

# **Evolving a Secure Grid-Enabled, Distributed Data Warehouse: A Standards-Based Perspective**

**Xiaoyu Li**

# **Evolving a Secure Grid-Enabled, Distributed Data Warehouse: A Standards-Based Perspective**

by

**Xiao-yu Li**

**Dissertation**

submitted in fulfilment  
of the requirements  
for the degree of

**Magister Technologiae**

in

**Information Technology**

in the

**Faculty of Engineering**

of the

**Nelson Mandela Metropolitan University**

**Supervisor: Dr. Maree Pather  
January 2007**

# **Declaration**

I, **Xiao-yu Li**, hereby declare that:

- The work in this dissertation is my own work.
- All sources used or referred to have been documented and recognized.
- This dissertation has not previously been submitted in full or partial fulfilment of the requirements for an equivalent or higher qualification at any other recognized educational institution.

---

Xiao-yu. Li

12 January 2007

# **Abstract**

As digital data-collection has increased in scale and number, it becomes an important type of resource serving a wide community of researchers. Cross-institutional data-sharing and collaboration introduce a suitable approach to facilitate those research institutions that are suffering the lack of data and related IT infrastructures.

Grid computing has become a widely adopted approach to enable cross-institutional resource-sharing and collaboration. It integrates a distributed and heterogeneous collection of locally managed users and resources. This project proposes a distributed data warehouse system, which uses Grid technology to enable data-access and integration, and collaborative operations across multi-distributed institutions in the context of HV/AIDS research.

This study is based on wider research into OGSA-based Grid services architecture, comprising a data-analysis system which utilizes a data warehouse, data marts, and near-line operational database that are hosted by distributed institutions. Within this framework, specific patterns for collaboration, interoperability, resource virtualization and security are included.

The heterogeneous and dynamic nature of the Grid environment introduces a number of security challenges. This study also concerns a set of particular security aspects, including PKI-based authentication, single sign-on, dynamic delegation, and attribute-based authorization. These mechanisms, as supported by the Globus Toolkit's Grid Security Infrastructure, are used to enable interoperability and establish trust relationship between various security mechanisms and policies within different institutions; manage credentials; and ensure secure interactions.

## **Acknowledgements**

I am most grateful for the support and help of my supervisor, Dr Maree Pather, and Mrs Bron Kaplan, language specialist in the School of ICT, both of whom spent many hours correcting my special flavour of Sino-English.

Also, I am most grateful to my parents and brother, for their love, support and encouragement. In addition, I would like to thank my colleagues, for their assistance and understanding.

Furthermore, I would like to thank Nelson Mandela Metropolitan University, whose financial support made this research possible.

# **Table of Contents**

List of Figures .....	viii
List of Tables.....	ix
List of Acronym .....	x
Chapter 1. Introduction .....	1
1.1. Problem-Context.....	2
1.2. Problem Statement .....	6
1.3. Research Objectives .....	6
1.4. Methodology.....	7
1.5. Layout of Dissertation .....	7
Chapter 2. The Grid Approach to Cross-Institutional Collaboration.....	9
2.1. Grid Technology .....	9
2.1.1. Grid definitions .....	10
2.1.2. Virtual organization .....	12
2.1.3. Grid architecture .....	13
2.1.4. Standards-Bodies .....	17
2.1.5. Grid evolution .....	17
2.1.5.1. <i>The First Generation</i> .....	18
2.1.5.2. <i>The Second Generation</i> .....	18
2.1.5.3. <i>The Third Generation</i> .....	21
2.2. Combining Database and Grid Technology.....	23
2.2.1. Current database-management technologies.....	24
2.2.1.1. <i>Basic Concepts</i> .....	24
2.2.1.2. <i>Single-Site DBMSs</i> .....	25
2.2.1.3. <i>Federated DBMSs</i> .....	27
2.2.1.4. <i>Security</i> .....	29
2.2.2. Data life-cycle classification.....	29
2.2.3. Structured data, sources and resources.....	32
2.2.4. Categories of structured data and its applications.....	33
2.2.5. Integration strategies.....	35
2.3. Summary .....	36
Chapter 3. Grid-Enabled Distributed Data-Warehouse Systems.....	37

3.1.	Overview of Data Warehousing and Data-mining .....	38
3.2.	Description of Example VO .....	40
3.3.	System Functional-Capabilities .....	41
3.3.1.	Virtualization .....	42
3.3.1.1.	<i>Data-access Transparency</i> .....	42
3.3.2.	Data-collection .....	43
3.3.2.1.	<i>Extraction and Transport</i> .....	44
3.3.2.2.	<i>Transformation and Loading</i> .....	45
3.3.3.	Data operations .....	46
3.3.3.1.	<i>Data-Access and Integration</i> .....	46
3.3.3.2.	<i>Data Analysis and Interpretation</i> .....	47
3.3.4.	Data resource publishing and discovery .....	47
3.3.5.	Provenance .....	48
3.3.6.	Resource management .....	48
3.3.7.	Job execution management .....	49
3.3.8.	Metadata management .....	50
3.3.9.	Monitoring .....	51
3.3.10.	Security .....	52
3.4.	Summary .....	53
<b>Chapter 4. A Proposed System Framework.....</b>		<b>54</b>
4.1.	Evolution of Distributed Computing .....	55
4.1.1.	Traditional distributing computing .....	55
4.1.1.1.	<i>Socket Programming</i> .....	56
4.1.1.2.	<i>RPC</i> .....	56
4.1.1.3.	<i>Java RMI</i> .....	57
4.1.1.4.	<i>DCOM</i> .....	57
4.1.1.5.	<i>CORBA</i> .....	58
4.1.2.	Service-Oriented Architecture .....	58
4.1.3.	Web Services .....	60
4.1.3.1.	<i>SOAP</i> .....	61
4.1.3.2.	<i>WSDL</i> .....	62
4.1.3.3.	<i>UDDI and WS-Inspection</i> .....	63
4.2.	Open Grid Services Architecture .....	64
4.2.1.	Advantages of Web Services for Grid computing .....	64

4.2.2.	Service-Oriented view .....	65
4.2.3.	The Grid Service .....	66
4.2.4.	OGSA core services .....	67
4.3.	WSRF and WS-Notification .....	70
4.3.1.	OGSA requires stateful services.....	70
4.3.2.	WS-Resource .....	71
4.3.3.	WSRF and WS-Notification family of specifications .....	73
4.3.4.	WSRF and OGSF .....	76
4.4.	System Framework .....	77
4.4.1.	Infrastructure services .....	80
4.4.2.	Core services .....	82
4.4.2.1.	<i>Publishing and Discovery Services</i> .....	82
4.4.2.2.	<i>Resource Management Services</i> .....	83
4.4.2.3.	<i>Job Execution Management Services</i> .....	86
4.4.2.4.	<i>Information Services</i> .....	89
4.4.2.5.	<i>Data Services</i> .....	93
4.4.3.	Grid portal .....	97
4.4.4.	Models.....	100
4.4.4.1.	<i>Data-Collection Model</i> .....	101
4.4.4.2.	<i>Data-Access and Integration Model</i> .....	102
4.5.	Summary .....	104
Chapter 5.	Security.....	106
5.1.	A Brief Security Primer .....	107
5.2.	Security Technology .....	108
5.2.1.	Firewalls .....	109
5.2.2.	Intrusion Detection Systems .....	109
5.2.3.	Cryptography.....	110
5.2.3.1.	<i>Symmetric Cryptosystems</i> .....	111
5.2.3.2.	<i>Asymmetric Cryptosystems</i> .....	111
5.2.3.3.	<i>Digital Signatures</i> .....	112
5.2.3.4.	<i>Digital Certificates</i> .....	113
5.2.3.5.	<i>Public Key Infrastructure</i> .....	113
5.3.	Grid Security Problems .....	114
5.3.1.	Terminology .....	115

5.3.2.	Security requirements .....	116
5.3.3.	Security policy .....	117
5.4.	A Grid Security Architecture .....	118
5.5.	Grid Security Infrastructure .....	121
5.5.1.	Authentication .....	121
5.5.1.1.	<i>Kerberos and SSH</i> .....	122
5.5.1.2.	<i>Using PKI for Authentication</i> .....	122
5.5.1.3.	<i>Grid Certificate Authority</i> .....	125
5.5.2.	SSO and delegation .....	127
5.5.2.1.	<i>Proxy Certificates</i> .....	127
5.5.2.2.	<i>Uses for SSO</i> .....	129
5.5.2.3.	<i>Uses for Delegation</i> .....	129
5.5.3.	Authorization .....	130
5.5.3.1.	<i>Grid Authorization Framework Concepts</i> .....	131
5.5.3.2.	<i>Grid Authorization Architecture</i> .....	134
5.5.3.3.	<i>Grid Authorization Framework</i> .....	136
5.5.4.	GSI security model for OGSA .....	139
5.5.4.1.	<i>Web Services Security</i> .....	139
5.5.4.2.	<i>OGSA Security Model</i> .....	143
5.5.4.3.	<i>GSI Security Model for OGSA</i> .....	145
5.6.	Security Solutions .....	147
5.6.1.	Message protection .....	148
5.6.2.	Authentication, delegation and SSO .....	150
5.6.3.	MyProxy protocol .....	151
5.6.4.	Authorization .....	154
5.6.4.1.	<i>SAML</i> .....	156
5.6.4.2.	<i>Shibboleth</i> .....	157
5.6.4.3.	<i>GridShib</i> .....	158
5.6.4.4.	<i>XACML Authorization Framework</i> .....	158
5.6.4.5.	<i>GT4 SAML/XACML Authorization Framework</i> .....	159
5.6.5.	Putting it all together .....	161
5.7.	Summary .....	164
Chapter 6.	Conclusion .....	166
References	.....	172

# **List of Figures**

## **Chapter 2**

Figure 2.1 Grid Architecture .....	14
------------------------------------	----

## **Chapter 3**

Figure 3.1 VO Description .....	41
Figure 3.2 Building a Data Warehouse .....	44

## **Chapter 4**

Figure 4.1 SOA.....	59
Figure 4.2 SOAP Envelope .....	62
Figure 4.3 WSDL Document.....	63
Figure 4.4 Relationships between OGSA, WSRF, and Web Services.....	71
Figure 4.5 System Framework .....	77
Figure 4.6 Service Framework .....	80
Figure 4.7 Service Discovery .....	82
Figure 4.8 Job Execution Management.....	88
Figure 4.9 An Example of Monitoring.....	91
Figure 4.10 Grid Portal Architecture.....	98
Figure 4.11 Data-Collection Model.....	100
Figure 4.12 Data-Access and Integration Model.....	104

## **Chapter 5**

Figure 5.1 Grid Security Architecture .....	120
Figure 5.2 X.509 v3 Certificate Structure .....	123
Figure 5.3 Grid Authorization Architecture .....	134
Figure 5.4 Web Services Security Specifications.....	141
Figure 5.5 Point-to-Point and End-to-End Security .....	148
Figure 5.6 SOAP Message Implementing WS-Security .....	150
Figure 5.7 MyProxy Protocol .....	152
Figure 5.8 XACML Authorization Model.....	159
Figure 5.9 GT4 Authorization Framework.....	160
Figure 5.10 Security Model.....	162

## **List of Tables**

Table 4.1 Resource Management .....	84
-------------------------------------	----

## **List of Acronyms**

<b>AA:</b>	Attribute Authority
<b>ACL:</b>	Access Control List
<b>ADO:</b>	ActiveX Data Objects
<b>ADF:</b>	Access control Decision Function
<b>AEF:</b>	Access control Enforcement Function
<b>API:</b>	Application Programming Interface
<b>CA:</b>	Certificate Authority
<b>CAS:</b>	Community Authorization Service
<b>CERN:</b>	European Council for Nuclear Research
<b>CIM:</b>	Common Information Model
<b>COM:</b>	Component Object Model
<b>CORBA:</b>	Common Object Request Broker Architecture
<b>CRL:</b>	Certificate Revocation List
<b>CSG:</b>	Candidate Set Generator
<b>DBMS:</b>	Database Management System
<b>DCE:</b>	Distributed Computing Environment
<b>DCOM:</b>	Distributed Component Object Model
<b>DES:</b>	Data Encryption Standard
<b>DMTF:</b>	Distributed Management Task Force
<b>DN:</b>	Distinguished Name
<b>DNS:</b>	Domain Name System
<b>DSS:</b>	Digital Signature Standard
<b>EPR:</b>	(WS-Addressing) EndPoint Reference
<b>EPS:</b>	Execution Planning Services
<b>ETL:</b>	Extract, Transform, and Load
<b>FIPA:</b>	Foundation for Intelligent Physical Agents
<b>FIPA-ACL:</b>	FIPA - Agent Communication Language
<b>FTP:</b>	File Transfer Protocol
<b>GGF:</b>	Global Grid Forum
<b>GMA:</b>	Grid Monitoring Architecture
<b>GPDK:</b>	Grid Portal Development Kit

<b>GSI:</b>	Grid Security Infrastructure
<b>GT:</b>	Globus Toolkit
<b>GUI:</b>	Graphical User Interface
<b>HTTP:</b>	HyperText Transfer Protocol
<b>IdP:</b>	Identity Provider
<b>IDL:</b>	Interface Definition Language
<b>IDS:</b>	Intrusion Detection System
<b>ITU-T:</b>	International Telecommunication Union - Telecommunication Standardization Sector
<b>JDBC:</b>	Java Database Connectivity
<b>JM:</b>	Job Manager
<b>JSP:</b>	Java Server Page
<b>LCG:</b>	LHC Computing Grid Project
<b>LDAP:</b>	Lightweight Directory Access Protocol
<b>LHC:</b>	Large Hadron Collider
<b>MAC:</b>	Message Authentication Code
<b>MIDL:</b>	Microsoft's Interface Definition Language
<b>OCSP:</b>	Online Certificate Status Protocol
<b>ODBC:</b>	Open Database Connectivity
<b>OGF:</b>	Open Grid Forum
<b>OGSA:</b>	Open Grid Services Architecture
<b>OGSA-DAI:</b>	Open Grid Services Architecture - Data-access and Integration
<b>OGSA-DQP:</b>	Open Grid Services Architecture - Distributed Query Processor
<b>OGSA-EMS:</b>	Open Grid Services Architecture - Execution Management Services
<b>OGSI:</b>	Open Grid Services Infrastructure
<b>OID:</b>	Object Identifier
<b>OMG:</b>	Object Management Group
<b>ONC:</b>	Open Networking Computing
<b>OODBMS:</b>	Object-Oriented Database Management System
<b>OSF:</b>	Open Software Foundation

<b>PAM:</b>	Pluggable Authentication Module
<b>PAP:</b>	Policy Administration Point
<b>PCI:</b>	Proxy Certificate Information
<b>PDP:</b>	Policy Decision Point
<b>PEP:</b>	Policy Enforcement Point
<b>PGP:</b>	Pretty Good Privacy
<b>PIP:</b>	Policy Information Point
<b>PII:</b>	Personally Identifiable Information
<b>PKI:</b>	Public Key Infrastructure
<b>QoS:</b>	Quality of Service
<b>RA:</b>	Registration Authority
<b>RDF:</b>	Resource Description Framework
<b>RDN:</b>	Relative Distinguished Name
<b>RDBMS:</b>	Relational Database Management System
<b>RMI:</b>	Remote Method Invocation
<b>RPC:</b>	Remote Procedure Call
<b>SAML:</b>	Security Assertion Markup Language
<b>SASL:</b>	Simple Authentication and Security Layer
<b>SCT:</b>	Security Context Token
<b>SDK:</b>	Software Development Kit
<b>SDSC:</b>	San Diego Supercomputer Centre
<b>SOA:</b>	Service-Oriented Architecture
<b>SOAP:</b>	Simple Object Access Protocol
<b>SP:</b>	Service Provider
<b>SQL:</b>	Structured Query Language
<b>SRB:</b>	Storage Resource Broker
<b>SSH:</b>	Secure Shell
<b>SSO:</b>	Single Sign-On
<b>STS:</b>	Security Token Service
<b>TLS:</b>	Transport Level Security
<b>UDDI:</b>	Universal Description, Discovery and Integration
<b>UNICORE:</b>	UNiform Interface to COmputing RESources
<b>URI:</b>	Uniform Resource Identifier
<b>URL:</b>	Universal Resource Locator

<b>VO:</b>	Virtual Organization
<b>W3C:</b>	World Wide Web Consortium
<b>WBEM:</b>	Web-based Enterprise Management
<b>WSDL:</b>	Web Services Description Language
<b>WSDM:</b>	Web Services Distributed Management
<b>WSDM-MUWS:</b>	Web Services Distributed Management - Management Using Web Services
<b>WSDM MOWS:</b>	Web Services Distributed Management - Management Of Web Services
<b>WSIL:</b>	Web Services Inspection Language
<b>WSRF:</b>	Web Services Resource Framework
<b>X.509 EEC:</b>	X.509 End-Entity Certificate
<b>XACML:</b>	eXtensible Access Control Markup Language
<b>XML:</b>	eXtensible Markup Language
<b>XML GED:</b>	XML Global Element Declaration

# **Chapter 1.**

## **Introduction**

Digital data is now fundamental to all branches of science and engineering. It plays a major role in medical research and diagnosis, and supports business and governmental decision-making processes.

