfills one requirement of the Grid information service's replication framework design and more details will be described in the next chapter.

Multiple Transports: Axis2's communication infrastructure provides multiple alternatives in terms of SOAP message transports between web service client and server. In Axis2 for example, a web service developer has the freedom to specify the SOAP transport over HTTP, JMS, SMTP, etc.

Multiple Data bindings: In a web services environment, high-level communication is done through SOAP which is an XML-based protocol for communication In this context data binding proved to be a valuable tool which simplifies the overhead of parsing and reconstruction incoming and outgoing messages by generating data objects of input and output messages used in a particular web service. Web services are platform and language-independent where a server may be written in Java and a client is developed in C#, and may be another is written in Perl. Thus data binding is there to simplify the message exchange between server and client entities. There are various tools for XML

data binding such as XML Beans, ADB or JAXB. Axis2 provides data binding support for ADB (Axis Data Binding), XML Beans, JibX, etc.

4.1.2 Berkeley DB XML

The Berkeley DBXML is a native Open Source XML database developed by Sleepy Cat software which is now part of Oracle. The DBXML stores XML documents as XML nodes and plain files. XQuery is used as a tool for document retrieval. The database also manages document level meta-data with an efficient indexing mechanism. As shown in Figure 5.1, DBXML sits on the Berkeley DB layer, by inheriting all features it provides. The features include distributed transactional support with ACID properties. As far as security feature is concerned, Berkeley DB stores documents AES encrypted in the storage media. Additionally, the DBXML can store both XML and non-XML documents into the repository.

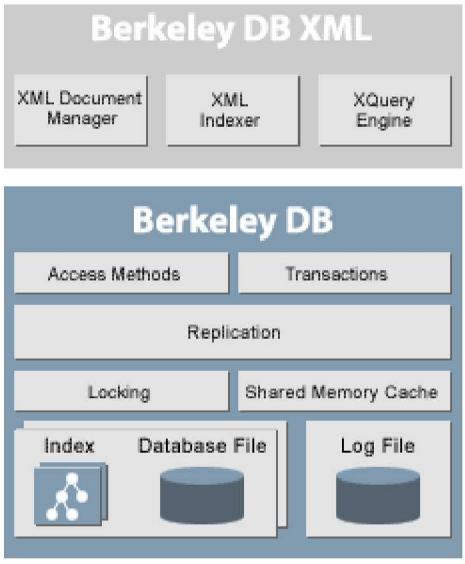

Figure 4.1 [32]: Berkeley DB XML architecture.

In summary the DBXML possesses core XML-oriented functionalities such as support of XQuery [33], document indexing for fast searching, a document parser, full support of XML Schema validation and automatic naming mechanism.

4.1.3 Eclipse Framework

Eclipse is an Open Source IDE and framework for platform and language agnostic software development. This project is an initiative by IBM and its alliance. For application developers it is convenient in terms of using plug-ins and writing application code. Framework developers can develop plug-ins for advancement in terms of Eclipse integration with other tools. Eclipse is a Java-based framework and a variety of open source and commercial plug-ins exist, which highly facilitate application development. Notwithstanding the Eclipse framework also offers development environments for other languages like HTML, C/C++, Perl, XML, etc.

The Grid information service is based on Java using Eclipse as the development platform. Hence plug-ins used for the information service's development are Log4E, CVS, and Web Developer Tools.

4.1.4 Ganglia

Ganglia is an initiative of the University of California, Berkeley Millennium project. It embodies a distributed and scalable monitoring tool to exploit fine grained monitoring information from high performance clusters and Grids. It extracts monitoring, hardware and software information from computing resources such as clusters in order to support job and resource scheduling in computing environments. Ganglia is considered to be state-of-the-art since its inception because of the granularity of information it provides and support for monitoring up to 2000 nodes. In the architecture of the Grid information service, Ganglia acts as a resource owner gets resources' information and monitoring status. Some significant static/dynamic parameters Ganglia captures are processor type, memory usage, CPU load, available disk space, etc.

4.2 Grid Information Model

The Grid information service consolidates a set of components customized for Grid middlewares and contains backend data structured based on an information model. The information model (also known as schema) represents Grid resources and services. It is an essential part of any Grid middleware and most of the time the Grid execution framework

and other systems are highly dependent on its structure; therefore it should guaranty a simple, correct and consistent hierarchy of concepts. Here correctness and consistency implies to the associations and categorization of concepts which prevents them to result in ambiguous and inconsistent behavior while instantiating.

There are three main reasons to construct and formalize an information schema: i) to manage resource information consistently and with integrity, ii) to advertise resource information from the resource provider's perspective, and iii) to specify resource requirements that are produced by resource consumers such as job schedulers or resource managers.

In order to derive the Grid information schema, the major goal is to formalize a unified data model which suits especially the requirements of the UNICORE resource model in terms of semantic and syntactic behavior. Actually the UNICORE middleware is one of the core stakeholders of the Grid information service. This section describes firstly the fisheye view of UNICORE resource model and secondly the transformation of the UNICORE model into DMTF's CIM model. Finally, this section will conclude with the implementation of information model into an XML schema using design patterns and best practices.

4.2.1 UNICORE Resource Model

As described in the second chapter, UNICORE has a client-server architecture where the client communicates transparently with the server without knowing the underlying complexity of the architecture. The client can submit a job by specifying resource requirements to fulfill its request in the same way a resource owner can advertise resources and describe its capabilities. Both, resource owner and consumer (client) communication, is done through the AJO (Abstract Job Object). The NJS is the core middleware component which receives requests from the Gateway and provides an interface to handle the process of resource advertisement and consumption for job execution and management. Mainly the advertised resource information is stored in the IDB (Incarnation Database) using proprietary text based description. In the event of a job request, the AJO is validated against the IDB entry.

If enough resources are available and capable to handle the request, the job will be started. In summary, all the consumer and producer resource description is based upon the

UNICORE resource model. In order to analyze and transform the UNICORE resource model into CIM, it necessary to discuss its general view and categorization.

The base class of all the concepts is a Resource class which represents all type of resources down from its hierarchy. Generally there are four types of resources as shown in Figure 4.2.

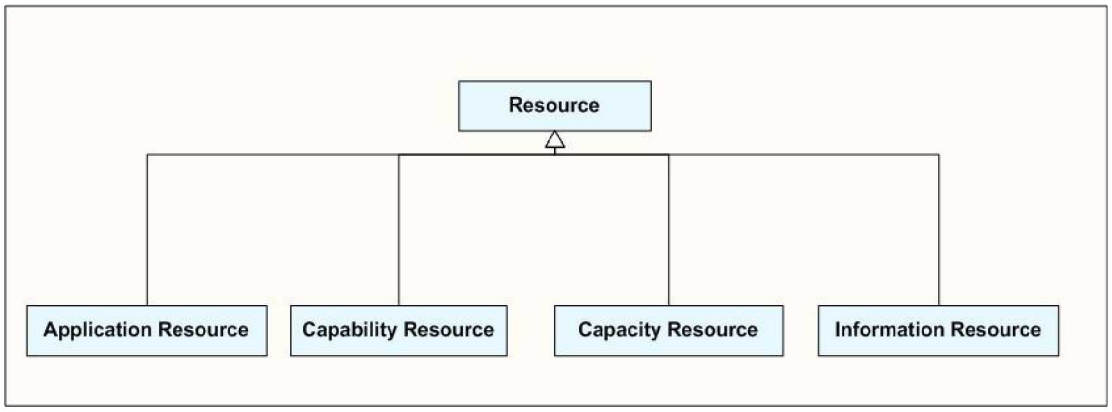

Figure 4.2: UNICORE resource model showing all concepts inherited from Resource class.

Application Resource is a class which portrays the resource information pertaining to the application characteristics that is irrespective to the computational and data-specific attributes. This class has two sub-concepts: Tasks and Threads. A Task details the number of unit of tasks a particular application is running. And Threads defines its containment within each Task. Thus Threads and Tasks form a hierarchical structure, in which Tasks are composite of Threads.

Capability Resource exploits the concept of capabilities that any resource offers. This class can be used by a resource owner while advertising the resource characteristics and by a consumer while issuing the resource request. A resource consumer can inquire about resource capabilities through the set of query functions, e.g. GetResourceDescription (returns particular resource data) or GetJobs (retrieves all jobs on particular VSite). The subclasses of Capability Resource are Host, Priority, Software, etc. A resource advertiser and requestor uses the same classes to specify capabilities (but within a different context).

Capacity Resource is a concept describing resource properties with respect to measurable quantity in the event of its sharing and acquisition. Concretely, this class is essential for both resource provider and consumer. It allows resource providers to specify the number of resource units to share, whereas the consumer specifies the requirement in numbers. For instance, the scenario could be thata resource owner advertises its resource by describing the number of nodes, processors and memory. The main attributes of interest to resource provider are max, min and default units whereas the requestor specifies only the request attribute of this class. Capacity Resource is a parent class of all UNICORE resources such as Processor, Node, Host, Storage, and Memory, etc.

Information Resource specifically aims to capture meta-information. It has two specializations: Numeric and Textual Information. Normally any resource model cannot be considered as complete because no one can predict how many new resource types can be part of a Grid environment. This also applies to new resource attributes. The UNICORE resource model deals with Information Resource. Usually this is useful from the resource owner perspective who can describe some additional information and sometimes helpful for consumers while searching for particular resource.

4.2.2 UNICORE - CIM Transformation

The discussion above emphasized upon the UNICORE information model describing the concept's hierarchy in abstract terms. As types of resources and Grid entities day by day are increasing, this model should cope with the problem of extensibility and scalability. In order to overcome this deficiency, there are two distinct solutions: to formalize a new model from scratch or to adopt an existing open standard information model which caters the UNICORE model requirements. Concerning this thesis, the second solution has been selected, i.e. to use an existing open standard model in UNICORE perspective. Actually, there are three major concerns with respect to the UNICORE model which led to the transformation of this model into a new open standard model.

Extensibility: UNICORE model does not provide a clear way to extend concepts and associations for future entities with concrete details. Th emajor reason could be that the model does not comply to object-oriented design principles.

Interoperability: In a Grid environment applications do not exist in isolation but need interfaces to communicate with each other through some means of transparent communication. This can be achieved through interoperability. By keeping in view its significance with respect to UNICORE model, one should not underestimate future integration with other enterprise and distributed applications. The major requirement to provide interoperability is that the model should provide an interface which hides internal complexity and through which other Grid middlewares can seamlessly access its underlying instances.

Model covering limited resource information: The UNICORE model handles resource information in two major aspects: resource capabilities and its measurable aspect. The missing information is about the description of resources itself. For example missing information is about processor containing attributes such as processor family (Intel, AMD), CPU load, clock speed and number of bit,s etc. This information however plays an important role for resource sharing and consumption.

After considering the above problems, the proposed solution is to choose an existing schema which not only represents UNICORE concepts but also allows the modeling of other general computing and software entities fostering the needs of other Grid and application middlewares. In this context DMTF's CIM schema has been taken to express UNICORE-related information.

UNICORE-CIM Mapping

The actual transformation is done through the mapping of UNICORE classes into the CIM model. The majority of the new UNICORE mapped classes extends the CIM common models. Hence, these classes form a layer of CIM extensions which supposedly allows customization with respect to any domain and technology. In our case UNICORE is the target extension of CIM-extension model.