Individual collections of data typically specialise in holding information of interest to particular communities. This information is held in databases, that is, structured documents in structured assemblies of binary files. In an increasing number of scientific disciplines, large data-collections are emerging as important resources serving a wide community of researchers. The communities of researchers that need to access and analyze this data are often owned by multiple institutions and geographically distributed, as are the computing and storage resources that these communities rely upon to store and analyze their data.

Grid Computing (Foster & Kesselman, 1998) has emerged as a new field, using multiple distributed resources to cooperatively work on a single application, as there is a need for coordinate resource-sharing and problem-solving across multiple institutions, both scientific and commercial. Applications in this context include distributed computing for computationally demanding data analysis (pooling of compute power and storage), the federation of diverse distributed datasets, collaborative visualization of large scientific datasets (pooling of expertise), and coupling of scientific instruments with remote computers and archives. Both science and industry can benefit from Grids at present.

In South Africa, there is an urgent need, for example, to establish well-structured data and corresponding IT infrastructure (e.g., network and programs) in order to facilitate data-based analysis in HIV/AIDS research areas. In other words, it is necessary that the established resources (e.g., data, computing facilities, and the Internet) can be used efficiently and effectively to

promote cooperation between HIV research institutions. Grid Computing is a suitable approach to realize these considerations.

This chapter, firstly, explores the HIV/AIDS research problem-context in South Africa as an example of more general problems in which Grid Computing can be applied. The difficulties and challenges implicit in this particular problem-context make it a pertinent example. The research objectives, methodology and the layout of this dissertation are described subsequently, based on this problem-context.

## **1.1. Problem-Context**

In South Africa, some HIV/AIDS research institutions suffer from a lack of digital-data storage resources, data-management infrastructure, connectivity infrastructure and corresponding data-intensive analysis facilities to perform valid HIV/AIDS-specific research tasks. HIV/AIDS patients' data is mostly owned privately by medical institutions, such as hospitals and clinics. This data is primarily used for recording patients' personal and diagnostic information and treatment histories, rather than for supporting HIV/AIDS-specific data analysis. On the other hand, some research institutions have already established their own data storage resources (such as databases, data warehouses and data marts) to aggregate patients' demographic data, and have data-based analysis facilities to support their research. However, this data can, typically, only be used for their own purposes. Cross-institutional resource-sharing and systems-integration is an economical way to provide high-performance computing to those institutions that lack resources like server clusters, high-performance networks, huge data storage resources or expensive analysis (data-mining) applications. Collaborative HIV/AIDS research would obviously benefit from this approach, particularly in developing countries. Ideally, this should enable data-sharing and facilitate cooperation across multiple, disparate systems across research institutions, by integrating existing standards and technologies and compensating for lack of resources and infrastructure. Given the pandemic nature of HIV/AIDS, it is imperative to build HIV/AIDS data storage-resources (e.g. to store patients'

demographic data) and corresponding analysis services for access by all who require such information. This dissertation examines how Grid Computing and Data Warehousing can be utilised to provide interoperable collaboration among research institutions and how secure access to “mined” information can be provided.

To this end, data collection, data analysis and data-sharing are three key initial steps towards collaborative HIV/AIDS research. They are elaborated below.

## ● **Data-Collection**

Data-collection in this problem-context implies collecting data from distributed data-providers. It is used for building data storage resources on HIV/AIDS patients (e.g. to allow for demographics-based analysis). This digital data is typically stored in on-line operational databases that are owned by medical institutions, such as hospitals or clinics, to support their day-to-day business processing. The data normally includes a patient’s personal information (such as name, ID number, contact information, diagnosis, treatments, etc.), and is frequently changed/edited.

However, a researcher might be interested in testing hypotheses and examining trends in order to derive patterns from the statistical manipulation of this data. For example, a researcher might need to gather as much as possible valid HIV/AIDS patients’ data to test whether the number of HIV positive patients in the Eastern Cape Province in South Africa has changed significantly since 2004, or to establish what the average age of HIV-positive patients in the Eastern Cape is. Access to such information (and the means for processing the data and making it available) is crucial to HIV/AIDS research. Obviously, the volume of data collected and processing-accuracy of the information produced affects the analysis results directly. Data-collection typically involves the collecting of the required data from multiple, disparate, physically-distributed data-providers. The data in different institutions may be used for different purposes, may be managed by a variety of management systems, and may be in a range of possible formats.

Various permutations for data-collection are possible. In the HIV/AIDS demographics-based research, only the data relevant to demographics research need be extracted, according to the requirements of the HIV/AIDS data analysis (the relevant fact tables and dimension tables, in data warehousing parlance). For example, HIV/AIDS patients' names and ID would be relatively unnecessary in this analysis, but patients' age, treatment, diagnostic information, etc., would be important. Moreover, personal names and IDs may lead to ethical and privacy violations. The extracted data needs to be transformed and loaded to appropriate destination storages; extract, transform and load are typical data warehousing procedures.

In the problem-context under discussion, such data-collections would be performed periodically. The collected data would only be updated when the original data is changed.

## ● **Data Analysis**

In this scenario, assume HIV/AIDS research mainly concerns analyses based on HIV/AIDS patients' demographic data. Patients' data is collected from medical institutions through data-collection processes and becomes part of a research institution's data storage. This provides the fundamental data to perform analysis tasks. Basically, the purposes of data analysis are: to identify trends, collections/groupings and anomalies, to test theories and to derive patterns. The development of specific data-analysis services' is based on researchers' requirements. Typically, these are information-mining procedures (programmatic functions) which may be called to interpret and satisfy a user's query. Various considerations emanate from this situation: resource-location, resource-management, execution-management, security, etc.

## ● **Data-sharing**

Data-sharing is, obviously, an important capability to achieve collaborative HIV/AIDS research. It implies access to, and integration of, data from multiple data storage resources hosted by physically distributed institutions. Data-collection can involve preparing the HIV/AIDS patients' demographic

data, to varying degrees. Each research institution could, for instance, build its own aggregated data and data-analysis services. The collected data contents would most likely be different because of different research interests or directions. One institution may wish to perform a specific analysis task, but the required data is only available at another institution.

Data-sharing across multiple institutions provides a way to supply data to the analysis process with no need to collect new data, which may be time-consuming and costly. However, if institution A should need data from institutions B and C to complete its analysis, institution A should have the ability to locate and then access the data at institutions B and C. On the other hand, institutions B and C should be able to publish their data in order to notify other institutions which data storage resources are accessible. The data from institutions B and C may be managed by different data-management systems in different formats, so the retrieved data must be transformed into a unified format and integrated into a single data set which can be interpreted by institution A's analysis process. The variety of data formats, models or schemas used by different institutions could create an interoperability issue.

Thus, while data-sharing is crucial to support cross-institutional collaborative operations, it needs to be highly managed. Security issues, such as authentication and authorization, must also be handled effectively.

In brief, the collaborative HIV/AIDS research scenario discussed above is intended to introduce a feasible way to facilitate such research. Multiple (HIV/AIDS research) institutions can share their data and participate in collaborative operations. The ultimate goal is building data-intensive analysis capabilities, based on aggregated (HIV/AIDS patients' demographic) data from multiple distributed institutions. Data from all involved institutions is integrated as a virtual data pool; similarly information-mining applications can constitute a virtual processing pool. Furthermore, the sharing relationship can be initiated among certain parties, accommodating new participants dynamically. Thus, the nature and composition of the virtual organization will vary over time. Obviously, the sharing relationship, therefore, has to be highly

controlled. The interoperability and security issues must be handled pre-emptively, for example.

## **1.2. Problem Statement**

Based on requirements exemplified by the HIV/AIDS collaborative research problem-context, can a secure distributed data warehouse system be proposed for providing the key capabilities required by such a scenario using the Grid paradigm? Can such a system a (Grid) system provide the required infrastructure and capabilities for data-sharing and data-based collaborative operations such as managed data-analysis methods executed over managed distributed resources?

A Grid system typically involves a complex environment comprising heterogeneous resources, dynamic administration mechanisms, and various security policies within different institutions. Due to the nature of the Grid environment, a proposed Grid-enabled, distributed data warehouse system needs to address wide-ranging problems, including: interoperability, resource virtualization, scalability, and security. This is the focus of this research.

## **1.3. Research Objectives**

The first objective is to provide justification for using OGSA-based Grid architecture and distributed data warehouse resources as the point-of-departure for this system. OGSA (Foster, Kesselman, Nick & Tuecke, 2002) stands for “Open Grid Services Architecture” which is a widely adopted Grid architecture for developing a Grid system. This objective contains two sub-objectives: the first is exploring all the essential capabilities specified in OGSA; the second is deriving (two) models from the defined capabilities for data-collection, data-access and integration.

The second objective is to examine particular relevant security concepts, including authentication, single sign-on, dynamic delegation and authorization for the proposed system. One sub-objective here, is to define a security model

that integrates all the identified components necessary to provide the overall required protection for the proposed system.

## **1.4. Methodology**

The methodology to be employed is primarily based on literature survey and theoretical argument. The literature study is divided into two parts: the first is a broad study of current Grid technology, including the concept, the essential Grid architecture, and Grid evolution. This study intends to give a comprehensive understanding of the Grid paradigm and related technologies.

The second part is the study of the OGSA specification and all its supporting standards and specifications, together with related implementations. OGSA is the fundamental standard for the proposed system framework. Around OGSA, there are a number of supporting standards and specifications that help define the standard interfaces and protocols for handling various aspects (e.g., interoperability, security, etc.) in an OGSA-based Grid system.

Three possible models are proposed and argued for as the solutions to the problems of realizing the key capabilities (including data-collection, data-access and integration) required by HIV/AIDS collaborative research, and addressing various security issues.

## **1.5. Layout of Dissertation**

This dissertation is organized as follows:

### **Chapter 1. Introduction**

The problem-context, the research objectives and the research methodology are introduced here.

### **Chapter 2. The Grid Approach to Cross-Institutional Collaboration**

A comprehensive introduction to the Grid is followed by a discussion of an approach combining Grid and database technology. In the area of Grid technology, the concept, architecture, characteristics and evolution are

discussed. Due to the weaknesses of current database technology in supporting distributed, heterogeneous data sources, the Grid approach is employed to enable data-access and integration across distributed institutions. Based on the classification of data in a Grid environment and categories of data-based distributed applications, integration strategies are discussed.

### **Chapter 3. Grid-Enabled Distributed Data Warehouse System**

Based on the discussion of the problem-context and approach, this chapter starts discussing the proposed distributed data warehouse system enabled by the Grid technology. The features of the envisaged environment are followed by a thorough analysis of the system's functional requirements.

### **Chapter 4. System Framework**

The chapter focuses on the framework for the proposed system according to the study of existing Grid-related standards and specifications. This framework recognizes all supportive standards and defines essential services to meet the requirements discussed in chapter 3. Two models are included to illustrate how to realize the main capabilities within the problem-context.

### **Chapter 5. Security**

This chapter firstly identifies the main security concerns in this dissertation. A broad study of the Grid security architecture, mechanisms, standards and implementations is followed by the discussion of solutions for each security aspect in the proposed system. A model is included to show how to combine all necessary security components to provide protection to the proposed system.

### **Chapter 6. Conclusion**

A summary of the findings of this research is followed by a discussion of possible future research.

## **Chapter 2.**

# **The Grid Approach to Cross-Institutional Collaboration**

The term ‘e-Science’ (<http://www.nesc.ac.uk/nesc/define.html>) is used to describe computationally intensive science that is carried out in highly distributed network environments, or science that uses immense data sets that require Grid computing. It was created by Dr. John Taylor, the Director General of the United Kingdom's Office of Science and Technology, in 1999 and was used to describe a large funding initiative started in November 2000. E-Science is essentially about global collaboration in key areas, including scientific, engineering and medical research, and the next generation of infrastructure that will enable it. One of its primary features is a new and systematic way of collecting, managing, sharing, publishing and exploiting large volumes of data. Another crucial feature is growing international, multi-disciplinary collaboration, which jointly addresses challenging problems. The e-Science approach requires new technological infrastructure as well as new behaviours. Existing database technology has been implemented extensively by distributed applications in both scientific and commercial areas. It is a determinant in planning and implementing data-access and integration. Grid technology has been adopted widely to enable efficient cross-institutional resource-sharing and collaboration. The combination of database and Grid technology introduces a suitable approach for building infrastructure in the e-Science environment. Therefore, the essential features of the Grid and database technology, and the approach combining them is introduced in this chapter.

## **2.1. Grid Technology**

Grid computing has emerged as a new field using multiple, distributed resources to cooperatively work on a single application. Grid concepts and technologies

were originally developed to enable resource-sharing within scientific collaboration, first within early gigabit/sec test-beds, and then on increasingly larger scales. Applications in this context include distributed computing for computationally demanding data analysis (pooling of computer power and storage), the federation of diverse distributed datasets, collaborative visualization of large scientific datasets (pooling of expertise), and coupling of scientific instruments with remote computers and archives. More recently, similar requirements arose in commercial settings, not only for scientific and technical computing applications, but also for commercially distributed ones, including enterprise-application integration and business-to-business partner collaboration over the Internet. Both science and industry can benefit from the Grid at present. In brief, the Grid approach coordinates resource-sharing and problem-solving across multiple organizations.

### **2.1.1. Grid definitions**

Grid is not a new idea. The concept of using multiple distributed resources to cooperatively work on a single application has been around for several decades. The word 'grid' is used by analogy with electric power grid, which provide pervasive access to electricity. The term 'the Grid' was coined in the mid-1990s to denote a proposed distributed computing infrastructure for advanced science and engineering. In 1998, Foster and Kesselman defined a computational Grid as, "a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities" (Foster & Kesselman, 1998). Before this, efforts to coordinate wide-area distributed resources were known as metacomputing (Smarr & Catlett, 1992). Metacomputing represents applications enabled by the construction of networked virtual supercomputers, or metacomputers. The term 'metacomputer' denotes a networked virtual supercomputer, constructed dynamically from geographically distributed resources linked by high-speed networks. An example of such a system is the I-WAY experiment (DeFanti, Foster, Papka, Stevens & Kuhfuss, 1996). Since the earliest definition, there have been a number of other attempts to define what a Grid is. For example, "A Grid is a software framework providing layers of services to access and

manage distributed hardware and software resources” (<http://www.extreme.indiana.edu/ccat/glossary.html>) or a “widely distributed network of high-performance computers, stored data, instruments, and collaboration environments shared across institutional boundaries” (<http://www.ipg.nasa.gov/ipgflat/aboutipg/glossary.html>). In 2001, the definition of a Grid was refined by Foster, Kesselman and Tuecke as: “coordinated resource-sharing and problem-solving in dynamic, multi-institutional virtual organizations” (Foster, Kesselman & Tuecke, 2001). This is the most commonly used definition at present. From a commercial view point, IBM (IBM Grid Computing, [http://www-1.ibm.com/grid/grid\\_literature.shtml](http://www-1.ibm.com/grid/grid_literature.shtml)) defines a Grid as “a standard-based application/resource-sharing architecture that makes it possible for heterogeneous systems and applications to share, compute and storage resources transparently”.

A simple checklist (Foster, 2002<sup>2</sup>) is used to help understand the essence of the Grid definition:

- Coordinating resources that are not subject to centralized control. A Grid integrates and coordinates resources and users that live within different control domains — for example: different administrative units of the same company; or different companies; and addresses the issues of security, policy, payment, membership, and so forth that arise in these settings.
- Using standard, open, general-purpose protocols and interfaces, a Grid is built from multi-purpose protocols and interfaces that address such fundamental issues as authentication, authorization, resource discovery, and resource access. These protocols and interfaces are standard and open.
- Delivering non-trivial qualities of service. A Grid allows its constituent resources to be used in a coordinated fashion to deliver various qualities of service, relating, for example, to response time, throughput, availability, and security, and/or co-allocation of multiple resource types to meet complex user demands, so that the utility of the combined system is significantly greater than that of the sum of its parts.

### **2.1.2. Virtual organization**

The Grid concept is about coordinated resource-sharing and problem-solving in dynamic, multi-institutional virtual organizations. A set of individuals and/or institutions, defined by such sharing rules, is called a virtual organization (VO) (Foster et al., 2001). The LHC (Large Hadron Collider) Computing Grid Project (LCG, <http://lcg.web.cern.ch/LCG/>) at European Council for Nuclear Research (CERN) is a classic example of where VOs are being used. A VO normally consists of a number of mutually untrusted participants with varying degrees of prior relationship (perhaps none at all) who want to share resources in order to perform some tasks.

The Grid is about resource-sharing. The sharing concerned is not primarily file exchange, but rather direct access to computers, software, data storage, sensors, networks and other resources, as required by a range of collaborative problem-solving and resource-brokering strategies emerging in industry, science, and engineering. This sharing is highly controlled. It is obviously always conditional and based on factors like trust, resource-based policies, negotiation and how payment should be considered. For example, the resource providers and consumers define clearly and carefully just what is shared, who is allowed to share, and the conditions under which sharing occur. Sharing relationships can vary dynamically over time, in terms of the resources involved, the nature of the access permitted, and the participants to whom access is permitted. These relationships do not necessarily involve an explicitly named set of individuals, but rather may be defined implicitly by the policies that govern access to resources. The dynamic nature of sharing relationships requires mechanisms for discovering and characterizing the nature of the relationships that exist at a particular point in time. Sharing relationships are often not simply client-server, but peer-to-peer: providers can be consumers, and sharing relationships can exist among any subset of participants. Sharing relationships may be combined to coordinate use across many resources, each owned by different organizations. The same resource may be used in different ways, depending on the restrictions placed on the sharing and its goals. The Grid also includes coordinated problem-solving,

which may need combinations of distributed data-analysis, computation and collaboration. These characteristics and requirements define what is called VO. The broad applicability of VO concepts makes the Grid paradigm important to modern computing.

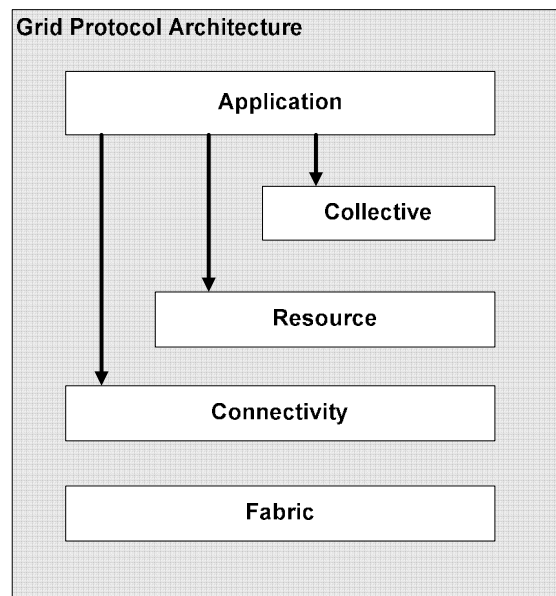
### **2.1.3. Grid architecture**

This section introduces a high-level Grid architecture (Foster et al., 2001) that identifies fundamental system components, specifies their purpose and function, and indicates how they interact with one another. It does not provide a complete enumeration of all required protocols (and services, APIs, and SDKs), but rather identifies requirements for general classes of components. It is an extendible, open architectural structure within which can be placed solutions to key VO requirements.

Interoperability is the central issue to be addressed. In a VO environment: the sharing relationship can be initiated among arbitrary parties, accommodating new participants dynamically, across different platforms, languages, and programming environments. In this context, the standard protocols and syntaxes for general resource-sharing are required to enable interoperability across organizational boundaries, operational policies and resource types. A protocol definition specifies how distributed system elements interact with one another in order to achieve a specified behaviour, and the structure of the information exchanged during this interaction. This focuses on externals (interactions) rather than internals (software, resource, characteristics), because VOs complement, rather than replace, existing institutions. Sharing mechanisms cannot require substantial changes to local policies and must allow individual institutions to maintain ultimate control over their own resources. A service is defined in terms of the protocol one uses to interact with it, and the behaviour expected in response to various protocol-message exchanges. The definition of standard services, i.e., access to computation and data, resource discovery, co-scheduling, data replication, and so forth, enhance the services offered to VO participants and also take out resource-specific details that would otherwise hinder the development of VO applications. Moreover, Application

Programming Interfaces (APIs) and Software Development Kits (SDKs) are also need to be considered. These provide the programming abstractions required to create a usable Grid. In brief, this approach to Grid architecture emphasizes the identification and definition of protocols and services firstly; and APIs and SDKs secondly.

As shown in Figure 2.1 (Foster et al., 2001), the components of this architecture are organized into layers. The description of this architecture is high level and places few constraints on design and implementation. Each layer is introduced as follows:



*Figure 2.1 Grid Architecture*

- **The Fabric layer** provides the resources to which shared access is mediated by Grid protocols: for example, computational resources, storage systems, catalogues, network resources, and sensors. A 'resource' may be a logical entity, such as a distributed file system, computer cluster, or distributed computer pool. Fabric components implement the local, resource-specific operations that occur in specific resources (whether physical or logical) as a result of sharing operations at higher levels. At a minimum, resources should implement enquiry mechanisms that permit discovery of their structure, state, and capabilities on the one hand, and resource management mechanisms that provide some controls of delivered

quality of service on the other.

- **The Connectivity layer** defines core communication and authentication protocols required for Grid-specific network transactions. Communication protocols enable the exchange of data between Fabric-layer resources. Communication requirements include transport, routing, and naming. The alternatives to these protocols are drawn from TCP/IP protocol stack: specifically, the Internet (IP and ICMP), transport (TCP, UDP), and application (DNS, OSPF, RSVP, etc.) layers of the Internet layered protocol architecture (Baker, 1995). The security aspects of the Connectivity layer should be addressed based on existing standards, whenever possible. As with communication, many of these security standards, which were developed within the context of the Internet protocol suite, are applicable. Authentication solutions for VO environments should have characteristics such as single sign-ons, delegation, and user-based trust-relationships (Butler et al., 2000) (Foster, Kesselman, Tsudik & Tuecke, 1998). They should integrate with, rather than replace local security solutions. Grid security solutions should also provide flexible support for communication protection (e.g., control over the degree of protection, independent data unit protection for unreliable protocols, and support for reliable transport protocols other than TCP) and enable stakeholder control over authorization decisions, including the ability to restrict the delegation of rights in various ways.
- **The Resource layer** builds on the Connectivity layer communication and authentication protocols in order to define protocols (and APIs and SDKs) for the secure negotiation, initiation, monitoring, control, accounting, and payment of sharing operations of individual resources. Resource-layer implementations of these protocols call on Fabric-layer functions to access and control local resources. Resource-layer protocols are concerned entirely with individual resources, and hence ignore issues of global state and atomic actions across distributed collections. Resource-layer protocols can be divided into two primary classes: information protocols and management protocols. Information protocols are used to obtain information about the structure and state of a resource, for example, its

configuration, current load, and usage policy. Management protocols are used to negotiate access to a shared resource, specifying, for example, resource requirements (including advanced reservation and quality of service) and the operations to be performed, such as process creation, or data-access. Since management protocols are responsible for instantiating sharing relationships; they must serve as a “policy application point”, ensuring that the requested protocol operations are consistent with the policy under which the resource is to be shared.

- While the Resource layer is focused on interactions with a single resource, **the Collective layer** contains protocols and services (and APIs and SDKs) that are not associated with any one specific resource, but are rather global in nature and capture interactions across collections of resources. The collective components can implement a wide variety of sharing behaviours without placing new requirements on the resources being shared, for example, directory services, co-allocation, scheduling, and brokering services, monitoring and diagnostics services, etc. Collective functions can be implemented as persistent services, with associated protocols, or as SDKs (with associated APIs), designed to be linked with applications. Collective components may be tailored to the requirements of a specific user community, VO, or application domain, for example, an SDK that implements an application-specific coherency protocol, or a co-reservation service for a specific set of network resources. Other collective components can be more general-purpose.
- **The Application layer** comprises the user applications that operate within a VO environment. Applications are constructed in terms of, and by calling upon, services defined at any layer.

In summary, it is a standards-based open architecture that facilitates extensibility, interoperability, portability, and code-sharing. It uses standard protocols to make it easy to define standard services that provide enhanced capabilities.

### **2.1.4. Standards-Bodies**

This section gives a brief overview of Grid standards-bodies. For Grid-related technologies, tools and utilities to be taken up widely by the community at large, it is vital that developers design their software to conform to the relevant standards. The most important standards organization is the Global Grid Forum (GGF, <http://www.ggf.org>). In September 11 2006, the GGF and the Enterprise Grid Alliance merged to form the Open Grid Forum (OGF, <http://www.ogf.org>).

OASIS (<http://www.oasis-open.org>) is a non-profit consortium that drives the development, convergence and adoption of e-business standards, which is having an increasing influence on Grid standards. Other bodies that are involved with related standards efforts are the Distributed Management Task Force (DMTF, <http://www.dmtf.org/>): Common Information Model (CIM) and the Web-based Enterprise Management (WBEM). In addition, the World Wide Web Consortium (W3C, <http://www.w3.org>) is also active in setting Web services standards, particularly those that relate to eXtensible Markup Language (XML) and Simple Object Access Protocol (SOAP).

### **2.1.5. Grid evolution**

The Grid technology is evolving at a very fast rate. At present, three generations of Grid systems are identified by De Roure (Eds.) (De Roure, Baker, Jennings & Shadbolt, 2003): the first-generation systems were the forerunners of the Grid; the second-generation systems focused on middleware to support large-scale data and computation; and the current third-generation systems shift the emphasis to distributed global collaboration, a service-oriented approach and information layer issues. As discussed in section 2.1.1, the commonly used definition of a Grid as “a flexible, secure, coordinated resource-sharing among dynamic collections of individuals, institutions, and resources”, emphasizes the importance of information aspects, essential for resource discovery and interoperability. Current Grid projects are beginning to take this further, from information to knowledge. These aspects of the Grid

are related to the evolution of Web technologies and standards, such as XML to support machine-to-machine communication and the Resource Description Framework (RDF, <http://www.w3.org/RDF/>) (W3C, 2004<sup>1</sup>) (W3C, 2004<sup>2</sup>) to represent interchangeable metadata. The Grid currently has a close relationship with World Wide Web. Evolving Web technology will provide the basis for the next generation of Grid systems.

#### ***2.1.5.1. The First Generation***

The first-generation systems were recognised as forerunners of Grids. The early to mid 1990s marked the emergence of the early metacomputing or Grid environments. Typically, the objective of these early metacomputing projects was to provide computational resources for a range of high-performance applications. These projects differed in many ways, but they all had to solve a number of similar problems, including communications, resource management, and the manipulation of remote data, to be able to work efficiently and effectively. Two representative projects in the vanguard of this type of technology were FAFNER (<http://www.npac.syr.edu/factoring.html>) and I-WAY (Foster, Geisler, Nickless, Smith & Tuecke, 1997). Both FAFNER and I-WAY were highly innovative and successful. FAFNER was tailored to a particular factoring application that was, in itself, trivially parallel and was not dependent on fast inter-connectivity. I-WAY was designed to cope with a range of diverse high-performance applications that typically needed a fast inter-connectivity and powerful resources. However, both of them lacked scalability.

#### ***2.1.5.2. The Second Generation***

The second-generation systems focus on middleware to support large scale data and computation. Middleware is generally considered to be the layer of software sandwiched between the operating system and applications, providing a variety of services required by an application to function correctly. Recently, middleware has re-emerged as a means of integrating software applications running in distributed heterogeneous environments. In a Grid, middleware is used to hide the heterogeneous nature and provide users and applications in a

homogeneous and seamless environment by providing a set of standardized interfaces to a variety of services. The key second-generation Grid technologies include core technologies, distributed object systems, resource brokers and schedulers, complete integrated systems and peer-to-peer systems.

There are growing numbers of Grid-related projects, such as Globus and Legion, dealing with areas such as infrastructure, key services, collaborations, specific applications and domain portals. These projects represent the core technologies of the second generation. The Globus project is a multi-institutional research effort that seeks to enable the construction of computational Grids. Globus (Foster & Kesselman, 1997) provides a software infrastructure that enables applications to handle distributed heterogeneous computing resources as a single virtual machine. A central element of the Globus system is the Globus Toolkit (GT), which defines the basic services and capabilities required to construct a computational Grid (Foster & Kesselman, 1999). The toolkit consists of a set of components that implement basic services, such as security, resource location, resource management, and communications. Globus has evolved from its original, first-generation incarnation as I-WAY, through version 1 (GT1) to version 2 (GT2) (Foster & Kesselman, 1998). The protocols and services that Globus provide have changed as it has evolved, and its emphasis has moved away from supporting high-performance applications towards more pervasive services that can support virtual organizations. Legion (Grimshaw et al., 1997) is an object-based metasystem which encapsulated all of its components as objects. This method has all the normal advantages of an object-oriented approach, such as data-abstraction, encapsulation, inheritance and polymorphism. It provides the software infrastructure so that a system of heterogeneous, geographically distributed, high-performance machines can interact seamlessly.