UNICORE	CIM	CIM Model
AlternativeUSpace	UCR_USpace	Extension from System
Home	UCR_Home	Extension from System
Processor	UCR_ProcessorResource	Extension from Device
Memory	UCR_MemoryResource	Extension from Device
PathedStorage	UCR_FileStorageResource	Extension from System
Node	UCR_NodeResource	Extension from System
SoftwareResource	CIM_SoftwareElement	Part of Application model

Table 4.1: Mapping UNICORE classes into CIM model.

Table 4.1 shows the transformation of UNICORE classes into the CIM model. The first column represents some of the UNICORE classes and the second column describes the corresponding derived and extended CIM classes. The third column elaborates the model category (System, Device or Application) in which mapping classes are residing. Due to lack of space all classes cannot be represented in the above shown table .

It is important to mention that the CIM model can map service-oriented entities, thus the UNICORE-CIM model implicitly inherits all its features. For example, if the data has to be modeled for a service-oriented architecture scenario (consider a web service which encapsulates computing resources) and accessed through a service access point using any communication protocol for message transport, then this type of information can easily be modeled using UNICORE-CIM model. By keeping in view the current trend of service-oriented Grid computing this model hopefully helps to build scalable Grid applications.

4.2.3 XML Schema of UNICORE-CIM Model

Any information model cannot be useful until there is a way to utilize it in a simple way. The UNICORE-CIM model is currently implemented as an XML Schema [34] which is an open standard technology to represent and define XML based data models. However, an XML schema realizes object-oriented constructs such as associations, inheritance, and constraints. A subtle alternative to this technology is the DTD schema, which describes a model as a list element in the form of a non-XML representation, not capable to model

object-oriented constructs. By considering the complexity and requirements of UNICORE-CIM model, DTD will not be a wise choice for the implementation.

To deal with complexity and future changes the implementation prototype of the UNICORE-CIM model has been developed using XML schema's best design practices. There are two major reasons to apply best practices: i) to avoid any inconsistencies and complexities if the model grows or changes and ii) the use of design practices is like a template solution of recurring problems [35]. The second argument is more convincing in terms of software development, especially in the object design phase [27]. Consequently, the adequate use of patterns leads to an extensible and scalable model. The UNICORE-CIM model is purely of object-oriented nature as it is based on CIM. Hence, the use of design practices is the perfect choice to implement it as XML Schema. The best practices taken from [36, 37], are applied to the UNICORE-CIM model, and are listed below:

- *Use Namespaces for name conflicts.*
- *Use Substitution Group for element inheritance.*
- *Use Complex Content extension for type inheritance.*
- *Use wildcards to specify multiplicity for class associations and relationship.*

Figure 4.3 depicts the real use of the above mentioned design patterns in the UNICORE-CIM model.

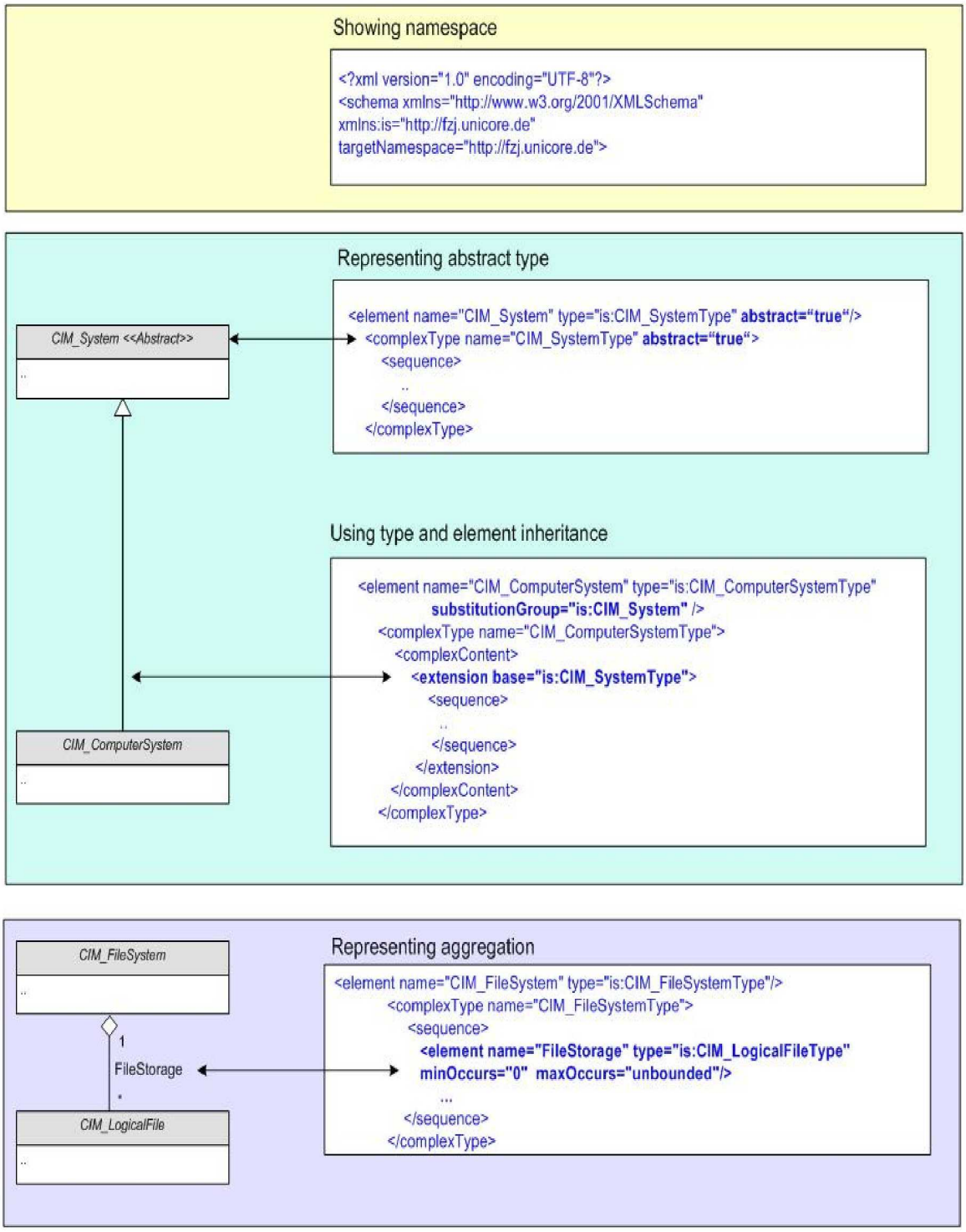

Figure 4.3: Showing the application of XML Schema design patterns in the UNICORE-CIM model.

4.3 Grid Information Service: Architecture and Design

This section focuses on the design and architectural details of the Grid information system. In order to expose the underlying architectural components, this section starts describing the Grid information service in terms of service-oriented architecture and then each of the architectural entities will be explored.

As outlined above, the architecture of the Grid information service is based upon service-oriented architectural concepts that are normally followed by web service applications. The respective picture can be seen in Figure 4.4.

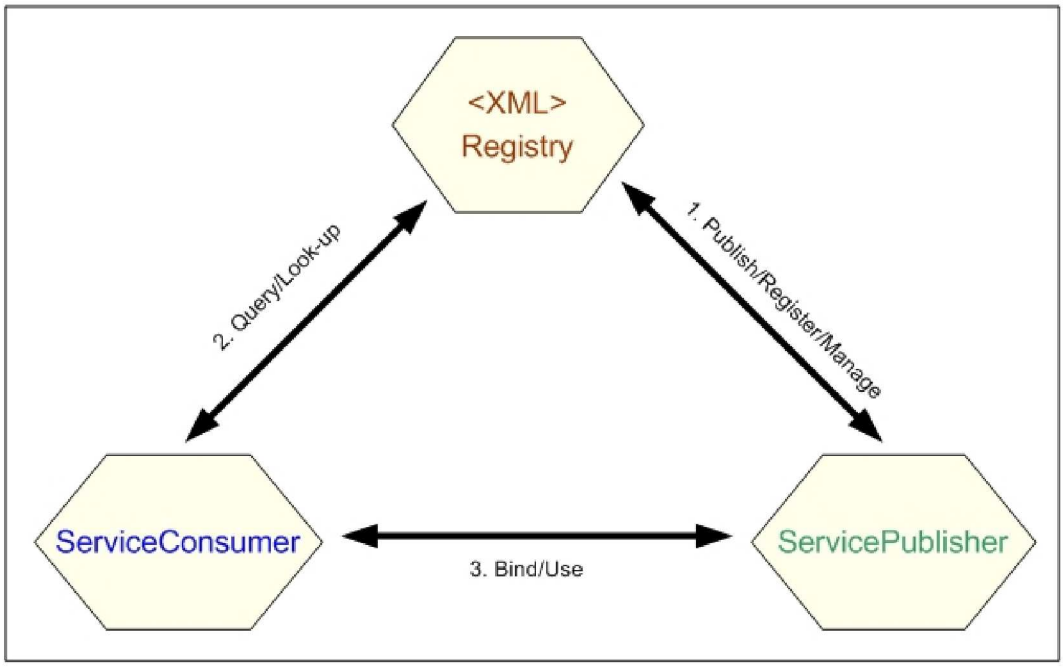

Figure 4.4: Service-oriented Grid information service's architecture.

The three primary steps described in Figure 4.4 are:

1. *Publish/Register/Manage:* The ServicePublisher advertises the services it wants to share on the network by specifying service information with compliance to a particular schema. This information is maintained by an XML-based registry. At any point of time, the ServicePublisher may update or remove the resource information.

2. *Query/Lookup:* The ServiceConsumer may request for desired services published on the registry. It sends requests to the registry in the form of a query. The registry then performs a search on the repository and returns the set of endpoint references (EPR) of the available services. Normally search results are based on the search criteria provided by the ServiceConsumer

.

3. *Bind/Use:* As soon as the ServiceConsumer gets the EPR of the intended service it tries to contact the ServicePublisher in order to utilize service functionality.

Figure 4.4 depicts the coarse-grained view of Grid information service's components. The fine-grained details of each component will be envisaged in upcoming section.

4.3.1 ServicePublisher / Provider

The ServicePublisher is an entity which publishes structured resource information using an XML-based (SOAP) communication protocol. While publishing, the ServicePublisher sends resource details in the form of an XML document that should be compliant with the UNICORE-CIM schema. At the registry side there is a specialized component which only handles requests of the ServicePublisher by exposing particular business functions. After successful registration of a resource document, the registry attaches a unique identifier to document and then sends to publisher. This identifier can help the publisher to update and renew its resources. Practically known information providers like Ganglia and Hawkeye [38] are systems acting as ServicePublishers to promulgate cluster information (see Figure 4.5). The ServicePublisher comprises four subsystems that processes the publishing and registration of service information, they are described as follows:

- Information Provider
- Schema Translator
- Information Detector
- Information Service Delegate

The ***Information Provider*** is an external entity that provides detailed information about resources and publishes their capabilities and characteristics on the information service. It

plays a vital role of getting low-level information from clusters and even a hierarchy of clusters. Here "low-level" corresponds to the information regarding installed hardware and software components, their current state (number of processes running, CPU load, available memory, etc.). Among them some of the parameters are real time, i.e. continuously changing so the information provider efficiently extracts this information using low-level scripts. As said before, the Information Provider is an external entity, but it is an integral part of the publisher. Please refer to Figure 4.5, where the Ganglia information provider is residing on the top suppling information to the ServicePublisher.

The ***Information Detector*** senses dynamically changing information supplied from the Information Provider. Specifically, the Information Provider deployed software agents on hosts that continuously extract changing resource parameters, e.g. the CPU load and the number of running processes. There are potentially innumerous attributes sensed by the information provider, but the information detector only filters interesting attributes specific to the information service.

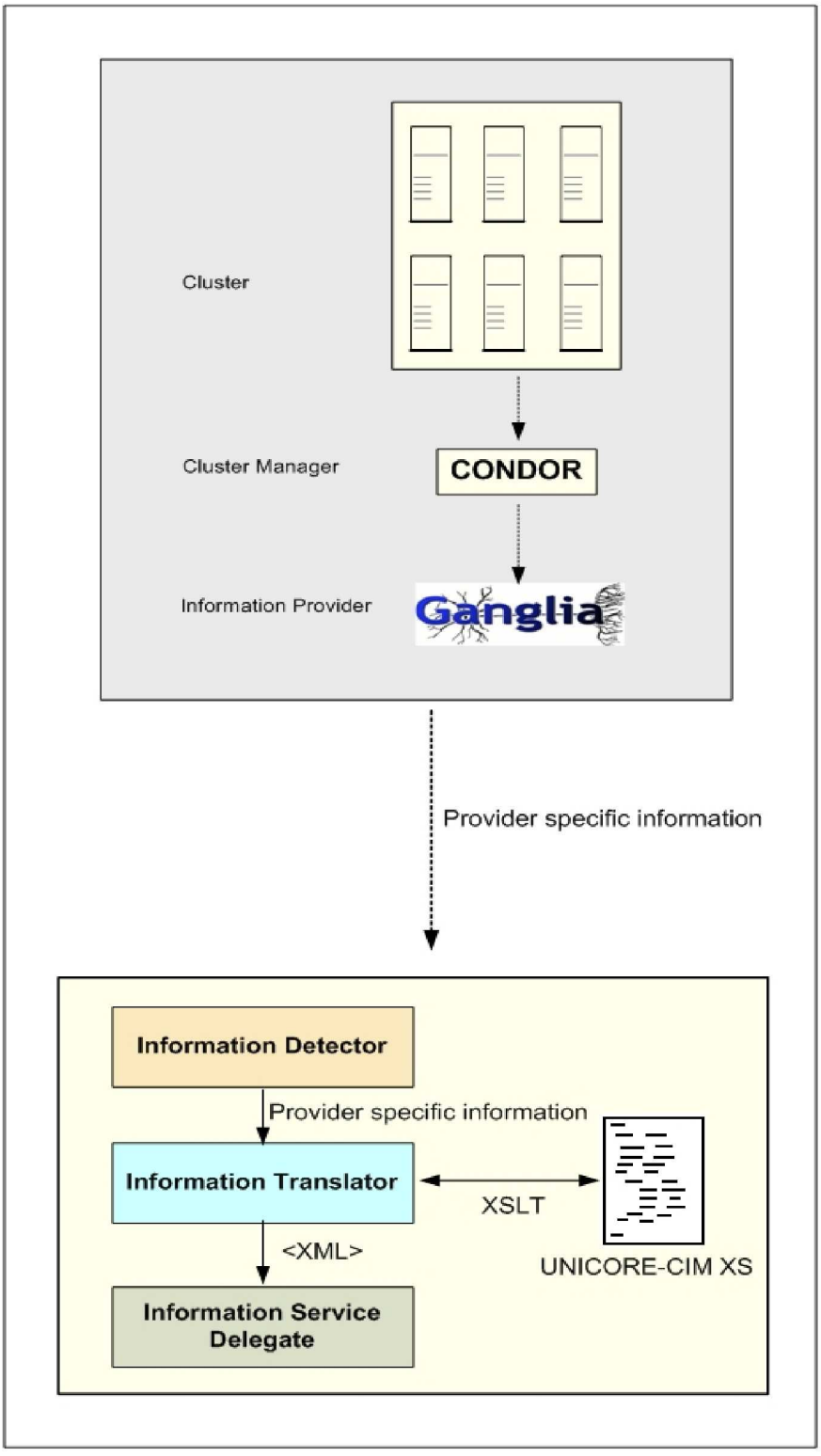

Figure 4.5: A cohesive architecture of the ServicePublisher containing subsystems fetching and manipulating resource information from a cluster (represented by Ganglia).

The ***Schema Translator*** gets resource information from the detector and transforms it into the UNICORE-CIM format. As the information service's registry is based upon the UNICORE-CIM format, any data entry has to follow this format. Currently the registry only supports Ganglia as a sample information source (see Figure 4.5). Normally Ganglia outputs resource information in its proprietary format. It may be possible that in future, in case the registry has to interface with other providers, the Schema Translator should be capable to transform into the targeted schema. Due to this requirement of scalability, it is based on strategy design pattern [35]. For example, the translator may interface Condor pools represented by Hawkeye.

The ***Information Service Delegate*** follows the business delegate J2EE design pattern [39]. It is responsible for reducing the coupling between business and presentation tier details. In this case, the business tier is exposed as the Grid information service which can be accessed via delegate. The main functionality of the delegate is to reduce remote function calls, to transparently locate remote service and to wrap the information service's business logic in a way that the complexity is not exposed to the ServicePublisher. This component is an end point of the ServicePublisher and delegate of information to the service registry. A respective information service delegate is also present at the consumer side.

4.3.2 ServiceConsumer / Requester

The resources which are advertised or published on the registry are eventually to be utilized by the respective ServiceConsumers. The ServiceConsumer is an entity which performs lookup by calling the respective information service's methods. In response, the information service outputs the list of service EPR's based on the specified search criteria. These EPRs provide a means of access to the respective services. Figure 4.6 pictures the detailed view of ServiceConsumer along with its sub-components' interactions and also shows how the ServiceConsumer interacts with users like a job scheduler, resource manager, or web client.

The ServiceConsumer consists of the following six components:

The ***Information Service Delegate*** has the same responsibility as the respective component at the ServicePublisher. The only difference is here that it is located inside

the ServiceConsumer and delegating consumer-oriented calls (queries) (and resource publishing functions) to the information service.

The ***Consumer Information Model*** draws two distinct types of object representations: the objects wrapping query request and search result objects. This is also a J2EE design pattern known as data value objects or transfer objects [40], normally used in the design of distributed systems. Especially they are objects used to communicate data to/from one tier to another in the form of light-weight objects encapsulating model instances.

The ***Consumer Information Controller*** is in charge of processing user requests and it prepares the value objects required to call information service. For example if a user's request is to fetch services with CPU load below some threshold then the controller prepares specific query value objects using the information model API and forwards it to information service delegate, which then sends it to the particular information service interface.

The ***Consumer Information Subject*** is a view entity in terms of the MVC architecture. It generates responses based on the type of request or user. Specifically, it behaves according to the type of user (as shown in Figure 4.6) which can be a web client, a scheduler, or a resource broker. For instance, if a user is considered to be as a web client (internet browser) then the information subject would be a JSP page generating customized HTML page depending upon the response of the information service. In case of a scheduler or resource broker, the information subject is a set of Java classes generating output which is compatible to their environment (XML or plain text file format).

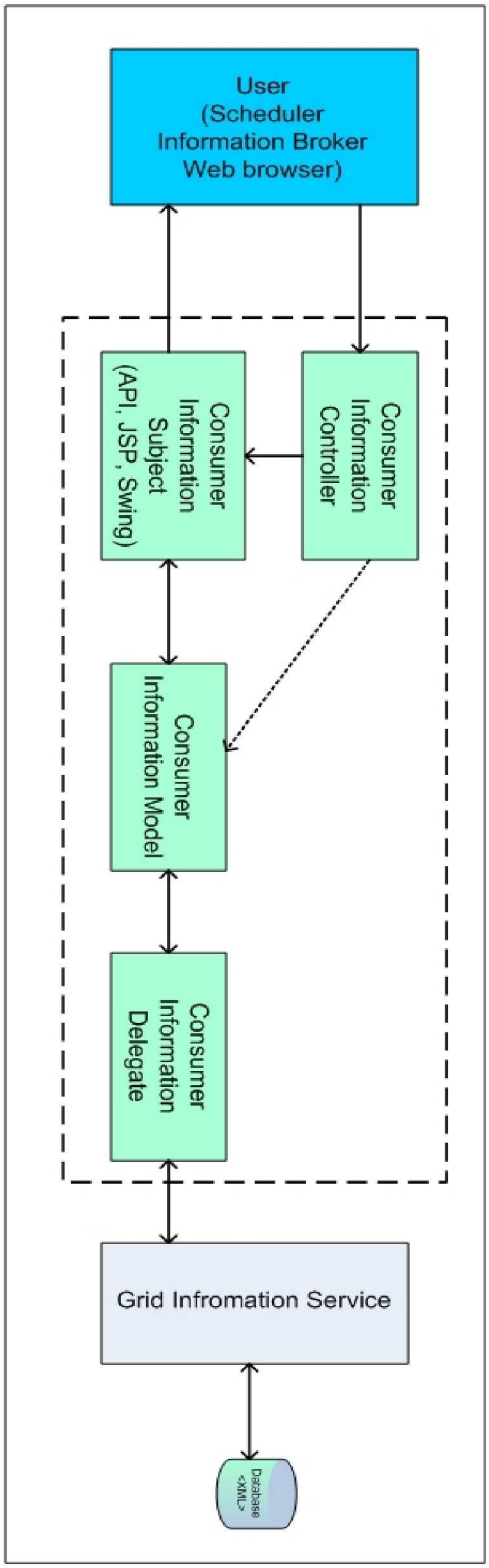

Figure 4.6: MVC based ServiceConsumer architecture.

The ***User*** acts as a front end of ServiceConsumer and may be a Web client, a job scheduler or a resource broker. The Web client is simply an internet browser-based GUI

through which users can query the details of resources. A job scheduler is an external software agent that schedules jobs based on the resource/service availability. A resource broker does match making for optimum resource available based on the supplied criteria, hence in this concern it often queries the information service. Excluding the Web client, broker and scheduler have frequent interaction with Consumer Subject in case of querying real time monitoring data.

4.3.3 Grid Information Service

The Grid information service encapsulates the information model and allows real use of information schema. All the effort of requirements gathering was aiming towards the realization of this component. In abstract terms, the Grid information service is composed of two parts: Service Discovery and Replication Framework. The details concerning the replication framework will be outlined in the next chapter and service discovery will be discussed here.

Service Discovery is based on a layered architecture providing a clean separation of layers and subsystems. Its two significant layers are business logic and data integration layer, as shown in Figure 4.7. The business logic layer corresponds to two core business functions: Service Registration and Service Lookup. Service Registration allows creating, updating and removing service information from the database. Service Registration is customized for the ServicePublisher, whereas lookup allows the ServiceConsumer to perform search for desired resources.

While registering or adding a service, the registration component receives resource data in the UNICORE-CIM format, in response it returns a unique identifier. In the course of updating service information, the registration component can only perform updates on the existing resources, and by using an assigned identifier. Service removal will follow the same interaction of events.