Earlier representatives of the distributed object system include the Common Object Request Broker Architecture (CORBA, <http://www.corba.org>) and Jini (<http://www.jini.org>). CORBA automates many common network programming tasks, such as object registration, location, and activation; request de-multiplexing; framing and error-handling; parameter marshalling and de-marshalling; and operation dispatching. Although CORBA provides a rich

set of services, it does not contain Grid-level allocation and scheduling services. Jini has been designed to provide a software infrastructure that can form a distributed computing environment that offers network plug and play. In Jini, applications are normally written in Java and communicated using the Java Remote Method Invocation (RMI) mechanism. The Common Component Architecture Forum (Armstrong et al., 1999) attempts to define a minimal set of standard features that a high-performance component framework would need to provide in order to use components developed within different frameworks.

There are a number of systems available whose primary focus is batching, resources scheduling and resource brokering. Condor (<http://www.cs.wisc.edu/condor/>) is a software package for executing batch jobs on a variety of UNIX platforms, in particular those that would otherwise be idle. The major features of Condor are automatic resource location and job allocation, check pointing, and the migration of processes. The Storage Resource Broker (SRB) (Rajasekar & Moore, 2001) has been developed at San Diego Supercomputer Centre (SDSC) to provide uniform access to distributed storage across a range of storage devices, via a well-defined API. A key feature of the SRB is that it supports metadata associated with a distributed file system, such as location, size and creation date information. It also supports the notion of application-level (or domain-dependent) metadata, specific to the content, which cannot be generalized across all data sets. Nimrod-G is a Grid broker that performs resource-management and scheduling of parameter-sweep and task-farming applications (Buyya, Abramson & Giddy, 2000). The Nimrod-G scheduler has the ability to lease Grid resources and services depending on their capability, cost, and availability. It also supports resource discovery, selection, scheduling, and the execution of user jobs on remote resources.

As the second generation of Grid components emerged, a number of international groups started projects that connected these components into coherent systems, called integrated systems. The European DataGrid project (<http://eu-datagrid.web.cern.ch/>), led by CERN, is funded by the European Union with the aim of setting up a computational and data-intensive Grid of resources for the analysis of data coming from scientific exploration (Hoschek,

Jaen-Martinez, Samar, Stockinger & Stockinger, 2000). The DataGrid is built on top of the GT. The primary objective of the DataGrid project is implementing middleware for fabric and Grid management, including the evaluation, test, and integration of existing middleware and research and development of new software as appropriate. Uniform Interface to Computing Resources (UNICORE) (Almond & Snelling, 1999) is another important integrated system that needs to be mentioned. It is a project funded by the German Ministry of Education and Research. The design goals of UNICORE include a uniform and easy-to-use Graphical User Interface (GUI), an open architecture based on the concept of an abstract job, consistent security architecture, minimal interference with local administrative procedures, and exploitation of existing and emerging technologies through standard Java and Web technologies. UNICORE provides an interface for job preparation and secure submission to distributed supercomputer resources.

The traditional client-server model can be a performance bottleneck and a single point-of-failure. Peer-to-Peer (P2P) computing (Clark, 2001) implemented by several systems are examples of the more general computational structures that are taking advantage of globally distributed resources. In P2P computing, machines share data and resources, such as spare computing cycles and storage capacity, via the Internet or private networks. Machines can also communicate directly and manage computing tasks without using central servers. This permits P2P computing to scale more effectively than traditional client-server systems. Project JXTA (<http://www.jxta.org/>), created by Sun Microsystems, is an open-source development community for P2P infrastructure and applications. JXTA is a specification, rather than software.

### ***2.1.5.3. The Third Generation***

Current third-generation systems shift emphasis to distributed global collaboration, a service-oriented approach and information layer issues. New Grid applications desire to be able to reuse existing components and information resources, and to assemble these components in a flexible manner. The adoption of a service-oriented model and increasing attention to metadata are

two key characteristics of third-generation systems. The third generation is a more holistic view of Grid computing and can be said to address the infrastructure for e-Science. By 2001, a number of Grid architectures were apparent in a variety of projects, for example, the one introduced in section 2.1.3. Around this time, the emergence and wide adoption of Web services technology (W3C, 2004<sup>3</sup>) introduced promising standards to support the service-oriented approach. In fact, one research community, focussing on agent-based computing, had already undertaken extensive work in this area: software agents can be seen as producers, consumers and indeed, brokers of services. Web services and software-agent technologies are closely aligned to the third-generation Grid at present.

“The Open Grid Services Architecture (OGSA) Framework”, the Globus-IBM vision for the convergence of Web services and Grid computing, was presented at the GGF meeting held in Toronto in February 2002. It was initially described as a “physiology” paper (Foster et al., 2002). It introduced a service-oriented Grid architecture which tailors a Web services approach to meet some Grid-specific requirements. The OGSA supports the creation, maintenance, and application of ensembles of services maintained by VOs. Here, a service is defined as a network-enabled entity that provides some capability, such as computational resources, storage resources, networks, programs and databases.

Web services provide interoperability to Grid computing. However, they do not provide a new solution for many of the challenges of large-scale distributed systems. Agent-based computing (Jennings, 2001) (Foster, 2004) is suggested as another input to inform the service-oriented Grid vision. An agent “is an encapsulated computer system that is situated in some environment, and that is capable of flexible, autonomous action in that environment in order to meet its design objectives” (Wooldridge, 1997). Agent-based computing is particularly well suited to a dynamically changing environment, where the autonomy of agents enables the computation to adapt to changing circumstances. The Foundation for Intelligent Physical Agents (FIPA, <http://www.fipa.org>) is an international organization, dedicated to promoting the commercial application

of intelligent-agent technology, by openly developing specifications supporting interoperability for agents and agent-based services. FIPA produces software standards for heterogeneous and interacting agents and agent-based systems, including extensive specifications. In the FIPA abstract architecture, agents communicate by exchanging messages using speech act-based FIPA Agent Communication Language (FIPA-ACL) (FIPA, 2002) to promote interoperability. FIPA-ACL is a language with precisely defined syntax, semantics and pragmatics that is the basis of communication between independently designed and developed software agents.

## **2.2. Combining Database and Grid Technology**

Key elements of e-Science are in-silico experimentation and design, and information-based discovery through data-mining and visualization of integrated, large data-collections. The maturity of database technology means that it is used extensively in applications, including virtually all of those in the e-Science domain, and in building the infrastructure itself. However, the integration of large-size data-sets, the geographic distribution of users and resources, and computationally-intensive analyses result in complex and stringent performance demands not being satisfied by any existing data-management infrastructure. The Grid approach provides a platform that potentially enables this integration.

Combining Grid technology with database access and integration technology is an essential approach to meeting those infrastructure requirements and to supporting the evolving behaviours. The infrastructure should provide the technology required to enable e-Scientists to make effective and convenient use of structured data. The data-access and integration are primary targets to be considered for two reasons. Firstly, a great many Grid applications include a significant data-access and integration requirement. Virtually every scientific, engineering, medical and decision-support application depends on accessing distributed heterogeneous collections of structured data. Secondly, the Grid itself uses many structured data-collections for its own operation and administration. As the Grid technology becomes more sophisticated and

autonomic, the number, volume and diversity of these collections will increase. It is, therefore, imperative that Grid designers and developers support and use systematic data-access and integration methods.

This section, firstly, reviews the current data-management technologies and limitations, and then introduces the approaches that have been adopted to support integration of data-intensive services, according to the scope of applications and the variety of structured data.

## **2.2.1. Current database-management technologies**

Data management technologies, such as Database Management Systems (DBMSs), have been studied for several decades, and are well established. The Grid introduces new challenges like scale, heterogeneity, distribution, and autonomy. However, current data-management software has already addressed these challenges to some extent, especially heterogeneity and distribution transparency. This section reviews current database-management technologies. A good database text (Elmasri & Navathe, 2000) provides more related information.

### ***2.2.1.1. Basic Concepts***

A database is defined as an organized collection of data. DBMSs are frequently chosen to provide a higher-level interface to manage structured data. A schema is metadata that describes the logical structure of a database in terms of some data model, such as the relational model. There may be additional metadata available, describing the physical organization of the database, the access policies and some of its enforced invariants, called integrity constraints. Notations for describing subsets of data are called query languages. These have been extended to include object-oriented features and to allow embedded programming language expressions, such as Java static methods. The term ‘query language’ also includes those parts of the language used to perform updates. A view mechanism represents the state of a subset of a database, specified as a query. Views are used to give application developers a controlled and simplified view of the data, appropriate to their task.

Materialised views avoid the repeated evaluation of the query defining the view. Updates are propagated to the view by a consistency protocol that may be driven by triggers.

### ***2.2.1.2. Single-Site DBMSs***

Traditionally, DBMSs have focused on structured data, laid out in tabular form. In recent years, they have been extended to provide support for non-tabular data, including large objects, abstract data types, user-defined functions, and so on (Stonebraker & Moore, 1996). The organization of data held in a DBMS tends to be highly structured, and the logical groupings must conform to a defined data schema. The grouping can be based on fields, records and files. In a relational database management system (RDBMS), the grouping is based on columns, rows, and tables, and in an object-oriented database (OODBMS), it is based on objects and classes. XML (W3C, 2006<sup>1</sup>) has now become popular, not only as a markup language for data exchange, but also as a data format for semi-structured data. Therefore DBMSs have been adapted to store XML data, either shredded into relational rows (without the markup), or stored intact in the tables (with the markup). XML offers great flexibility in how the structure, and the syntactical and semantic rules of data are conveyed, and how structural links and non-linear pathways between data can be defined. A new query language for XML, called XQuery (W3C, 2006<sup>6</sup>), is under design in the W3C. Prototype implementations for XQuery are gradually becoming available, including implementations that query over XML views of relational data (Funderburk, Kiernan, Shanmugasundaram, Shekita & Wei, 2002).

DBMSs provide a high-level interface to manage structured data. This interface virtualizes the details of the physical data organization, and provides applications with a high level, declarative query language, SQL (Structured Query Language). Besides declarative access, DBMSs also provide several business-critical features such as transactional updates, data integrity, reliability, availability, high performance, concurrency, parallelism, and replication. They automatically control access to data, manage the referential integrity of data within transactions, log changes made to data, audit database activity,

synchronise data replicated in a distributed environment, and recover data to a consistent state. Applications only need to specify what task they want to perform, and do not have to program how the task is to be performed. DBMSs also provide facilities to reorganize database contents in order to manage space efficiently, to optimize queries, and to balance available resources dynamically. For example, the DBMS optimizer automatically searches among several possible implementations and chooses the best one. Most DBMSs support the specification of detailed authorization and privacy mechanisms.

The efficient evaluation of statements in query languages is well developed, and includes sophisticated data organizations, such as caching, indexes, fragmentation and optimising transformations. This optimization has been extended to distributed-queries necessary for integrating distributed resources (Kossmann, 2000). The development of transaction mechanisms that deliver atomicity, consistency, isolation and durability, has greatly simplified the task of writing applications. More complex transaction mechanisms support larger and longer-lived operations. Distributed transactions are supported by standard two-phase commit protocols, e.g., Web Services Transaction (WS-Transaction) (Cabrera et al., 2005). Standard mechanisms for connecting programs to data typically establish a session, and then submit a series of query statements within some transaction regimes. Each submission obtains a response, either a result-set or a status-response, indicating whether the execution has succeeded or has failed. DBMSs provide schema evolution, representation of complex data, data-sharing, caching, effective management of data resources, backup, and recovery. Schema evolution is the process of changing the description of the data and how it may be used. As it is a common operation, DBMSs support it and deal with the implicit changes required to make the existing data comply with the new definition. DBMSs also support efficiently implemented triggers: assertions or integrity constraints, paired with actions, that fire, e.g., cause a transaction to abort, compensation operations, or notifications, when they are triggered by an update.

Using standard mechanisms, such as Open Database Connectivity (ODBC) and Java Database Connectivity (JDBC), the application program is independent of

some aspects of the database to which it connects, and remote connection is supported. ODBC and JDBC (SUN, 2002) are the most widely used programmatic interfaces to access SQL DBMSs. They provide support for a variety of database operations, including SQL query specification and execution, transaction processing, etc. The popularity of Web services as a method for programs to communicate with one another has introduced another interface to relational DBMSs. It is now possible for Web service clients to issue SQL requests and to invoke database stored-procedures (e.g., Web Services Object Runtime Framework (IBM, 2001)). ADO.NET (<http://www.microsoft.com/data/ado/default.htm>) is an application-level interface to Microsoft's OLE DB data sources. Derived from the earlier ActiveX Data Objects (ADO), it enables applications to query databases using SQL, access information in a file store over the Internet, access email systems, save data from a database into an XML file, and perform transactional database operations. ADO.NET also supports a disconnected mode of operation, where clients can work on cached copies (datasets) of prior query results without a connection to the data source. However, all these interfaces still have further limitations with respect to virtualization. JDBC and ODBC provide some heterogeneity transparency, handling SQL DBMSs and a few other simple data types (like comma-separated-value files). They do not handle many kinds of data sources well, especially ones without database-like query and transaction interfaces.

### ***2.2.1.3. Federated DBMSs***

To achieve performance and dependability, and to deliver multi-site data services, many DBMSs are able to exploit high levels of parallelism and to operate on distributed computers. A distributed database is one that has been deliberately distributed over a number of sites, is designed as a whole (Stonebraker, Aoki et al., 1996), and is subject to considerable centralised control – local DBMSs may have different physical schemas even when there is a shared global schema (e.g., there may be explicit fragmentation schemas).

The data management community has developed federated database technology, which provides unified access to diverse and distributed data (Sheth & Larson, 1990). In a federated database, many databases contribute data and resources to a multi-database federation, but each participant has full local autonomy. In a loosely-coupled federated database, the schemas of the participating databases remain distinguishable, whereas in a tightly-coupled federated database, a global schema hides (to a greater or lesser extent) schematic and semantic differences between resources (McBrien & Poulouvasilis, 2002) – single, logical schema mapped to multiple physical schemas. Based on these and other more detailed database achievements, there are products that provide access to and integration of distributed data resources, e.g., IBM DiscoveryLink (Haas et al., 2001) and Kliesli or K2 (Davidson et al., 2001). The WebSphere Information Integrator (Lee, Magowan, Dantressangle & Bannwart, 2005) is the IBM information-integration middleware that provides a range of integration technologies: enterprise-search, data-federation, replication, transformation, and event-publishing, to meet varied integration requirements.

In a federated architecture, a federated DBMS serves as middleware, providing transparent access to a number of heterogeneous, distributed data sources. A federated system appears to the application developer like a regular DBMS. Applications can use any supported interface (including ODBC, JDBC, or a Web service client) to interact with the federated DBMS. A user can run queries to access data from multiple sources, joining and restricting, aggregating and analyzing it at will, with the full power of a DBMS query language like SQL or XQuery. A user can also update the data, if they have the right permissions at the sources. Yet, unlike with JDBC, ODBC, or ADO, the data sources in a Grid-based federated system need not be DBMSs at all, but, in fact, could be anything, ranging from sensors to flat files to application programs to XML, and so on.