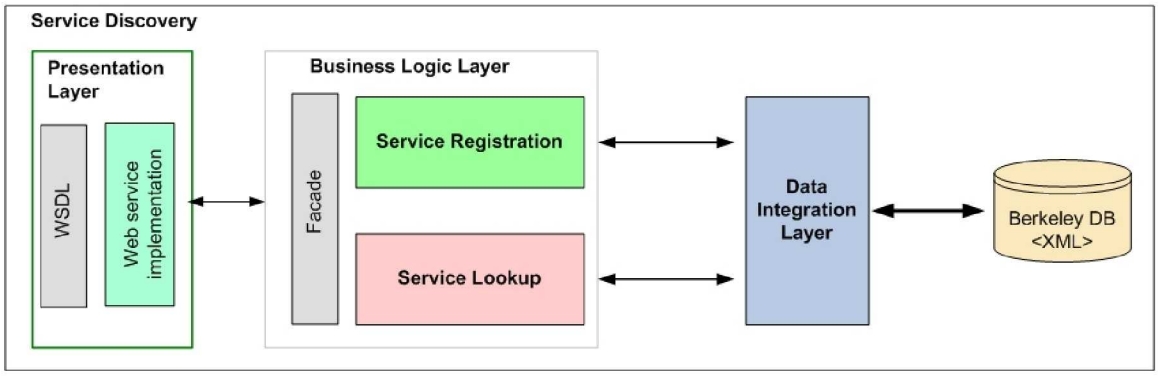

Figure 4.7: A layered architecture of the Service Discovery subsystem.

Service lookup is solely targeted for searching functions. It sends queries in the form of data value objects to the database layer. Furthermore it converts objects into the XQuery format. Lookup provides both template and free form query functions; the template function takes certain parameters and prepares the respective query on the user behalf, whereas the free form functions gives a user an ample freedom to specify any query or projection with arbitrary parameters in the form of an XQuery format. Template functions appear to be handy functions for users, who can specify the general criteria. A user, for example, wants information about services representing a cluster of five nodes, and since the information service already provides this method, in that case the user only has to specify the number of nodes and a conditional equality operator to template function.

The data integration layer acts as a bridge between the database and the business logic layer by hiding low-level database functions. It encapsulates ground-level database functions, such as database connectivity, and environment handling functions. It also contains data value objects for waving queries between the business layer and the database. Moreover it manages the local transaction and recovery mechanism to ensure ACID properties against any database operation.

The presentation layer is more concerned towards the exposition of information service methods through web services. More details covering web services technical aspects have been discussed in Chapter 2. Figure 4.7 depicts the presentation layer showing WSDL for service interface specification and besides that the corresponding implementation providing realization of service. For further details on web services architecture and technologies please refer to [41].

Chapter 5

Replication Infrastructure for a Reliable Grid Information Services

Replication is a basic infrastructure to provide fault tolerance, and highly available and scalable applications [42]. Distributed systems, which are expected to handle massive client requests and deliver responses without failure and delays, are indeed requiring redundancy in terms of software and hardware resources. In this thesis, replication is taken as second auxiliary layer of the core information service, which handles redundancy issues in the context of fault tolerance and availability. This chapter will describe in detail the replication framework of the Grid Information Service. After briefly describing the motivating factors for replication, the requirements regarding the GIS are introduced followed by a proposal for a multi-tier replication architectural model.

Performance Enhancements

In Grid computing, services can scale into different locations implying that it is not necessary that they reside in one location. Consider the following scenario: there is a central data source which is frequently accessed by Grid services which want to retrieve data from it. In this case it is possible that it may take a long time for the data source to respond to service requests. This will eventually lead to low performance of the applications utilizing the Grid services. To overcome performance problems and to make data access efficient, one of the solutions is to replicate the centralized data. By then data requests from remote services will be handled by different replicas, thereby resulting in less delays and faster response times. Consequently, this will enhance the performance of the applications.

Increased Availability

One of the frequent user's requirements concerning services like registries or information services is high availability, ideally such services are permanantly up and running. In other words applications should be able to rely on such service despite of any failures and data inconsistency. High availability is further defined as

"The proportion of time for which the service is accessible with reasonable response times should be close to 100%." [42]

The mere fact is that high availability factor is measured in number of 9's (99.999%) some where close to 100%. Hence, to achieve optimal availability, replication is the only solution to guarantee highly available services which can handle client requests through data and process redundancy.

In a Grid context, it is likely that at any moment processes are pipelined, e.g. one task is initiated while another could be in execution and maybe several others are waiting. And, also what resources are being consumed by those tasks. If the information service goes down due to any failure (network partition, hardware or software failure) then all the application state related to tasks and resources will be lost. After recovery, every process has to be restarted, state has to be recovered and it is still cannot be assured that every process initiated by clients is stored in their machines. Replication will solve this problem by replicating the hardware/software components of the information service.

Fault Tolerance

A service which exhibits high availability and better performance surely does not meet all clients' criteria. Another criterion to take into account is that data or output from a service should be accurate. Specifically, when a service returns data, the client is also concerned with the correctness of the data. The message which has been delivered to the client should contain legitimate content instead of an arbitrary message, since such failure will result in ambiguous process states. Replication can solve the problem of failures and guarantee reliability of client invocation by first analyzing the response for correctness and then return it to the client. Different fault tolerance algorithms in the replication setting exist [42, 43]. How information services can be made fault tolerant and reliable will be discussed in the next section.

5.1 Requirements for Grid Information Service Replication

Consistency

As discussed in the previous chapter, an information service should use a backend repository to handle and manage service data. In order to adopt replication, the repository will also be replicated across different hosts using a replication framework, thus providing up-to-date and consistent data to guarantee the consistent application state in case of failure and recovery.

Replication Transparency

In order to govern replication, an application possesses a complex architecture consisting of different machines connected through different network topologies. Clients of information services should not be exposed with this complexity and the arrangement of information service nodes. Replication should be transparent to clients. Another important requirement is failover transparency, that is, if any of the hosts failed within the replication group then it should update its configuration dynamically without letting clients know about this problem and it also should handle pending information service requests and responses which were blocked due to the problem.

Transactional Control

The replication mechanism shall provide means to guarantee ACID properties in a replication network and to ensure execution of any transaction at-most once. Usually the current replication frameworks assure that the data is up-to-date in the repository, but they lack the management of redundant transactions. The proposed replication framework should provide an "exactly once" transaction mechanism.

Fault Detection and Recovery

If any of the information service replicas experiences any level of hardware and software failure, the replication framework should detect and remove it from the replication group. Hence, if the failed replica recovers after maintenance or updation, the replication framework should dynamically add it to the list of replicas.

Replicate Multi-tier Entities

Managing replication in distributed systems with a multi-tier architecture ensures highly available and reliable applications. Conventionally, replication architectures have been developed for single tier. Today most of the applications are trying to cope with multi-tier architectures by introducing replication on every tier with considerable factors of high availability and fault tolerance. It is, however, complex to handle all this complexity at once, indeed it is not trivial to design multi-tier replication; notwithstanding it is an essential requirement of an information service's replication framework.

As discussed in the previous chapter, a pragmatic approach towards an information service would be based upon a two-tier architecture comprising a web service layer and a data base layer. The web service layer acts as a presentation layer providing a seamless programmatic interface to the information service's functionalities. The web service layer will be deployed on any SOAP container or web server for handling client requests. To this end, imagine a scenario where a web service is replicated and uses a reliable and centralized database on the backend. Then there is less complexity since only the web service tier is replicated, making the task to design such architecture less challenging. Indeed there are research prototypes, commercial off the shelf, and custom applications that are providing web service replication. For instance [44-47] emphasize upon the application/web server replication but without considering any backend tier reliability. These solutions will still not match with our perspective and requirements. In a second scenario consider a centralized application server uses a replicated database. This still is trivial enough to rectify the problems in the design of such replication. Again, there has been a lot of research done in the area of database replication, for example [48-50] consider only database replication without any assumption of application/web server dependencies. Both approaches shown represent a trade-off and both do not fulfill the requirements listed above. Therefore an architecture should be formalized which encompasses multi-tier replication considering entities of all tiers candidates for reliable, consistent and fail-safe design. As the ADAPT framework [51] has done research in this area and presents tangible multi-tier replication solutions, the replication architecture of the Grid information service is based upon the ADAPT framework's replication approach.

5.2 An Architectural View of Replicated Information Service

As discussed earlier, a multi-tier replication is required to support high reliability and fault tolerance on all tiers. This section proposes an abstract replication architecture for the web service and database tier in an information service context. This section starts with a high level view and then discusses the architectural entities in detail.

To decide which type of architecture (loosely or tightly coupled) is best-suited for the information service, an analysis of existing strategies is required. Following the discussion in [52], where a strong rationale of taking loosely-coupled techniques is presented, the replication of the information service is based upon the loosely-coupled paradigm.

Specifically loosely-coupled construes that each tier uses a separate replication algorithm. In that case independence of the individual tier's replication can be gained, but both replication tiers should provide error free coordination for consistent application state even in the case of failures and crashes at any time. Therefore at architectural level a set of flexible interfaces is required to enforce coordination and consensus on each of the replication phases [53].

Figure 5.1 depicts a diagrammatic view of a loosely-coupled information service architecture showing how the different tiers handle replication independently.

The architecture is based on the existing replication approaches. In the very first step a client requests load balancer which virtualizes the WS tier transparently to the client and chooses the best server where to float the requests. The WS tier is deployed as a set of replicas interacting with the DB tier. The independent replication at both tiers is devised by considering independent failover scenarios. Assume the WS tier manages its own replication algorithm and relies on a centralized database. If any of the serving WS replicas crashes the next server will take over and it will not affect the database tier as it is being considered as centralized. In the same way imagine the DB replica tier connected to a reliable and centralized WS container. The failure of any DB replica will be transparent to the WS tier as it pretends to be centralized, and reliable. In fact the corresponding tier's

replication framework will be responsible for providing transparent failover without letting know its client. Hence, this scenario depicts the replication between both tiers in case of failures and crashes are transparent. This proposition holds inc case both tiers' algorithms are correctly designed and provide reliable communication between each other.

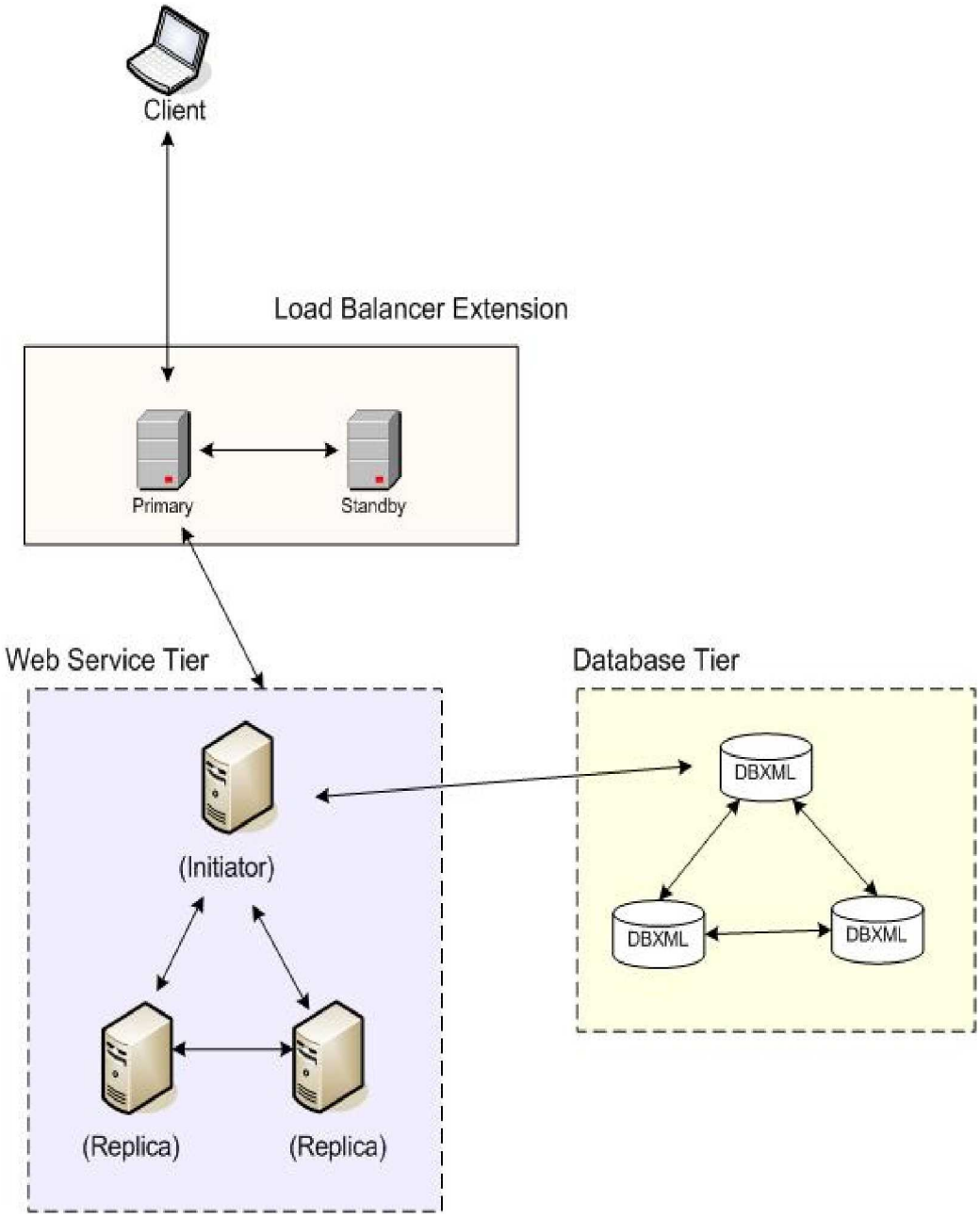

Figure 5.1: Distributed Architecture of Independent Replication

To derive an entirely new algorithm for WS and DB replication is out of scope in this thesis. Indeed, suitable approaches exist to enable replication which provides highly

available, scalable, and fault tolerant services. Regarding the DB tier, replicas avoid more than one commitment of the same transaction, thereby allowing "exactly once" semantics [52]; the same holds for the WS-tier. The replication at WS and DB level follows the primary-backup approach, i.e. the client request will be first executed by any primary replica and after the execution the updated state will be propagated to all replicas.

For performance reasons the load balancer is responsible for submitting requests to the server that offers the shortest response time. For one particular request the chosen server, known as Initiator in Figure 5.1 will be the primary server for the whole session. In this way, there can be many primaries serving distinct clients. There is no such scenario that the load balancer sends the same request to more than one server, but exceptions are possible in the case of failures. During the crash of any client's WS primary, a new WS replica will take over the request, with the failover being transparent to the client. The failover transparency is also managed by load balancer, thus reflecting as a façade between client and WS tier.

The information service places the DB tier below the WS layer (as shown in Figure 5.1). The DB replication follows the eager primary copy replication paradigm [53], in which every update operation is firstly performed by the primary or master DB, and then propagated to all other replicas. After getting an acknowledgement from all the replicas, the primary DB replies to the WS replica which initiated the request. In case of a failure of the (primary) DB, a new primary DB will be assigned. If the (primary) DB fails, the WS tier loses the connection to that primary and will establish the connection to newly elected primary by replication group mechanisms. The consensus and agreement is also a privilege of the group communication primitives [50] that handle the election of a new primary and maintain the dynamicity of the replication group in case a replica fails or a new one joins.

5.2.1 Web Service Replication

The replication on the WS tier is based on WS-Replication [54], a recent specification for highly available web services. WS-Replication ensures the reliability and scalability at the WS tier. It has to be noted that WS-Replication does not facilitate the DB replication and yet it does not have any component that can be extended to allow such functionality. This drawback does not undermine the potential of the WS-Replication architecture; instead

introducing some tuning in the architecture with DB replication aware components can provide its missing features.

In Figure 5.2 two WS replicas are connected via a transport interface and each replica contains replication and group communication services offered by WS-Replication. Additionally, for the requirement of integration with the DB replication tier, the DBRC (Database Replication Controller) interface acts as a remote database driver to the primary database.

The two major components of WS-Replication are the replication framework and the group communication system. The replication framework is responsible for the execution of replication algorithms and manages the preparation of web service invocations for the group communication among other replicas. The group communication system is responsible for providing totally ordered atomic, reliable messaging, and failure detection of replicas. Furthermore, it handles the replica membership service which is capable of managing new replicas joining and faulty ones leaving the group. Below each of the architecture's components are described briefly.

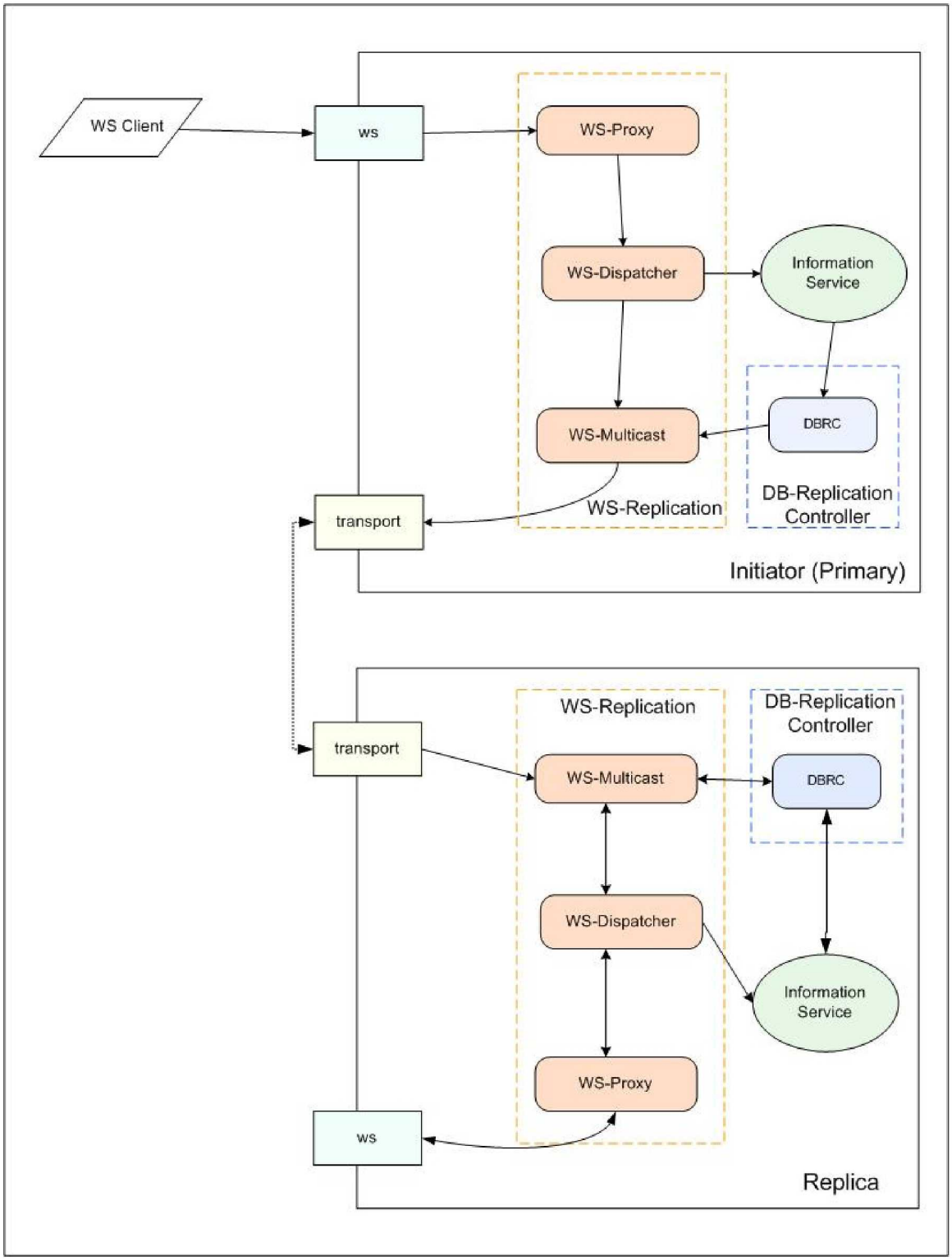

Figure 5.2: WS-Replication architecture with DB-Replication Controller

WS-Proxy is a proxy generator for each of the web service methods. It intercepts the client request via the **ws** interface (the rectangle box in Figure 5.2) and sends proxies to the dispatcher. WS-Proxy acts as an entry point of the web service to the replication framework: i) it intercepts the client request and forwards the request in the form of proxies to the dispatcher ii) and at the time of response, it replies to client, i.e. it transforms the response into required format.

WS-Dispatcher is a core component of the replication framework that manages the conversion of invocations to messages and vice versa. In addition it is responsible for preparing the web service state for all replicas. It also interfaces between the messages and the group communication component. Specifically, as a sender it takes web service invocations from the WS-Proxy and invokes locally targeted web service. Each client request is treated as a new transaction and gets a unique identifier assigned. All information is accumulated in the request table which contains: i) the request identifier, ii) the web service method call with arguments, and iii) potentially an additional transaction identifier, in case the database is called. The WS-Dispatcher prepares checkpoint as a request table and propagates to all the replicas through WS-Multicast. Finally, after successful invocation it returns a response to the client via WS-Proxy. At the replica machine the dispatcher gathers communication messages and maintains the remote request table for checkpoint.

WS-Multicast is a web service interface to the group communication system (GCS). In any replication framework the GCS plays a significant role in terms of communication, message ordering and reliability, and fault detection [42]. As another important service it provides is group membership service by which it tracks newly created and faulty replicas. This component is an interface to the transport system which is the outlet and inlet for replica messages. The transport interface uses GCS micro-protocols to communicate messages either with a SOAP layer or with JAVA-based communication. In short, the group communication system realizes replication with multicast communication and group membership service.

WS-Multicast also encapsulates reliable multicast and unicast functionality on the protocol layer. In the client's request phase the multicast component receives messages from the WS-Dispatcher and multicasts it to all the replicas that must acknowledge the

reception of message. WS-Multicast provides communication services to both the DBRC and the WS-Dispatcher, and the DBRC uses its services to unicast the database-related message to the DB (primary).

WS-Deployer is a part of the WS-Replication specification that handles the deployment of web services into the new replicas joining the replication group. Imagine the information service has been deployed into arbitrary number of replicas, and after some time a new replica appears to join the group. Any old stable replica WS-Deployer copies all the information service binaries into the new replica, in fact the selection of this candidate will also be done by the group communication system. The WS-Deployer supports the transport of binaries as SOAP attachments and serialized Java objects as well. Underlying communication of files will be carried out through the WS-Multicast service.