An ideal federated DBMS provides good heterogeneity transparency and some distribution transparency. A key limitation is that applications have to explicitly specify the data sources in a federated query. This means that the addition of new data sources can involve changing the application, typically a

very expensive task. Each data source must also be explicitly registered to the federated DBMS, along with its wrapper.

#### ***2.2.1.4. Security***

DBMSs currently handle security on a per-data source basis. Data security includes authentication, authorization and auditing. Current DBMSs systems offer various “sign-on” mechanisms to identify the user. The simplest of these mechanisms rely on trust; the DBMS assumes that the application above the DBMS authenticates the user. A slightly more robust authentication mechanism allows the application to provide a user ID and password, either as plain text or encrypted. Some DBMSs also support more sophisticated schemes, like Kerberos, which uses encrypted authentication tickets. Once a user is authenticated, a DBMS must determine what data they may access, and what tasks (query, update, create table, etc.) they may perform. SQL DBMSs use “GRANT” and “REVOKE” statements to maintain permissions internally. Authorization can be granted at the level of a single user or a group and can also be managed by an external security service. An advantage of external security services is that they can easily control access to multiple DBMS systems, thus reducing the overhead of managing them. Most DBMS systems support auditing of data-access to verify that authorizations are being enforced correctly. Audit records can log who accessed the data, and what they did, and allow the detection of unauthorized access. The main limitation of security support in current systems lies in security across multiple sources. Authentication, authorization, and auditing are typically enforced and managed separately for each data source, even for the same user. This results in a loss of ownership transparency, because applications have to separately handle security with each data source.

### **2.2.2. Data life-cycle classification**

Data in a Grid environment can be classified in different ways. No attempt was made in the analysis exercise to distinguish between data, information, and knowledge when identifying requirements, on the basis that one worker’s

knowledge can be another worker's information or data. However, a distinction can be drawn between each stage in the data life cycle that reflects how data-access and operations vary. This classification is based on a combination of creation, purpose, and usage. The types of data in this classification are: raw data, reference data, processed data, results data, derived data and metadata.

- **Raw data** is created and put out from a data source, either as an instrument or an application program, for example, meteorological data from remote sensors or astrophysical data from optical surveys. The structure and format of raw data are determined by the data source. Instrument data sources tend to create raw data with limited value until processing has taken place. A raw data set is characterized by being read-only, and is normally accessed sequentially. It may be repeatedly reprocessed and is commonly archived once processing is complete. Therefore, the Grid needs to provide the ability to secure this type of data off-line and to restore it back on-line.
- **Reference data** is frequently used in processing raw data, when transforming it, as control data in simulation modelling, and when analyzing, annotating, and interpreting data, for example, the constants for a map projection or a coordinate system. Common types of reference data include: standardized and user defined coding systems, parameters and constants, and units of measure. By definition, most types of reference data seldom change. A feature of all types of reference data is that their values remain static. Although classification systems and ontology are defined as metadata, their shorthand codes are a type of reference data.
- **Processed Data.** Almost all raw data sets undergo processing to apply necessary corrections, calibrations, and transformations. Producing processed data sets may involve filtering operations to remove data that fails to meet the required level of quality or integrity, and data that does not fall into a required specification tolerance. Conversely, it may include merging and aggregation operations with data from other sources. Therefore, the Grid must maintain the integrity of data in multi-staged processing, and should enable check-pointing and recovery to a point in time, in the event of failure. It should also provide support to control processing through the

definition of workflows and pipelines, and enable operations to be optimized through parallelization.

- **Result data** is created as the output of a data retrieval or interrogation operation, normally within an application, or when examining data content during the discovery process. Result data sets are subsets of one or more database that match a set of pre-defined conditions. Typically, a result data set is extracted from a database for the purpose of subjecting it to focused analysis and interpretation. The types of application functionality that produce result-data output include algorithmic analysis, simulation, and data transformation. It may be a statistical sample of a very large data resource that cannot be feasibly analyzed in its entirety, or it may be a subset of the data with specific characteristics, e.g., gene-expression data. A result data set may also be used as input data for a simulation run in a modelling application. It may also be a set reference data for a visualization application, e.g., map projection reference data, and oceanographic data for a regional study. Users may choose to create a copy of the result data and retain it locally for reasons of performance or availability.
- **Derived data** sets are created from other existing processed, result, or other derived data. Statistical parameters, summarizations, and aggregations are all types of derived data that are important in describing data, and in analyzing trends and correlations. There are two main ways to create derived data. The first is by performing statistical analysis on other data to create statistical parameters, summaries and aggregations. This type can be considered to be a form of results data, and it is particularly important when analyzing trends and correlations in data. The second way to produce derived data is through recording observations on, or drawing inferences from, other data, or any subject (e.g., image, scientific sample or environment) under investigation. An observation is an annotation, or a description, of the features, properties or behaviour of data, experiments, or subjects. An inference is a deduction or conclusion drawn from analysis and interpretation. Inferences add to understanding and knowledge, and they are used to explain correlations, trends and anomalies. They may also include evidential reasoning. An important feature of derived data

created during analysis and interpretation is volatility. Data can change as understanding evolves, and as hypotheses are refined over the course of study. Equally, derived data may not always be definitive, particularly in a collaborative work environment. For this reason, it is important that the Grid provides the ability to maintain personalised and multiple versions of inference data.

### **2.2.3. Structured data, sources and resources**

This section clarifies several terms used in this dissertation. These terms are “imported” from database technology and are critical to data manipulation in the Grid environment.

The term ‘structured data’ is used to denote any data whose structure is explicitly defined, so that operations may exploit that structure, e.g., queries may extract subsets of data based on it. Typically, this is relationally structured data or an XML document. However, it may be binary data, for which there is an explicit structure definition, and software that exploits that structure. One of the goals of data-access and integration is to treat all of this data, whatever its origin, within a uniform framework (Abiteboul, Buneman & Suciu, 1999).

A ‘data source’ denotes any facilities, including instruments, devices, or application programs that create and output structured data. A data source need not be connected to the Grid infrastructure, or defined in a Grid environment in order to do this. The Grid must provide the ability to capture output data directly from a data source that is connected to a Grid environment, and it must provide the ability to import output from a data source that is not connected. It must also have the ability to integrate output with existing data in a Grid environment, e.g., an historical processed output, and make the combined result data available for pseudo-real time analysis and interpretation. This type of ability is necessary to enable dynamic monitoring and control of other types of Grid resources.

A ‘data resource’ is a persistent data store held in either file structures or in

DBMSs in a Grid environment. It has an owner and a name, and is stored at a physical location in a file system or in a raw device. There are no constraints over type, structure, volume, or status of the content a data resource can hold. A data resource may conform to an agreed standard, or be totally owner-defined. In a Grid environment, a data resource should be readable by someone other than the owner. Users can be granted privileges to read all or part of its contents, to create new data, and to modify and delete data content. The Grid must support any type of data resource, such as relational database, XML database or file system. It must also provide the ability for data owners and custodians to manage data resources online. Data resource replication across multiple sites must also be provided in order to satisfy the service level requirements identified for a Grid environment.

#### **2.2.4. Categories of structured data and its applications**

Five categories of structured data illustrate diversity and prevalence. This diversity needs to be considered in approaches to data-access and integration, primarily because of the many different usage patterns.

- **Primary structured data** is usually used for recording observations by scientists and instruments. Direct recording in databases also supports many other forms of data-collection, from a field-worker's personal record-keeping to the output of highly automated laboratories.
- **Ancillary data** represents the structured metadata that is used to support bulk binary or structured data. Several forms of structured metadata are used for different purposes. Technical metadata is used to support management technology that underpins data Grid operations, and also used to organize the interpretation of primary data. Application metadata guides the interpretation of the primary data. Data products, including summaries, catalogues and indexes that are produced by successive steps in deriving information from the primary data, may also be treated as primary data by some scientists and as metadata by others.
- **Collaboration data** is that collected to enable scientific information to be shared quickly and precisely. Scientists increasingly collaborate by

recording and sharing data via databases, normally using agreed terminologies. Groups of scientists have communicated by writing to publicly available structured data-collections. This leads to many collections of related data, each curated by a particular group, who impose standards and structure. Mechanisms have emerged for others to annotate some of the collections in independently stored databases that refer to data in curated collections.

- **Personal data** is assembled by, or about, individual users. It is, typically, private and may contain profile data, such as preferences and re-usable working methods, digital laboratory notebooks, representations of work in progress and personalised workflow scripts.
- **Service data** is used to provide a Grid, data-access and integration infrastructure and other e-Science-enabling technology, for example, data in registries, that describing services, defining authorization policies, describing progresses, enacting work flows and defining the current states of a system. Current systems include: the Grid Information Services (Dinda & Plale, 2001) and the Spitfire (Bell et al., 2002), which use relational storage to provide database administration, user management, data-dictionary access and statement evaluation.

In addition to the variety of structured data, the categories of applications also need to be considered to understand the requirements more clearly. For example, each e-Science application needs different combinations of data from multiple data resources. The application programs may retrieve data from data resources held at different sites, and the result of queries may be set as part of the workflow organizing the whole computation. Moreover, new discoveries become possible as scientists mine correlations and anomalies from multiple sources. This is becoming a major modus operandi for scientific collaboration (Pearson, 2002). More and more data will be recorded and organized as structured resources, available to large communities of scientists from many disciplines. More and more investigations will involve accessing subsets of these resources, extracting specific data from each, and using it in combination to test or develop some or other scientific models.

### 2.2.5. Integration strategies

Three integration strategies were identified based on the categories of structured data and applications (Atkinson et al., 2004).

- **The virtual database** introduces the ideal approach. It allows a set of databases to be presented as a single, integrated view with a single, federated schema, so that users then use them directly, unaware of the separate databases behind the view. The concept of a virtual database introduces transparencies that should be considered when designing any integration schema.
- **Bespoke integration.** Many of scientific projects combining multiple data resources depend on the application users developing their own integration systems, including queries, programs, data flows and workflows, specific to a particular project. The approach, that partitions the integration task into queries, program execution construction of intermediate data resources, explicit data transfers and transformations, and updates to data resources holding data products, requires much investment in designing, implementation, tuning, maintaining and operating. The Grid provides mechanisms for introspection of resources and services to support tools, dynamic adaptation and operation validation. It also enables query and result delivery, program and data transport, scheduling of the stages of the evaluations, and progress monitoring. These Grid functions will be used by the individual generic data-access and integration components and explicitly by code, written by the application developers.
- **Incremental integration.** A virtual database is the ideal goal. It is unlikely that the ideal will ever satisfy all requirements. Hence, scientists often take creative steps to combine data in new ways that involve encoding the science. In such cases, the arrangement for processing large volumes of data, at the limits of available computational resources, will be developed and negotiated. However, much scientific, diagnostic and analytic work requires repetitive routines. These repetitive processes generally operate in a relatively stable context with pre-chosen, mostly read-only data and established workflows. Such cases are supported today by products such

as IBM DiscoveryLink.

## **2.3. Summary**

This chapter gives a comprehensive introduction of the Grid technology, including its concept, architecture, evolution and the state-of-the-art. It can be used with federated database systems to enable data-access and integration across multiple distributed institutions in a virtualized, transparent manner. The combination of database technology and Grid technology thus introduces a suitable approach for realizing a VO, as depicted in the HIV/AIDS collaborative research problem-context. Database concepts discussed in this chapter will be extrapolated into the use of data warehousing, which will be discussed in the next chapter.

## **Chapter 3.**

# **Grid-Enabled Distributed Data-Warehouse Systems**

As illustrated earlier, the Grid concerns coordinated resource-sharing and problem-solving in dynamic, multi-institutional VOs. The combination of database and the Grid technology introduces a great opportunity to provide cross-institutional data-access and integration for multi-institutional collaboration and problem-solving within a specific problem domain. In the example of HIV/AIDS collaborative research described in chapter 1, data-collection from distributed data-providers, data analysis and sharing across multiple institutions are three key elements to realize collaborative HIV/AIDS research. This problem domain will be used to elucidate important concepts in this chapter.

Building data-analysis and data-mining methods (Pyle, 2003), (Groth, 1997), (Westphal & Blaxton, 1998) over data warehouses is a traditional approach to aggregate data and supply data-intensive analysis ability within one institution's boundary. In the context of HIV/AIDS collaborative research, it is imperative to build data warehouses for storing HIV/AIDS patients' (demographic) data collected from medical institutions, and to develop data-mining and data-analysis methods over established data warehouses. Data warehouses are essentially databases. Compared to on-line operational databases, data warehouses contain aggregated data which are not changed frequently. They represent a most suitable data source for information-mining and analysis operations.

This dissertation proposes a Grid-enabled, distributed data-warehouse system designed to facilitate HIV/AIDS collaborative research. It uses the Grid technology, combined with multiple data warehouses, to realize all required capabilities required by this type of research. Data warehouses within different institutions are primary data storage resources, which are used for different

purpose compared to on-line operational databases. Institutions may also have other kind of data storage resources, such as single databases and data marts. They are all essentially databases that may allow external institutions access conditionally. In this system, either data warehouses or data marts are all considered as data resources, which may be available from a variety of infrastructures and in a range of formats. These databases and data-analysis services construct the resource fabric of the proposed system. For building a data warehouse through data-collection, the Grid is used to operate the workflows that populate the data warehouse. For data-sharing, the Grid provides access to data warehouses or data marts. Other Grid capabilities include coordinating the data-sharing and handling interoperability and security challenges in the system. Further capabilities include resource-discovery and management, job scheduling, metadata management, authentication, authorization, etc.

The main concerns of this dissertation are to define a workable and feasible framework in-the-large, and to address security issues within the pre-defined VO environment, by using existing standards, specification and technologies. The envisaged framework contains a rich set of components for providing different capabilities, such as data-collection, access, integration, authentication, and authorization etc., according to the requirements of the proposed system. Identifying specific data-mining and data-analysis methods to be used is beyond the scope of this dissertation. In this chapter, the background of data warehouse and data/information-mining will be introduced, initially. The proposed VO environment, and the requirements of the Grid-enabled, distributed data warehouse system will be described in detail subsequently.

### **3.1. Overview of Data Warehousing and Data-mining**

A data warehouse is a subject-oriented, integrated, non-volatile, and time-variant collection of data in support of management's decision (Inmon, 1996). It contains granular corporate data (Inmon & Hackathorn, 1994). It is a logical collection of information, gathered from many different operational

databases, used to create business intelligence that supports business-analysis activities and decision-making tasks. Primarily, it is a record of an enterprise's past transactional and operational information, stored in a database designed to favour efficient data analysis and reporting. Basically, a data warehouse contains the aggregation of a company's data that is prepared to support data-based analysis operations.

Extract, transform, and load (ETL) is a process in data warehousing that involves extracting data from outside sources, transforming it to fit business needs, and ultimately loading it into a data warehouse. The first part of an ETL process is to extract data from different source systems. Common data source formats are relational database and flat files, but may include non-relational database structures. Extraction converts the data into a format for transformation processing. The transformation phase applies a series of rules or functions to the extracted data to derive the data to be loaded. The load phase loads the data into a data warehouse. Depending on the requirements of an organization, this process varies widely. Some data warehouses merely overwrite old information with new data. More complex systems can maintain a history and audit trail of all changes to the data. While an ETL process can be created using almost any programming language, creating one from scratch is quite complex. Increasingly, companies are buying ETL tools to help in the creation of ETL processes. A good ETL tool (e.g., SQL Server Integration Services) must be able to communicate with the many different relational databases and read the various file formats used throughout an organization.