DBRC (Database Replication Controller) basically acts as a bridge between DB and the WS tiers. This component is formally not part of WS-Replication specification, and it has been introduced in our thesis work. DBRC acts as a bridge between two tiers which synchronizes and controls redundancy infrastructure. More concretely, through this component the WS tier can send database requests to the DB tier. Figure 5.2 depicts its interaction with the web services, i.e. the information service and the WS-Multicast service. The information service invokes database transaction by sending a request along with transaction details to the DBRC which then prepares a request for the DB tier and forwards the request to the DB (primary). Indeed DBRC already knows the address of the target DB (primary). The DBRC communicates with the DB tier via WS-Multicast: while sending a request, the transport channel uses the uniform reliable unicast primitive of GCS, and even the same primitive will be used for the response from DB (primary). The DB replication framework has repercussions from the actions of the DBRC component, usually at the time of failure. When a new primary takes over, the DBRC will be informed of changes and it informs the WS tier components. The next section will envisage the details of DB-tier replication.

5.2.2 Database Replication

The DB tier replication is done introducing an additional middleware layer on the database engine which handles the DB replication. This middleware layer is known as **XRM** (**X**ML **R**eplication **M**iddleware). The architecture follows a decentralized approach

where one separate middleware instance is tightly coupled to each replica. This implies if either the middleware or the replica crashes on the same machine, then the crashed replica will be excluded from the replication group. A detailed analysis of this approach is described in [50]. The middleware layer provides the DB tier with group communication and transactional controls. Additionally the middleware manages the multiple connection instances within a pool for achieving high performance, and hides DB complexity by providing an interface to low-level functions.

As WS-Replication complements the WS tier with the functionality of replication and fault tolerance, XRM complements to the DB tier. It is embedded with transactional control and extended group communication. Normally every database provides transaction management, in our case it has already been provided by the native XML database. But it will not be trivial to rely on the same infrastructure where the DB environment is replicated and indeed distributed, therefore separate algorithms and tools are required to control remote and local transactions.

In terms of communication, the DB replication traditionally follows a classical communication style through sockets, such as BSD [55] and JAVA [56] sockets, and remote object technologies, but a current trend propagates the use of GCS. Since long GCS has been used in distributed systems' replication frameworks and is quite mature in that sense. Its use in DB replication will enhance its potential and application into that domain. If we do not use GCS for the DB, then there are some advantages and disadvantages. The advantage of not having a group communication layer is that the replication framework's development curve is not very high. The disadvantage is implementation overhead, due to the fact that developers have to write code from scratch for low-level message communication (transport, reliability, and ordering) and carefully observe the update propagation flow. Moreover they have to deal with replica group maintenance and failure detection. Hence, implementation cost prevails and impedes the efficient replica management system. In case group communication is used it has all features described in WS-Multicast.

Figure 5.3 depicts the DB-Replication architecture showing DB replicas communication with the WS-tier's primary. The DB-Replication architecture consist of four components, the details are described shortly below.

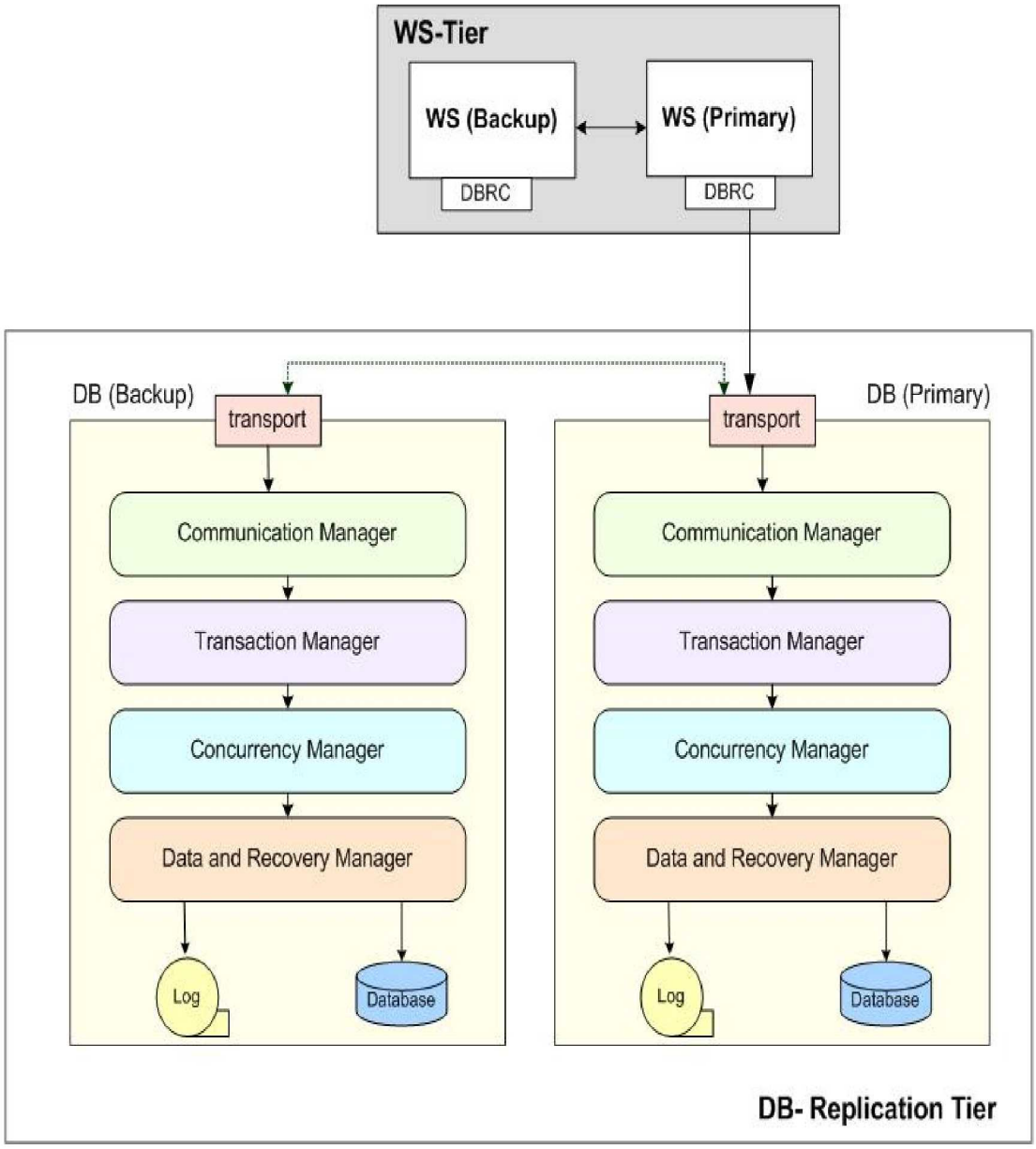

Figure 5.3: DB Replication architecture showing a primary and a backup database replica.

Communication Manager is on the top stack of all DB replicas. It handles group communication tasks and provides an interface to the group communication system. From a functionality point of view it is analogous to the WS-Multicast, but it has some additional layer of functionality which enables the group communication system to work with database-oriented message transmission. From the WS side the DBRC knows the address of the DB primary. It forwards the request to the target DB primary which is

intercepted through the communication manager. The original request sent by the DBRC contains formatted parameters like a request identifier, the address of the server which initiated the request, and the query to be executed. After the request has been formatted, the communication manager sends it to the transaction manager. First the query in the request is executed locally; if it is readset (projection), the response will be sent abruptly by the master, and if it is a write-set, the query message will be propagated to all the replicas.

The communication manager interfaces with the group communication system (GCS). With this GCS handles the fault detection and removes the faulty DB's from the replication group. As DB replication is based upon the eager primary backup algorithm [53], the GCS as a whole is responsible for electing a new master in the event of the current primary's failure. At the time of election of a new master a great deal of coordination and agreement is needed to achieve smooth election, which will be managed by the GCS.

Transaction Manager provides the foundation for transactional support for the DB replication environment. If there is no replication, it is however trivial to handle transactions coming from multiple clients. In the case of replication in place, the same transaction needs to be propagated or distributed among other hosts. In primary replication the transaction manager should provide transaction atomicity to all the replicas in the group. An example will be shown how this can be accomplished: one write-set transaction is updated at the primary site, broadcasted to all the other group participants, and the primary goes down just after propagation. The group communication instances at the non-primary sites will then undo this action by using their specialized primitives. In this way all the GCS instances will consider a transaction as single non-distributed unit.

The transaction manager issues commands to the data manager for commitments and aborts of database operations. For example common operations are BEGIN, READ and WRITE, and ABORT and COMMIT. These commands are issued locally and remotely as well. Therefore the transaction manager has two sub-components: a local and a remote transaction manager. The local transaction manager is active while the machine on which it executes is the primary one and it considers all operations coming locally. The remote transaction manager is active on all other replicas except the primary one and they are

aware of the primary and its remote operations. Both transaction managers guarantee atomicity at the replication group level and, besides executing client operations, also store transaction meta-data containing request information and WS host data which is used to manage the request. Moreover, the transaction manager prepares transactions for concurrency manager to serialize database operations.

Concurrency Manager is also an important component even for any non-replicated database environment. It provides isolation to transactions such that if two transactions access the same data and one of them is write request, then any of the operation should not maneuver the database into an inconsistent state. To achieve consistency on one centralized DB is non-trivial, but if there are replicated databases, the consistency issue is even more challenging. Therefore this component is more inclined towards replication-oriented operation. It is also viable to support cooperative deadlock detection between primary and backup copies. The concurrency manager takes transaction commands from the transaction manager and applies a consistency algorithm before forwarding the command to the DRM.

Data and Recovery Manager (DRM) is a component that handles durability and failure atomicity of transactions. The atomic property of transactions ensures that either a transaction completed or fails on all the replicas, hence the DRM is responsible for executing transactions by storing its data into the permanent storage. Therefore this component is directly attached to the database as shown in Figure 5.3. The DRM also prevents committing the aborted transactions in case of any faults. As shown in Figure 5.3 the Log manager stores data temporarily the transactions which are under process and aids DRM if the database recovers from a crash. In a replicated environment, the Log accumulates additional meta-data parameters of the incoming request: it stores request information of a WS (request identifier and host address) along with transactional details. Eventually, when a crash occurs in primary replica copy, the new primary's log container will help to recover undone transactions. Not only the Log helps to recover a DB from a crash, but it also indirectly supports to recover from a WS replica crash. For instance, a WS (initiator) sends a write-set to the DB and just after that the initiator crashes, but in between that time span the DB primary has committed this transaction to all replicas. As soon as a new WS is appointed for handling this request; it will be verified against the DB primary's log with the last incoming request's state on the old WS (primary). At the

database side, in case the transaction has been committed and the Log contains the request and transaction data, then newly appointed WS will easily track the last request's response from the DB primary and will deliver it to the client without executing the transaction again. The DRM reorganizes the log to improve the performance of crash recovery. Additionally, the DRM updates the new replica joining the replication group until its state is up-to-date, but without involving too much network load.