Data-mining is fully integrated with a data warehouse and flexible interactive business analysis tools. Many definitions of data-mining exist. Hand, Mannila and Smyth define it as, "The analysis of (often large) observational data sets to find unsuspected relationships and to summarize the data in novel ways that are both understandable and useful to the data owner" (Hand, Mannila & Smyth, 2001); it can also be defined as: "The nontrivial extraction of implicit, previously unknown, and potentially useful information from data" (Frawley, Piatetsky-Shapiro & Matheus, 1992). The essence of data-mining is the

finding of structure in data. It primarily provides these facilities:

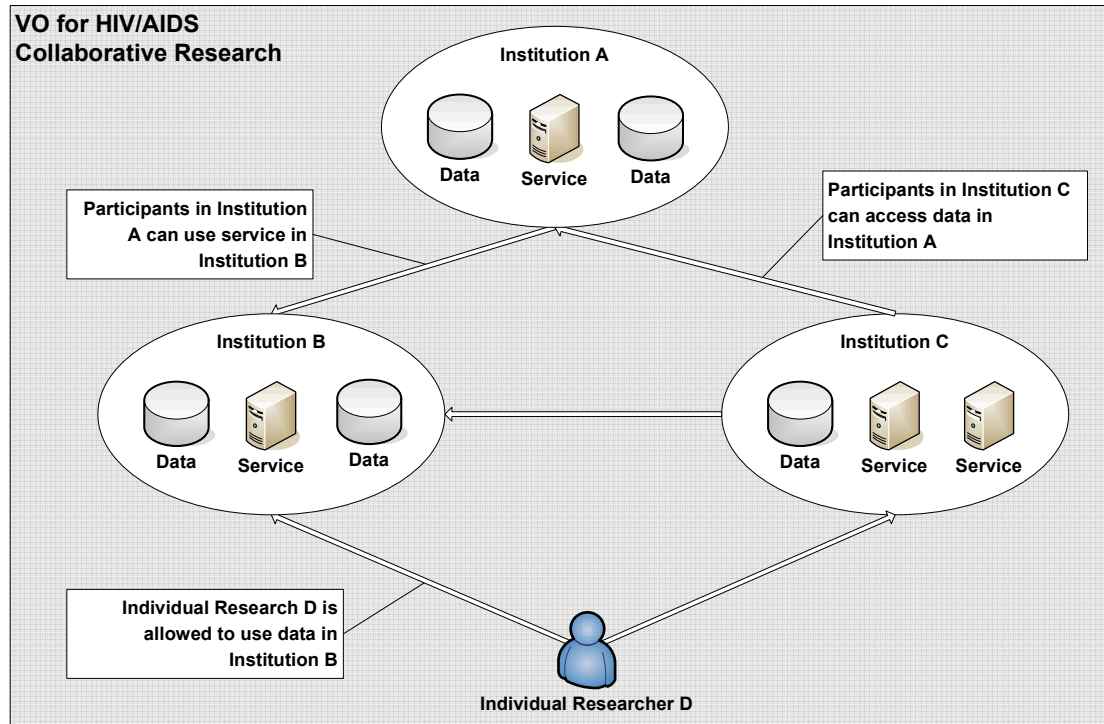
- Automated prediction of trends and behaviours. Data-mining automatically finds predictive information in large databases (e.g., linear regression makes predictions for all input values).
- Automated discovery of previously unknown patterns. A pattern refers only to the restricted regions of space spanned by the variables. Data-mining tools search databases and identify previously hidden patterns in one step.

Data-mining tools (e.g., SQL Server Analysis Services) can analyze massive databases in minutes, if these tools are implemented on high-performance, parallel-processing systems. Faster processing allows users to automatically experiment with more models to understand complex data.

### **3.2. Description of Example VO**

Collaborative HIV/AIDS research introduces a dynamic environment that consists of multiple, physically distributed institutions and/or individual researchers. The proposed Grid-enabled, distributed data warehouse system – for this problem domain - is essentially a Grid system intended to provide infrastructure-level capabilities to facilitate data-collection, data-sharing across multiple boundaries, and handle issues such as interoperability or security. All involved institutions participate in a VO that integrates all accessible heterogeneous data resources and users. Figure 3.1 illustrates the envisaged VO environment. The participants could be either HIV/AIDS institutions or individual HIV/AIDS researchers. The participant institutions are considered as resource providers, and, at the same time, as resource consumers. Individual researchers are solely considered as resources consumers. The data warehouses, data marts or other kind of data storage resources, hosted by participant institutions are all considered as data resources. The data-mining or data-analysis services are considered as computational (processing) resources. Resources are variably physically distributed within either one institution's boundary or federally across multiple institutions. Data and computational

resources within each participant institution form the resource fabric in this system.



**Figure 3.1 VO Description**

For example, the users in Institution C may need to retrieve data from Institution A to complete its data-analysis tasks/services. Institution A may need to use the data-analysis service in Institution B to perform another data-analysis task. A sharing relationship can be created across all three institutions, accommodating new participants dynamically; this can occur across different platforms, different DBMSs, and different programming environments. For example, Institution A may use an ORACLE database, which is hosted in a Java-Linux environment, and Institution B may use a SQL Server database in a .NET-Windows environment. Subsequent sections will describe how the sharing is controlled and how interoperability and security is to be handled in the proposed system.

### 3.3. System Functional-Capabilities

The proposed Grid-enabled, distributed data-warehouse system needs to provide a set of capabilities to enable data collection from distributed data sources, and

data-sharing across multiple and dispersed HIV/AIDS research institutions in pre-defined VO. This section will discuss necessary functional capabilities of the proposed system based on (Foster, Gannon, Kishimoto & Von Reich, 2004<sup>2</sup>) (Von Reich et al., 2004) (Foster et al, 2006) (Atkinson et al., 2003) (Atkinson et al., 2004) (Pearson, 2002). These capabilities include data-collection and operation, resource-management, job-execution management, security, activity-monitoring, metadata-management, data provenance, and data resource publishing and discovery.

### **3.3.1. Virtualization**

The fundamental value proposition of a Grid system is virtualization, or transparent access to distributed computer resources. Grids need to provide the ability to virtualize remote resources and capabilities, and make them appear as contiguous entities to end-user applications. For a data-intensive application to derive value from a Grid, this virtualization needs to include federated data sources as well. For example, a virtual database integrates a set of distributed databases presented in a single view, with a single federated schema, that could be used directly. Generally, data-access and processing in a Grid environment require two sets of transparencies: one that presents a unified view of data sources to applications accessing data, the second is a transparency for data-processing that presents a unified view of computational resources to applications processing data. Transparent access to distributed data sources is, therefore, a fundamental requirement for a Grid-enabled, distributed (HIV/AIDS) data-warehouse system.

#### ***3.3.1.1. Data-access Transparency***

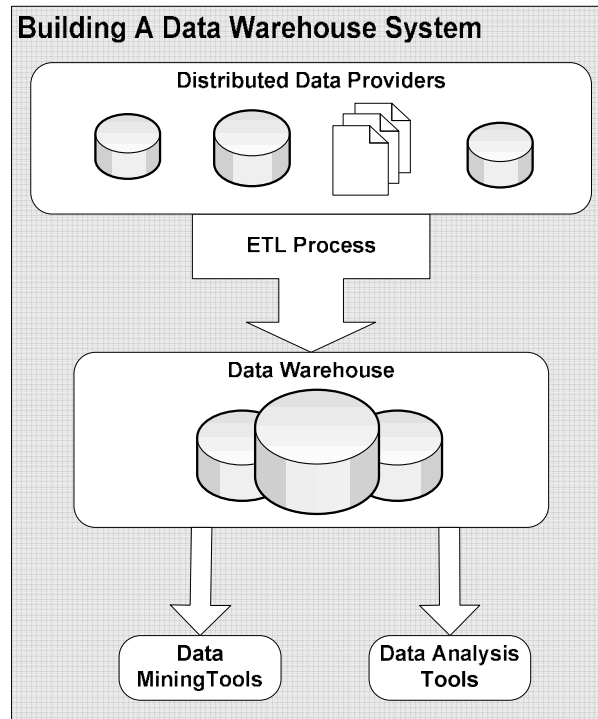
- **Heterogeneity Transparency.** The access mechanism should be independent of the actual implementation of data sources (such as whether it is a flat file system, a DB2 data base, or an Oracle database, etc.). Even more importantly, it should be independent of the structure (schema) of the data source. For example, a data source should be allowed to rearrange its data across different tables without affecting applications.
- **Name Transparency.** An application should be able to access data

without knowing its name or location. Some systems, like DNS (Domain Name System) and distributed file systems, provide a URL (Universal Resource Locator) or name as levels of indirection, but this still requires knowing the exact name of the data object. Instead, data-access should be via logical domains, qualified by predicates on attributes of the desired object. For example, in the proposed system, an HIV/AIDS researcher may want to calculate the total number of HIV-positive patients in a specific age group. “Patients” is the logical domain, spanning multiple HIV/AIDS research institutions. The researcher should not have to specify which institution’s data warehouse or database should be used in the query; rather a separate discovery service should be used by the query processor to map logical domains to data sources. Name transparency includes the traditional notions of location and replication transparency.

- **Ownership and Costing Transparency.** If Grids are successful in the long term, they will almost certainly evolve to span even virtual organizational boundaries, and will involve multiple autonomous data sources. As far as possible, applications should be spared from separately negotiating for access to individual sources, whether in terms of access authorization, or in terms of access costs.

### **3.3.2. Data-collection**

Data-collection, as discussed in Chapter 1, primarily focuses on providing required processes to build data warehouse for the institutions which have not yet built their own aggregated HIV/AIDS patients’ data storage. A data warehouses is constructed through a set of processes called ETL services. Figure 3.2 illustrates a typical approach to building a data-warehouse system. Data-collection essentially initiates the ETL process to populate a data warehouse.



**Figure 3.2 Building a Data Warehouse**

### ***3.3.2.1.Extraction and Transport***

The first part of an ETL process is to extract the data from the distributed sources. Each data source may use different organizations or formats, such as relational database, and flat files. The extracted data needs to be formatted for transformation processes. In the proposed (HIV/AIDS) distributed data-warehouse system, the extraction process comes from data-providers, because the data sources are mostly on-line operational databases within multiple institutional boundaries. In this system, the data-providers are responsible for providing extract components according to the data collectors' requirements. The extracted data can be formatted as binary files which will be submitted to data collectors. It is necessary to define the data and data-file schema which are stored as metadata in a data-providers' metadata repository. Data schema identifies which data is required by subscribers. A data file schema is a logical structure of an extracted data file. It identifies the data field name, format, etc., required by the subscribers. Both data and data file schema must be predetermined for both the data provider and the data collector. This metadata ensures that all data-providers supply data in a uniform format, usable by subscribers (users submitting queries). When an extract process

starts, it needs to reference the local metadata repository first to transform the data at this level. The extracted binary data files, combined with a information file in XML format, are either submitted by data-providers or retrieved by data-subscribers periodically by, for example, using FTP (File Transfer Protocol). The added information file contains descriptive information such as create-time, data-source, destination-name, etc.

### ***3.3.2.2. Transformation and Loading***

The transformation phase applies a series of rules or functions to the extracted data to derive the data to be loaded. These transformation processes depend on the requirements of the data warehouse system. The transformation types include:

- Selecting only certain columns to load;
- Translating coded values (e.g., if the source system stores M for male and F for female, but the warehouse stores 1 for male and 2 for female);
- Encoding free-form values (e.g., mapping "Male" and "M" and "Mr" onto 1);
- Deriving a new calculated value (e.g.,  $\text{sale\_amount} = \text{qty} * \text{unit\_price}$ );
- Joining together data from multiple sources (e.g., lookup, merge, etc);
- Summarizing multiple rows of data (e.g., total patients for each region);
- Generating surrogate key values;
- Transposing or pivoting (turning multiple columns into multiple rows or vice versa).

The loading phase loads the data into the data warehouse. Depending on the requirements of the owners of data warehouse, this process varies widely. For example, some data warehouses merely overwrite old information with new data. However, more complex systems can maintain a history and audit trail of all changes to the data which is contained in a metadata (information) file submitted with the extracted data file.

### **3.3.3. Data operations**

The ability to retrieve data is a basic requirement in the Grid-enabled, distributed data warehouse system. Users must be able to retrieve selected data directly into Grid applications.

#### ***3.3.3.1. Data-Access and Integration***

Data-access and integration provide the capabilities to enable cross-institutional data-sharing. Data-access generally involves the retrieval, insertion or modification of data, which may be available on a variety of platforms and in various formats. As pointed out previously, VOs typically contain diverse data resources with different storage systems, data types, data models and access mechanisms. Data-access primarily focuses on querying data from distributed and diverse data warehouse systems in order to provide uniform support for data analysis and data-mining. Data warehouses contain aggregated historical data that cannot be updated frequently. However, the ability to modify and insert still needs to be provided. The distributed data warehouses either reside in one institution or across multiple institutions. Clients from different institutions need to query data remotely from multiple data resources and retrieve result data sets from the local site for processing. These clients typically establish a session and then submit a series of query statements within some transaction regime. Each submission gets a response: either a result data set or a status report indicating whether the execution succeeded or failed. When linking data resources, the system must provide the ability to use data in one resource as the matching criteria or conditions for retrieving data from another resource, e.g., to, perform a sub-query. The system must be also able to construct distributed queries when the target data resources are located at different sites, and must be able to support heterogeneous and federated queries when some data resources are accessed through different query languages.

Data integration is the ability to combine data residing at different resources and to provide the user with a unified view of this data (Lenzerini, 2002). It ensures that the retrieved data from heterogeneous resources can be merged and

aggregated in order to return a single, logical set of results which can be interpreted by local data-mining or analysis tools. Data-access and integration should of course be independent of some aspects of the database (as discussed under “Virtualization”).

#### ***3.3.3.2. Data Analysis and Interpretation***

A principle aim of Grid systems is to make information more accessible across trust boundaries. The proposed Grid-enabled, distributed data warehouse system intends to provide a much greater opportunity for users to analyze and interpret data they have not created or do not own. The purpose of analysis is to identify features, properties, and behaviours in data, and to identify correlations and anomalies between data. The purpose of interpretation is to explain what has been identified and to derive inferences and conclusions. Both activities, in turn, lead to the creation of further data, information, and knowledge. The generalized tools for providing multi-dimensional and multi-variate analysis capabilities are particularly important for mining historical data to identify trends, correlations, and anomalies in a data warehouse. The multiple analysis techniques may need to apply to the same or related data sets concurrently. Ability is required to record inferences and conclusions drawn by assimilating evidence from each analysis and interpretation step in the process, and to capture the analysis workflow. It should also allow the workflow to be replayed in order to reproduce the analysis steps accurately and to demonstrate the provenance of any derived data. When new data content is created, or existing data content is modified during analysis and interpretation, it must be able to capture and save all the changes. When network performance is poor, data-access paths are slow, or data resources are at remote sites, data availability may be limited, and users may need to carry out analysis on locally maintained copies of data resources.

#### **3.3.4. Data resource publishing and discovery**

The ability to publish all types of data must be supported, regardless of volume, internal structure and format. It must also allow users to describe and

characterize published data in user-defined formats and terms. In addition, the physical characteristics of the data, e.g., volume, number of logical records, and preferred access paths, are necessary in order to access and transport the data efficiently. The minimum information that a user must know in order to reference a data resource is its name and location. Much of the functionality required for defining and maintaining publications are required for defining and maintaining metadata. The ability to register and deregister data resources dynamically is also needed. The burdens of manual metadata entry and editing need to be minimized.

Data resource discovery allows users to discover desired data resources from resources registry with metadata attributes by using interactive browsing tools. The discovery search criteria use user-defined terms and rules, and use defined naming conventions and ontology.

### **3.3.5. Provenance**

Provenance is a record of the origin and history of a piece of data. It is a special form of audit trail that traces each step in sourcing, moving, and processing data, together with “who did what and when”. It is an essential requirement for establishing the ownership, quality, reliability and currency of data when making use of other institutions’ data, particularly during the data resource discovery process. Provenance also provides information that is necessary for recreating data. Conversely, it can avoid time-consuming and data-intensive processing expended in recreating data. For example, the derived data is often originated from multiple sources, multi-staged processes and multiple analysis and interpretation, which could cause the content of a record of provenance to be complex. The capability to record data provenance and the ability for a user to access the provenance record need to be provided in order to establish the quality and reliability of data.

### **3.3.6. Resource management**

The resources in the Grid-enabled, distributed data warehouse system are mainly data resources such as a data warehouse and data marts, but also

computational resources, such as data-mining and data-analysis services, deployed within each institution. All kinds of resources need a standardized representation model for publishing and discovery. From the Grid perspective, resource management is categorized into three types at different levels: the management of resources themselves, resources management in the Grid environment and the management of Grid infrastructure to provide basic functionalities. The system must provide resource management at all these levels. Resource management in a Grid system is quite complex. This section only discusses the data resources management in relation to the proposed (HIV/AIDS) Grid system.

In this system, data warehouses are primary data resources located at distributed sites. They should be managed directly through their native manageability interfaces. Management at this level involves monitoring (i.e., obtaining the state of the resource, which includes events), setup and control (i.e., setting the state of the resource), and discovery. These resources are managed by following the description given by a resource model, which defines their properties, operations, events, and their relationships with each other.

Large numbers of data resources of every type and size could be made available in a Grid environment in the future. The system must provide the capability to manage these resources across multiple, heterogeneous environments globally. Data management facilities must ensure that data resource catalogues, or registries are always available and that the definitions they contain are current, accurate, and consistent. This applies equally to the content of data resources that are logically grouped into virtual databases.

### **3.3.7. Job execution management**

Job execution management is concerned with the problems of instigating and managing to completion, units of work. For example, a data-analysis task is likely to span multiple data warehouses located in distributed sites. A high-level analysis task may be broken down into subtasks prior to execution. Job execution management enables applications to have coordinated access to

underlying resources, regardless of their physical locations or access mechanisms. It is the key to making resources easily accessible to end-users.

In brief, the execution of a task may need these steps: execution planning and scheduling, task initialisation, preparation and execution. The Grid-enabled, distributed data warehouse system requires little planning because it performs repetitive tasks according to well-defined workflows, such as data-gathering for populating a data warehouse and data-analysis of historical data. These repetitive processes generally operate in a relatively stable context with pre-chosen, mostly read-only, data and established workflows. The ability to schedule, execute, prioritize and allocate resources to jobs based on such information is required, as is realizing mechanisms for scheduling across administrative domains, using multiple schedulers. Once the execution is started, the required applications and data resources should be able to be deployed and configured automatically. Jobs must be managed and monitored during their entire lifetime. For example, whether the job execution is successful or not, it must be logged into a log repository or a log database. Data-access ability can be used to access and update log storage.

### **3.3.8. Metadata management**

There are several types of metadata which are important for Grid data operations.

- **Technical metadata** defines the location of data sources and resources; the physical data structure, organization and grouping of data items into logical records; and those characteristics of the data that are important in deciding how data is best accessed and transported. Technical metadata also defines data currency and history; i.e., versions and ownership.
- **Contextual metadata** defines naming conventions, terminologies and ontology through which data can be logically referenced. Contextual metadata increases the quality and reliability of data because the definitions conform to an agreed-upon syntax and semantics, and also record structural associations and relationships within the data between definitions, and to define rules for conflicts between mappings.

- **Derived metadata** defines the context and meaning of data derived from any other data. This type of metadata is commonly used in data warehousing environments, where it is often more efficient to store derived data than to recalculate the values dynamically each time they are required.
- **Mapping metadata** defines equivalences between discrete contextual metadata definitions, and between contextual and technical metadata. The ability to map relationships between contextual metadata is particularly important because of the lack of agreed standards in scientific naming conventions for terminologies and ontologies. It enables users to compare classifications and ontology in terms of their naming conventions, and structural relationships and rules. It also enables them to establish what alternative definitions are available for referencing data content.

The ability to map contextual metadata to physical data structures and schemas is a requirement in order to enable users to access data content using logical references; i.e., without needing to know the definition of its underlying record structure or data schema. Mapping, in conjunction with contextual metadata, enables users to integrate data sets defined in different classifications and ontologies. This provides the ability to specify a single set of search criteria and data-matching rules when performing integrated or federated queries within multiple data resources, and for referencing data in a virtual database. The ability must be provided to update metadata when new data resources are created or updated.

### 3.3.9. Monitoring

System monitoring depends on the information about applications, resources and services in the Grid environment. The term ‘information’ refers to dynamic data or events used for status monitoring, relatively static data used for discovery; and any data that is logged. Information retrieval could make use of data-access and distributed query processing.

The administrative tools are a set of applications that provide system administrators with the ability to maintain monitor and control a system. The

interfaces to the services and resources available should be intuitive and easy to use, as well as being heterogeneous in nature. Typically, user and administrative access to Grid applications and services are Webbased interfaces.

### **3.3.10. Security**

The consequence of unauthorized access to data resources can be catastrophic, and thus, data owners must be able to control sharing of data within the VO. Such sharing requires user authentication, and access-control mechanisms for enforcing local and community policies for data-access and resources usage. For example, when performing a data transfer, users must be authenticated and authorized to access the data being transferred. Different institutions may have different security infrastructures, so the ability to integrate and operate within existing security architectures and models must be provided. Resources may have to be accessed across organizational boundaries. It is necessary to provide standard and secure mechanisms that can be deployed to protect institutions, while also enabling cross-domain interaction without compromising local security mechanisms. Furthermore, the data-access may have to operate within an environment where a variety of legal and ethical policies affect its operation. For example, some policies may restrict the entities that can access personal data (such as personal ID No, name, etc.) and limit the operations that they can perform (confidentiality). Privacy concerns may limit the queries that can be made about individuals, although, in some cases, the policies may permit queries that return results about a group as a whole, such as average income or total salary. The security mechanism needs to allow these restrictions to be specified. When used with data services, these mechanisms must allow the specification of policies that apply at the level of groups (e.g., tables) or elements within a resource. Other requirements such as delegation and single sign-on mechanism are all primary security challenges that need to be handled. Security issues in proposed system are a main concern of this dissertation and will be discussed in the security chapter.

### **3.4. Summary**

Section 3.2 primarily discussed the envisaged VO environment and requirements of the Grid-enabled, distributed data warehouse system. This system intends to use the Grid approach to build the foundation for realizing data-collection and data-sharing required by HIV/AIDS collaborative research. The proposed system introduces a number of challenges, like resource virtualization, interoperability, security and so on. This chapter identified a set of functional capabilities which are essential to coordinate data-sharing, access data, handle interoperability, security, etc. in a Grid-enabled data warehouse context.

## Chapter 4.

### A Proposed System Framework

This chapter proposes a Grid-enabled, distributed data warehouse framework within the context of building the infrastructure to support collaborative HIV/AIDS research. Data warehouses, located at distributed institutions, are the main data resources for data-mining and data analyses. The set of functional capabilities identified in the previous chapter are implicitly presupposed in this framework. (These capabilities include data-collection, data-access, data integration, resource management, data resources publishing, data resources discovery, system monitoring, metadata management, data provenance, security, etc.).

The Open Grid Services Architecture (OGSA), a GGF specification, represents an evolution towards a Grid system architecture based on Web service concepts and technologies. OGSA accommodates stateful services to compensate for plain Web services, which are usually stateless. It incorporates the Web Services Resource Framework (WSRF), which is emerging as a promising standard for modelling stateful resources using Web services. The latest version of Globus Toolkit (GT), developed by the Globus Alliance (<http://www.globus.org/>), is a realization of the OGSA requirements based on WSRF. OGSA and GT are *de facto* standards for building a service-oriented Grid system. The Open Grid Services Architecture Data-access and Integration (OGSA-DAI, <http://www.ogsadai.org.uk/>) (Karasavvas et al., 2005) is a middleware technology that can be used easily to access and integrate data from a variety of data sources, such as relational databases, XML databases and file systems. OGSA-DAI itself is both specification and (open-source) implementation.

The proposed Grid-enabled, distributed data warehouse system is essentially an OGSA-based Grid system that provides essential capabilities to facilitate HIV/AIDS collaborative research. OGSA, WSRF, GT and OGSA-DAI provide the infrastructure services, standard interfaces and protocol bindings to

coordinate resource-sharing, especially data resources, for developing the proposed system.

This framework proposes the standards, specifications, and implementations that can be used to develop such a system. Finally, two models, using the defined services, are proposed in order to realize data-collection, data-access and integration, which are key capabilities to HIV/AIDS collaboration.

The OGSA infrastructure and federal data warehousing have been pre-emptively suggested as the most suitable foundation for this proposed collaboration context. The following sections will attempt to first justify this contention, by examining the available distributed computing paradigms, before elaborating the proposed framework.

## **4.1. Evolution of Distributed Computing**

Service-Oriented Architecture (SOA) is an architectural style, the goal of which is looser coupling among interacting software systems, compared to traditional distributed computing approaches. The technology of Web services is the most likely connection technology of SOA. SOA and Web Services are the basis of service-oriented Grid architecture. It is necessary to provide a brief overview of SOA and related technologies.

### **4.1.1. Traditional distributing computing**

The traditional client/server applications use a variety of communication techniques, such as sockets, Remote Procedure Calls (RPC) (Srinivasan, 1995) (Birrell & Nelson, 1984), JAVA Remote Method Invocation (RMI, <http://java.sun.com/j2se/1.4.2/docs/guide/rmi/spec/rmiTOC.html>), Distributed Component Object Model (DCOM) (Horstmann & Kirtland, 1997), and Common Object Request Broker Architecture (CORBA, <http://www.omg.org/corba/>).

#### **4.1.1.1. *Socket Programming***

Sockets provide low-level APIs for writing distributed client/server applications. Before a client starts communicating with a server, a socket endpoint needs to be created. The transport protocol for communication can be either TCP or UDP in the TCP/IP protocol stack. The client also needs to specify the hostname and port number that the server process is listening on. The standard socket API is well-defined, but the implementation is language-dependent, which means the socket APIs vary with each language. Typically, the socket client and server can be implemented in the same language and use the same socket package, but can run on different operating systems. Socket programming has the advantage of a low latency and high-bandwidth mechanism for transferring large amounts of data compared with other techniques. However, the application development may be an onerous and time-consuming task due to the complexity of interaction between multiple components.

#### **4.1.1.2. *RPC***

RPC is another mechanism that is used to build distributed client/server applications and can use either TCP or UDP as transport protocols. RPC relies heavily on an Interface Definition Language (IDL) interface to describe the remote procedures executing on the server-side. From an RPC IDL interface, an RPC compiler can automatically generate a client-side stub and server-side skeleton. Client-side stub and server-side skeleton help RPC hide the low-level communication abstraction and provide high-level communication abstraction for a client to directly call a remote procedure. A client must specify the hostname or the IP address of the server

RPC itself is a specification and implementations, such as Open Networking Computing (ONC) RPC from Sun Microsystems and Distributed Computing Environment (DCE) RPC, from the Open Software Foundation (OSF), can be used directly for implementing RPC-based client/server applications. RPC is not restricted to any specific language, but most implementations are in C programming language. The same language and the same RPC package must

be used on both client and server sides. Compared with socket programming, RPC is easier to use for developing distributed applications because it provides high-level communication abstraction. However, RPC only supports synchronous communication (call/wait) between client and server and is not object-oriented.

#### **4.1.1.3. *Java RMI***

The Java RMI is an object-oriented mechanism from Sun Microsystems for building distributed client/server applications. It is essentially an RPC implementation in Java. Similar to RPC, Java RMI also hides the low-level communications between client and server by using a client-side stub and server-side skeleton. Java RMI itself is both a specification and implementation, and it is restricted to the Java language in that an RMI client and server have to be implemented in Java, but they can run on different operating systems in distributed locations. Java RMI applies an object-oriented approach. Unlike RPC, a client can pass an object as a parameter to a remote object. RMI has good support for marshalling, which is a process of passing parameters from client to a remote object. The main disadvantages of Java RMI are its limitation to the Java language, its proprietary invocation protocol, JRMP, and that it only supports synchronous communications.

#### **4.1.1.4. *DCOM***

The Component Object Model (COM) is a binary standard for building Microsoft-based component applications, which is language-independent. DCOM is an extension of COM for distributed client/server applications. Similar to RPC and Java, DCOM also hides the low-level communications between client and server by using a client-side stub (called a proxy in DCOM) and server-side skeleton (called a stub in DCOM) using Microsoft's Interface Definition Language (IDL). DCOM is language-independent, which means clients and DCOM components can be implemented in different programming languages. Although it is available on non-Microsoft platforms, it has only achieved broad popularity on Windows. The same as RPC and Java RMI, it only supports synchronous communications.

#### **4.1.1.5. CORBA**

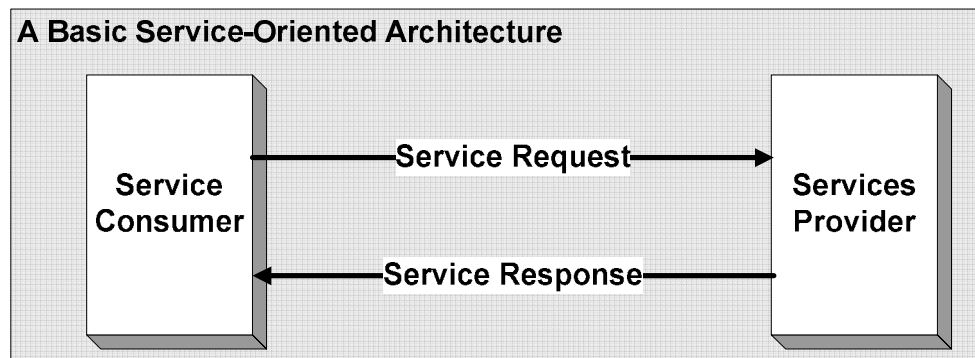
CORBA is an object-oriented middleware infrastructure from the Object Management Group (OMG, <http://www.omg.org>) for building distributed client/server applications. Similar to RPC, Java and DCOM, CORBA also hides the low-level communications between client and server by automatically generating a client-side stub (called a proxy in DCOM) and server-side skeleton (called a stub in DCOM) using IDL interface. Compared to Java RMI and DCOM, CORBA is independent of location, a particular platform or programming language. CORBA supports both synchronous and asynchronous communications. However, CORBA itself is only an OMG specification. There are many CORBA products available that can be used to build CORBA applications.