5.3 Implementation Strategy

5.3.1 Web Service Tier

In section 4.1 of information service architecture, it has been defined that the web service layer will use the Axis2 [57] web services framework for the development and deployment of the presentation layer. In order to realize the replication framework and enable the presentation layer to support WS-Replication, Axis2 needs to be extended at the architectural layer. Indeed Axis2 possesses a state-of-the-art architecture in comparison to other Java-based frameworks [58-60] and is likely to become the de-facto standard for the development of web service applications. In the pursuit of WS-Replication realization, the Axis2 architecture and its extension points for WS-Replication are portrayed in Figure 5.4.

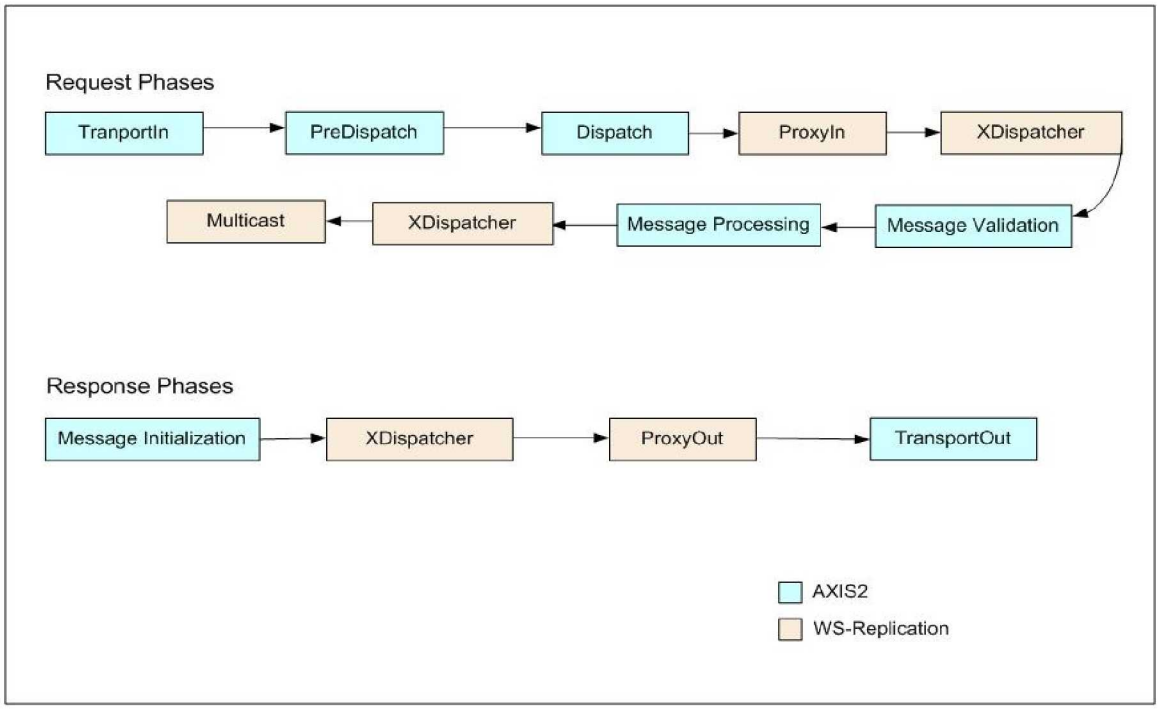

Figure 5.4: Integration of Axis2 with WS-Replication.

This section will describe the Axis2 architecture in the context of information service's replication framework requirements; more details related to the Axis2 framework can be found in [61]. Precisely, the Axis2 architecture is based upon a chain handler model: at the time when a request arrives at the Axis2 container, it flows through different request phases until the target web service is executed; the same is the case while returning responses. There are two types of phases in Axis2: system- and user-defined phases. The system defined phases cannot be altered or changed by the developer, but in case of additional application requirements (as in this case for replication) Axis2 allows extension points to add extra features. Figure 5.4 shows both, the system-defined phases (labeled Axis2) and the user ones (labeled WS-Replication), for the request as well as the response.

Request Flow (In Pipe)

In this paragraph the Axis2 request phases as pictured in Figure 5.4 are described.

The *TransportIn* phase deals with the type of transport used for communication, such as HTTP, JMS or SMTP. This phase intercepts client requests.

PreDispatch handles preliminary tasks like processing of SOAP headers.

The *Dispatch* phase contains dispatchers that find the correct service and operation to which the request is targeted. If no service or operation exists, this phase returns an exception.

ProxyIn contains the implementation of WS-Proxy, and it generates proxy objects of every web service request. In addition it forwards it to the XDispatcher.

The *XDispatcher (Extended Dispatcher)* is a realization of WS-Dispatcher, it prepares the request checkpoint for other replicas.

Message Validation probes whether XDispatcher and ProxyIn are completed successfully and validates the SOAP message for processing.

The *Message Processing Phase* executes the actual business logic of web service.

The *XDispatcher* will be again called after successful execution of the request; it prepares request information for message multicast.

Multicast is a realization of WS-Multicast, which is in fact a web service wrapper of a group communication system. For multicasting, JGroups [62] will be used as a group communication system.

Response Flow (Out Pipe)

In this paragraph the Axis2 response phases as pictured in Figure 5.4 are described.

Message Initialization acts as a starting phase for WS-Replication phases.

XDispatcher is the same component used in request phase to gather responses from the information service and translate them for the ProxyOut phase.

ProxyOut is part of WS-Replication which converts proxy objects to messages for client response and forwards them to TransportOut.

TransportOut carries out the actual transport of information service responses to clients.

5.3.2 Database Replication

The Berkeley DB XML (BDBXML) is mapped for the database tier (see Section 4.1.2). Actually BDBXML inherits a replication infrastructure from Berkeley DB as presented in [32], but the infrastructure is not flexible enough to integrate new replication algorithms and tools. The first problem with this database is that the replication architecture is tightly coupled with its own group membership service and for this reason one cannot define or use a third party tool. Secondly, in order to synchronize with WS-Tier replication, the framework needs to be customized and change its core classes; therefore it will not be a good choice to rely on Berkeley DB replication.

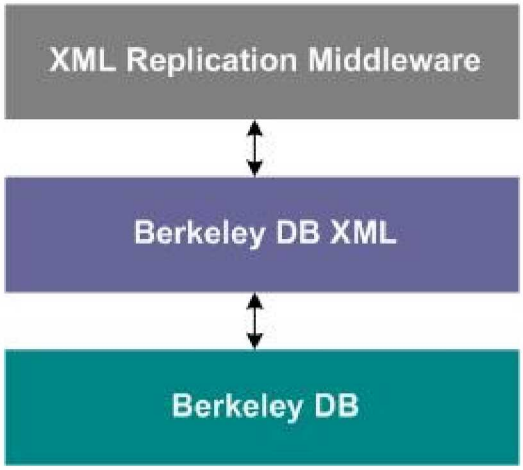

Figure 5.5: Integration of XRM with Berkeley DB XML.

To deal with these problems, XRM (XML Database Middleware) is suitable to handle dependency and to encapsulate logic to support WS-Tier. As XRM is based upon a layered architecture, it is expected to implicitly provide the required extensibility and scalability. Figure 5.5 shows the integration of XRM with Berkeley DB XML.

Chapter 6
Related Work

As the Grid information service is a core constituent of any Grid middleware, it has received a significant focus in research. A lot of research has been done to develop reliable information services catering particular Grid infrastructures. In order to give more vivid picture to our approach, this chapter discusses the related work. Concretely, this chapter elaborates the following information services used in Grids:

- MDS 4
- RGMA
- GMD

6.1 MDS 4

MDS (Monitoring and Discovery Services) 4 is an integral part of the Globus Toolkit 4.0 container, which facilitates the transparent sharing and access of resources across Globus-based Grids. GT4 comes with some basic services which are required for the deployment and invocation of other services, and MDS is the service which maintains information related to all deployed services (system- and application-oriented). It provides interaction via a command line interface and a web interface known as WebMDS [6]. The MDS infrastructure is based upon the aggregator service which further contains index, trigger and archiving services. In summary, the aggregator service accumulates a set of monitoring and discovery services [63]. More details on MDS 4 can be found at [64].

MDS 4 allows service providers to publish information instantiated from any schema.. The UNICORE_CIM model only supports the CIM model, which by itself is an open standard with wide community support. By using the GIS the user implicitly makes use of convenience methods which allow him to easily perform registry and lookup functions. But, MDS urges service providers to explicitly write convenience methods to achieve the desired functionality (provided by the GIS). While deploying any resource, the GIS requires only a resource descriptor XML file. However, it is bit cumbersome in MDS as it

needs some configuration files to manage registration. MDS 4 is tightly coupled with GT 4, it means without GT4, MDS cannot be used, but the GIS is web container agnostic application. The GIS can be deployed in nearly all containers that allow web services hosting, e.g. Tomcat [47] or Geronimo [65].

6.2 RGMA

RGMA (Relational Grid Monitoring Architecture) is a relational Grid information and monitoring system, a sub-project of the EU-DataGrid project. RGMA is based upon OGF's Grid Monitoring Architecture (GMA), but uses its own data model. GMA is a Grid application architected upon a consumer-producer model. A producer entity can publish information of resources to the central registry and a consumer can fire queries against the database in order to access the desired resources. RGMA's registry uses RDBMS technology to store backend information, thus allowing a producer/consumer to communicate in the form of SQL statements. Its core component is a mediator which, on the consumer's behalf, searches for the best producer based on its search criteria. RGMA provides support for VO-based Grid networks, but it requires separate a DB instance for every VO. It is built upon the Java Servlets technology [7] to realize producer/consumer entities, thereby allowing these entities to communicate via web based interfaces. Further details can be found at [7].

RGMA and the GIS have some tangible differences with respect to the information model and implementation. RGMA uses RDMS on its backend which is a server and not light-weight software. Indeed it provides efficient SQL interfaces for projection and data manipulation, but data has to be stored in the form of tuples (table instance). RGMA performs some internal logic to transform publisher information into an relational format. Instead the GIS is based upon a light-weight, native, and embedded XML database. It is light-weight in terms of installation and ease of projection and data manipulation on XML documents. As XML database stores resource information in XML format and allows less logic to realize publishing, but lacks performance in complex queries. RGMA stores information using its own proprietary data model. The GIS is based upon CIM, an open standard and widely known model to work with.

6.3 GMD

GMD (Grid Market Directory) is a project of GridLabs, University of Melbourne, Australia. It is a web service-oriented Grid information service to accumulate resource information and also to represent supply-demand information about resources in the context of Grid economy. It has two major components: the GMD Portal Manager (GPM) and the GMD Query Web Service (GQS) [66]. The GPM offers publication, management, and browsing of Grid services to service providers and consumers through web interfaces. The GQS is a web service which provides querying and searching of services which fulfil the requirements of a particular job which has to be executed. With respect to databases, the GMD uses RDBMS on the backend. To obtain more information please refer to [67].

Generally the difference between the GIS and the GMD is same as between the GIS and RGMA. The GMD uses a separate XML-based data model for service provider and consumer respectively. But the GIS is using one unified model to advertise services and to specify resource requests.

Chapter 7
Conclusion and Outlook

7.1 Conclusion

In this thesis, I have presented a web services-based interoperable Grid Information Service, the GIS. The service exposes programmatic interfaces covering general and specific aspects of functionality with respect to end users and other software agents. From the end user point of view, an information service offers a web user interface providing common functionalities such as registration and querying of resource information. Concerning other software agents such as job schedulers or information providers, a set of particular interfaces is provided in order to invoke information service methods.

The CIM-based information model, called UNICORE_CIM, has been implemented using XML schema (for CIM details please refer to Section 2.6.3). This schema is analogous to the database model implemented in the Berkley DBXML database. This CIM-based model provides extensions with respect to UNICORE resource model. With this, it is one of the applications of our information model in UNICORE Grids.

Furthermore, in this thesis I proposed a solution for a reliable and highly available Grid information service through fine grained design of a multi-tier replication architecture. This multi-tier architecture actually covers both information service and database tier. For the information service layer, WS-Replication has been taken as a specification targeting web service container reliability, but with additional customized components specially catered for our problem domain. As far as database tier is concerned, XRM (XML Replication Middleware) is introduced to manage the replication framework, and providing communication infrastructure among other database replicas and the web service's tier.

7.2 Future Work

The implementation of the Grid information service is currently a prototype, but it fulfills all of implementation's key requirements. There are still many extension points where the project can grow and hence be extended to include further requirements. Before deriving any innovation and extending the design of the Grid information service, it is however necessary to evaluate the feasibility of such extensions and judge the significance of new enhancements. Saying that and considering the current development and research trends in the fast pacing Grid area, the most valuable implementation requirements are:

- As described in the state-of-the-art section (see Section 2.5.3), service-oriented Grids, such as those based on WSRF and WSN, are gaining ample focus in current developments. The next step is to implement our architecture using next generation service-oriented Grid tools based on OGSA specifications.
- The support for distributed registries will also be considered in future implementations. Essentially, this feature will be very much helpful with respect to the realization of virtual organizations
- Grid information services are indeed essential to any Grid middleware as other middleware services are highly dependent on its functionality. In terms of information service application it will be an advantage to integrate with other Grid tools such as resource brokers, like e.g. Gridbus, and information providers such as Ganglia and Hawkeye.
- The information model on the backend of our service is based on the CIM schema. As there are other Grid middlewares relying on different standards, e.g. MDS uses the GLUE schema (see Section 2.6.2), interoperability between other information models will result in the integration of our information service with other Grid information systems, e.g. with MDS.
- The implementation of a replication framework is also one of the requirements which pose tremendous advantage in terms of highly-distributed and versatile Grids. Currently this requirement has a high priority, and the respective implementations will be carried out after the aforementioned requirements are fulfilled..

Bibliography

[1] I. Foster, C. Kesselman, and S. Tuecke, *Anatomy of the Grid: Enabling Scalable Virtual Organizations*.

[2] *UNICORE Project,* http://www.unicore.org/, accessed on:11.03.2006.

[3] *Globus Alliance,* http://www.globus.org.

[4] *gLite: Lightweight Middleware for Grid Computing,* http://glite.web.cern.ch, accessed on: 12.04.2006.

[5] M. Baker, R. Buyya, and D. Laforenza, *Grids and Grid technologies for wide-area distributed computing, Software - Practice and Experience,* 2002.

[6] G. Alliance, *GT 4.0 WS MDS WebMDS*, http://www.globus.org/toolkit/docs/4.0/info/webmds/.

[7] R. Byrom, B. Coghlan, A. Cooke, R. Cordenonsi, L. Cornwall, A. Datta, A. Djaoui, L. Field, S. Fisher, S. Hicks, S. Kenny, J. Magowan, W. Nutt, D. O'Callaghan, M. Oevers, N. Podhorszki, J. Ryan, M. Soni, A. W. a. Paul Taylor, and X. Zh, *R-GMA: A Relational Grid Information and Monitoring System*.

[8] *GLUE Information Model,* http://glueschema.forge.cnaf.infn.it/, accessed on:21.03.2006.

[9] *NextGRID Project,* http://www.nextgrid.org/, accessed on:11.03.2006.

[10] A. Patil, http://www.adarshpatil.com/grid, accessed on:24 June 2006.

[11] B. Jacob, *Grid Computing: What are the key components?,* 2003.

[12] D. Erwin, *UNICORE Plus Final Report: Uniform Interface to Computing Resources,* in *Dept. of Applied Mathematics*: Forschung Zentrum Jülich, 2000.

[13] I. Foster, *Globus Toolkit Version 4: Software for Service-Oriented Systems*.

[14] E. J. T. Ian Bird, *Middleware for the Next Generation Grid Infrastructure,* www2.twgrid.org/event/isgc2005/Abstract/Erwin Laure-Ian Bird_1.doc, accessed on:27/04/2006.

[15] *Enabling Grids for E-science,* http://www.eu-egee.org/, accessed on:15/05/2006.

[16] I. J. Taylor, *From P2P to Web Services and Grids: Peers in a Client/Server World*, 2004, pp. 43 - 56.

[17] T. J. David Chappel, *Java Web Services: Using Java in Service Oriented Architecture*, First Edition ed.: Oreilly, Inc., 2002.

[18] C. K. Ian Foster, Jeffrey M. Nick, Steven Tuecke, *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration* p. 18.

[19] J. Unger and M. Haynos, *A visual tour of Open Grid Services Architecture*, 2005, http://www-128.ibm.com/developerworks/webservices/library/gr-visual/.

[20] Globus, *WSRF and GT4: Chapter 1. Key Concepts* http://gdp.globus.org/gt3-tutorial/multiplehtml/ch01s05.html.

[21] *EU-DataTAG,* http://datatag.web.cern.ch/datatag/, accessed on:26.06.2006.

[22] *US-IVGDL,* http://www.ivdgl.org/, accessed on:26.06.2006.

[23] *EGEE Grids,* http://public.eu-egee.org/, accessed on:24.06.2006.

[24] *NorduGrid,* www.nordugrid.org/, accessed on:24.07.2006.

[25] *LCG,* http://lcg.web.cern.ch/LCG/, accessed on:26.06.2006.

[26] DMTF, *CIM Concepts White Paper For CIM v2.4+*.

[27] B. Bruegge and A. H. Dutoit, *Object-Oriented Software Engineering: Using UML, Patterns, and Java* 2nd Edition ed.: Pearson Education.

[28] R. C. Martin, *Liskov Substitution Principle*, http://www.objectmentor.com/resources/articles/lsp.pdf.

[29] DMTF, *CIM Schema Tutorial,* http://www.wbemsolutions.com/tutorials/DMTF/cim-schema.html.
[30] DMTF, *CIM Concepts Application For CIM v2.7.*
[31] WSO2, *Apache Axis2 1.0 Unleashed.*
[32] S. Software, *Berkeley DB XML 2.2,* http://www.sleepycat.com/products/bdbxml.html, accessed on:4 April 2006.
[33] W3C, *XQuery 1.0: An XML Query Language*, http://www.w3.org/TR/xquery/.
[34] W3C, *XML Schema,* http://www.w3.org/XML/Schema, accessed on:02 July 2006.
[35] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, 1995.
[36] D. Obasanjo, *W3C XML Schema Design Patterns: Avoiding Complexity*, 2002, http://www.xml.com/pub/a/2002/11/20/schemas.html.
[37] D. Obasanjo, *W3C XML Schema Design Patterns: Dealing With Change*, 2002, http://www.xml.com/pub/a/2002/07/03/schema_design.html.
[38] *Hawkeye: A Monitoring and Management Tool for Distributed Systems,* http://www.cs.wisc.edu/condor/hawkeye/, accessed on:04 May, 2006.
[39] S. Microsystems, *Core J2EE Pattern Catalog - Business Delegate,* http://java.sun.com/blueprints/corej2eepatterns/Patterns/BusinessDelegate.html, accessed on:06 May, 2006.
[40] *J2EE Patterns Catalog - TransferObject,* http://java.sun.com/blueprints/patterns/catalog.html.
[41] *W3C: Web Services Architecture,* http://www.w3.org/TR/ws-arch/, accessed on:11 June, 2006.
[42] J. D. George Coulouris, Tim Kindberg, *Distributed Systems, Concepts and Design*, Third ed.: Addison Wesley, 2001.
[43] M. V. S. Andrew S. Tannenbaum, *Distributed Systems, Principes and Design*: Prentice Hall.
[44] C.-L. F. Deron Liang, Chyouhwa Chen, *FT-SOAP: A Fault-tolerant web service.*
[45] D. Jayasinghe, *FAWS for SOAP-based Web services*, 2005, http://www-128.ibm.com/developerworks/webservices/library/ws-faws/.
[46] JBoss, *JBoss Application Server.*
[47] Tomcat, *Apache Tomcat Web Server.*
[48] C. Amza, A. L. Cox, and W. Zwaenepoel, *Distributed Versioning: Consistent Replication for Scaling Back-end Databases of Dynamic Content Web Sites.*
[49] B. K. Shuqing Wu, *Postgres-R(SI): Combining replica control with concurrency control based on snapshot isolation.,* in *International Conference on Data Engineering (ICDE).*
[50] Y. Lin, B. Kemme, M. Patino-Martınez, and R. Jimenez-Peris, *Middleware based data replication providing snapshot isolation.,* in *SIGMOD International Conference on Management of Data.*
[51] ADAPT, *The Adapt Framework for Adaptable and Composable Web Services,* http://adapt.ls.fi.upm.es/, accessed on:11.06.2006.
[52] B. Kemme, R. Jimenez-Peris, M. Patino-Martinez, and J. Salas, *Exactly Once Interaction in a Multi-tier Architecture.*
[53] M. Wiesmann, A. Schiper, B. Kemme, and G. Alonso, *Understanding Replication in Databases and Distributed Systems.*
[54] F. P.-S. Jorge Salas, Marta Patino-Martinez, Ricardo Jimenez-Peris, *WS-Replication: A Framework for Highly Available Web Services,* in *15th International World WIde Web Conference*, Edinburg, Scotland, 2006.

[55] C. Hafey, *Programming the BSD Socket interface,* http://www.ecst.csuchico.edu/~chafey/prog/sockets/.
[56] *Custom Networking: All About Sockets*, http://java.sun.com/docs/books/tutorial/networking/index.html.
[57] A. S. Foundation, *Axis2 Web Services Framework,* http://ws.apache.org/axis2/.
[58] CodeHaus, *XFire- SOAP Framework,* http://xfire.codehaus.org/, accessed on:4 May, 2006.
[59] A. S. Foundation, *Apache Axis,* http://ws.apache.org/axis/, accessed on:4 May 2006.
[60] S. Microsystems, *Java Web Services Developer Pack.*
[61] A. S. Foundation, *Axis2 1.0 Architecture Guide*, http://ws.apache.org/axis2/1_0/Axis2ArchitectureGuide.html.
[62] *JGroups: A Toolkit for Reliable Multicast Communication*, accessed on:21 June 2006.
[63] I. Foster, *A Globus Primer: Or, Everything You Wanted to Know about Globus, but Were Afraid To Ask Describing Globus Toolkit Version 4*, 2005, http://www.globus.org/toolkit/docs/4.0/key/GT4_Primer_0.6.pdf.
[64] G. Alliance, *GT 4.0: Information Services*, http://www.globus.org/toolkit/docs/4.0/info/.
[65] *Apache Geronimo,* http://geronimo.apache.org/, accessed on:12 June 2006.
[66] J. Yu, S. Venugopal, and R. Buyya, *Grid Market Directory: A Web Services based Grid Service Publication Directory*.
[67] *Grid Market Directory (GMD),* http://www.gridbus.org/gmd/.

Jül-4263
Februar 2008
ISSN 0944-2952