In summary, Java RMI, DCOM, and CORBA represent the most popular distributed, object-oriented middleware, which can be used to develop distributed client/server applications rapidly. Although they all share some similar features, they do differ in their specific implementations and features (Gopalan, 1998). In summary, middleware, such as Java RMI, DCOM, and CORBA, are not based on open standards, which make it difficult for them to be ubiquitously taken up in heterogeneous environments. Web Services has emerged as an open standards-based middleware infrastructure for building and integrating applications in heterogeneous environments.

#### **4.1.2. Service-Oriented Architecture**

SOA is not new (Box, 2003). It was first proposed by Roy W. Schulte and Yefim V. Natis, who are Gartner analysts. They specified SOA as “a style of multi-tier computing that helps organizations share logic and data among multiple applications and usage modes” (Schulte & Natis, 1996). There are multiple definitions of SOA, but currently, only the OASIS group has created a formal definition with depth, which can be applied to both the technology and business domains. OASIS defines SOA as a paradigm for organizing and utilizing distributed capabilities that may be under the control of different

ownership domains (OASIS, 2006<sup>1</sup>). In computing, the term 'SOA' represents a perspective of software architecture that defines the use of services to support the requirements of software users. In an SOA environment, resources on a network are made available as independent services that can be accessed without knowledge of their underlying platform implementation (Channabasavaiah, Holley & Tuggle, 2003).



*Figure 4.1 SOA*

SOA is essentially an architectural style (He, 2003) (Burbeck, 2000). It is a collection of services, and these services communicate with each other. The communication can involve either simple data passing, or it could involve two or more services coordinating some activity. Some means of connecting services to each other is needed. SOA can also be regarded as a style of information-system architecture that enables the creation of applications that are built by combining loosely coupled and interoperable services. These services inter-operate, based on a formal definition (or contract, e.g., WSDL), which is independent of the underlying platform and programming language. The interface definition hides the implementation of the language-specific service. SOA-compliant systems can therefore be independent of development technologies and platforms (such as Java, .NET etc). For example, services written in C#, running on .Net platform, and services written in Java, running on J2EE platform, can both be consumed by a common composite application. In addition, applications running on either platform can consume services running on the other as Web services, which facilitates reuse. Figure 4.1 illustrates a basic Service-Oriented Architecture. It shows a service consumer on the right sending a service request message to a service provider on the left. The

service provider returns a response message to the service consumer. The request and subsequent response connections are defined in some way that is understandable to both the service consumer and service provider. A service provider can also be a service consumer.

### **4.1.3. Web Services**

Web services technology has emerged as a promising infrastructure for building distributed computing. Web services are based on a SOA in which clients are service requestors and servers are service provider. These differ from traditional approaches, such as Java RMI, COBRA and DCOM, in their focus on simple open standards, such as XML and HTTP (HyperText Transfer Protocol), which have wide industry support and a chance to become truly ubiquitous.

Web services standards are being defined within the W3C and other standards' bodies, and form the basis for major new industry initiatives, such as Microsoft (.NET), IBM, and Sun (Sun ONE). Web service is defined as a software system, designed to support interoperable machine-to-machine interaction over a network. It essentially uses XML (W3C, 2006<sup>1</sup>) (W3C, 2001) to create a robust connection. It has an interface, described as a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization, in conjunction with other Web-related standards (W3C, 2004<sup>3</sup>).

Web services describe a computing paradigm, based on standard techniques for describing interfaces to software components, methods for accessing these components via interoperable protocols, and discovery methods that enable the identification of relevant service providers. These techniques are programming language, programming model, and system software-neutral. They provide a standard means of interoperating between different software applications, running on a variety of platforms and/or frameworks.

A Web service is a loosely coupled, encapsulated, platform and programming language independent, composable server-side component that can be described, published, discovered and invoked over an internal network or on the Internet (Li & Baker, 2005). This description summarizes the main features of Web services' implementation. A Web service can be implemented in any programming language and deployed on any platform. It has a server-side component that uses an XML-based interface to describe its functionalities and capabilities, and it registers with a service registry. Its implementation is free to change without impacting on the service client, as long as the service interface remains the same. A Web service client can discover a service by searching a service registry via Intranet or the Internet. Web services can be bound to by a service client by using standard transport protocols, such as HTTP or FTP.

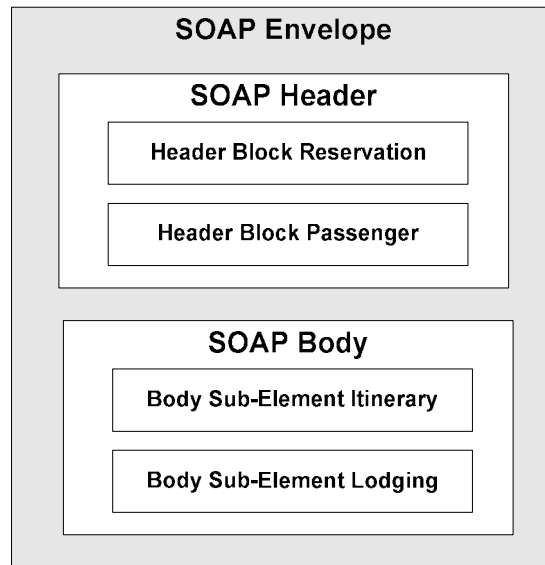
The core standards of Web Services, as defined by W3C, are SOAP (W3C, 2003), Web Services Description Language (WSDL) (W3C, 2006<sup>3</sup>), and Universal Description, Discovery and Integration (UDDI, [www.uddi.org](http://www.uddi.org)). Web Services Inspection (WS-Inspection) (Ballinger, Brittenham, Malhotra, Nagy & Pharies, 2001) is another standard for service discovery.

#### **4.1.3.1. SOAP**

Web services use only messages to communicate between services. SOAP stands for "Simple Object Access Protocol". It is a simple and lightweight communication protocol for clients and servers to exchange messages in a XML format, over a transport-level protocol, which is normally HTTP.

SOAP is a simple enveloping mechanism for XML payloads that defines a RPC convention and a messaging convention. Figure 4.2 shows the structure of a SOAP message. A SOAP header element is optional. A SOAP header is an extension mechanism that provides a way to pass information (such as authentications, transactions and payments) in SOAP messages that is not application payload. The SOAP body is the mandatory element within the SOAP, which implies that this is where the main end-to-end information conveyed in a SOAP message must be carried. The body is the main payload

of the message. When an RPC call is used in a SOAP message, the body has a single element that contains the method name, arguments and URI (Uniform Resource Identifier) of the service target address.

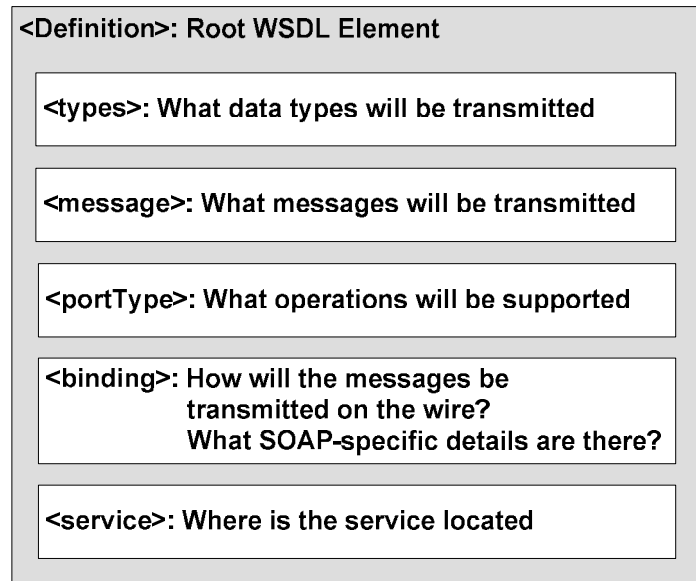


*Figure 4.2 SOAP Envelope*

#### **4.1.3.2. WSDL**

WSDL is an XML-based specification that is used for describing a Web service, e.g., the functionalities, capabilities and address of a service, and how to invoke it. It defines services as a set of endpoints, operating on messages containing either document-oriented (messaging) or RPC payloads. Using WSDL, a client can locate a Web service and invoke any of its publicly available functions. Service interfaces are defined, abstractly, in terms of message structures and sequences of simple message exchanges or operations and then bound to a concrete network protocol and data-encoding format to define an endpoint.

Figure 4.3 illustrates the common elements of a WSDL document. `<portType>` is the key element of a WSDL document. It defines a set of abstract operations provided by a service. Each operation uses messages defined in the `<message>` element to describe its inputs and outputs.



*Figure 4.3 WSDL Document*

#### **4.1.3.3. UDDI and WS-Inspection**

UDDI is an industry standard for service registration and discovery. A service provider uses UDDI to advertise its services, and a client uses UDDI to find the appropriate services for its purpose. Data in UDDI can be organized in white pages, yellow pages and green pages, representing different types of information. A client uses SOAP to access a UDDI registry. A UDDI registry exposes a set of APIs in the form of SOAP-based Web services. The API contains Inquiry and Publishing APIs for services discovery and publication.

WS-Inspection comprises a simple XML language and related conventions for locating service descriptions published by a service provider. A WS-Inspection Language (WSIL) document can contain a collection of service descriptions and links to other sources of service descriptions (Brittenham, 2002). A service description is usually a URL to a WSDL document; occasionally, a service description can be a reference to an entry within a UDDI registry. A link is usually a URL to another WS-Inspection document; and occasionally, a link is a reference to a UDDI entry. With WS-Inspection, a service provider creates a WSIL document and makes the document network accessible. Service requestors use standard Web-based access mechanisms (e.g.,

HTTP GET) to retrieve this document and discover what services the service provider advertises.

UDDI and WS-Inspection address different sets of issues with service registration and discovery (Nagy & Ballinger, 2001). UDDI provides a high degree of functionality, but it causes the costs of increased complexity. The WS-Inspection provides less functionality in order to maintain a low overhead. The two specifications can be used together or separately, depending on the situation.

## **4.2. Open Grid Services Architecture**

The Open Grid Services Architecture (OGSA) Framework, the Globus-IBM vision for the convergence of Web services and Grid computing, was presented at the GGF meeting, held in Toronto in February 2002. It was initially described in the “physiology” paper. The Open Grid Services Architecture, version 1.0 (Foster et al., 2005) produced by the OGSA working group within the GGF in 2005, provides a first version of this OGSA definition. In July 2006, the Open Grid Services Architecture, version 1.5 (Foster et al., 2006) was published by GGF.

In addition to defining a core set of standard interfaces and behaviours that address many of the technical challenges in a VO environment, OGSA provides a framework within which one can define a wide range of interoperable, portable services. OGSA provides a foundation on which can be constructed a rich Grid-technology ecosystem, comprising multiple technology providers. This section gives an introduction of OGSA.

### **4.2.1. Advantages of Web Services for Grid computing**

Grid computing concerns multi-institutional resources sharing and coordinating uses of diverse resources in a dynamic, distributed VO that assembles resources and services (and people). Since Web service has emerged as an XML-based open standard for building distributed applications in a heterogeneous

computing environment, the Grid can benefit from the Web services framework by taking advantages of several factors.

As described in previous section, Web services are independent of platforms, programming languages and locations. Web services can be described, published and dynamically discovered, and bound to WSDL, a rich interface description language. The Grid needs this support as the dynamic discovery and composition of Grid services in heterogeneous environments necessitates mechanisms for registering and discovering interface definitions and endpoint implementation descriptions, and for dynamically generating proxies based on (potentially multiple) bindings for specific interfaces. WSDL supports this requirement by providing a standard mechanism for defining interface definitions separately from their embodiment within a particular binding (transport protocol and data encoding format). The widespread adoption of Web-service mechanisms means that a framework based on Web services can exploit numerous tools and extant services

#### **4.2.2. Service-Oriented view**

OGSA introduces a service-oriented Grid architecture. OGSA focuses on services. A service is defined as a network-enabled entity that provides some capability to its clients by exchanging messages. A service is defined by identifying sequences of specific message exchanges that cause the service to perform some operations. Defining these operations only in terms of message exchange achieves great flexibility in how services are implemented, and where they may be located. A SOA is one in which all entities are services, and thus, any operation visible to the architecture is the result of message exchange. In OGSA, computational resources, storage resources, networks, programs, databases, and the like are all represented as services.

A critical requirement in a distributed, multi-organizational Grid environment is for mechanisms that enable interoperability. In a service-oriented view, the interoperability problem is divided into two sub-problems: the definition of service interfaces and the identification of the protocol(s) that can be used to invoke a particular interface—and, ideally, agreement on a standard set of such

protocols. A service-oriented view addresses the need for standard interface definition mechanisms, local/remote transparency, adaptation to local OS services, and uniform service semantics. A service-oriented view also simplifies virtualization—that is, the encapsulation behind a common interface of diverse implementations. Virtualization is easier if service functions can be expressed in a standard form, so that any implementation of a service is invoked in the same manner. WSDL is adopted for this purpose. WSDL supports a service interface definition that is distinct from the protocol bindings used for service invocation. WSDL allows for multiple bindings for a single interface, including distributed communication protocol(s) (e.g., HTTP), as well as locally optimized binding(s) (e.g., local IPC) for interactions between request and service processes on the same host. In brief, this service architecture supports local and remote transparency with respect to service location and invocation.

### **4.2.3. The Grid Service**

A basic premise of OGSA is that everything is represented by a service. Computational resources, storage resources, networks, programs, databases, and so forth are all services. More specifically, OGSA represents everything as a Grid service.

The ability to virtualize and compose services depends on more than standard interface definitions. OGSA defines what is called a Grid service: a Web service that provides a set of well-defined interfaces and that follows specific conventions. The interfaces address discovery, dynamic service creation, lifetime management, notification, and manageability; the conventions address naming and upgradeability. Authentication and reliable invocation are also viewed as service protocol bindings and are thus, external to the core Grid-service definition.

Grid services are characterized by the capabilities that they offer. A Grid service implements one or more interfaces, where each interface defines a set of operations that are invoked by exchanging a defined sequence of messages. Grid service interfaces correspond to `<portType>` in WSDL. The set of `<portType>`, supported by a Grid service, along with some additional

information relating to versioning, are specified in the Grid service's <serviceType>, a WSDL extensibility element defined by OGSA.

Grid services are stateful and dynamic. Grid services can maintain an internal state for the lifetime of the service. Grid service instance refers to a particular instantiation of a Grid service. Grid services can be created and destroyed dynamically. The existence of state distinguishes one instance of a service from another that provides the same interface. One instance can be created from another dynamically. The protocol binding associated with a service interface can define a delivery semantics that addresses, for example, reliability. Services interact with one another by the exchange of messages. Grid services may be upgraded during their lifetime.

#### **4.2.4. OGSA core services**

OGSA version 1.0 and OGSA version 1.5 are the specifications that focus on requirements and the scope of important capabilities required to support Grid systems and applications in both e-science and e-business. OGSA addresses the need for standardization by defining a set of core capabilities and behaviours that address key concerns in Grid systems.

OGSA is intended to facilitate the seamless use and management of distributed, heterogeneous resources. The virtualization and abstraction are directed toward defining a wide variety of capabilities that are relevant to OGSA Grids. The identified functional and non-functional requirements include: interoperability, resource-sharing across organizations, quality of service (QoS) assurance, job execution, data services, security, scalability, etc. OGSA realizes these capabilities in terms of services, the interfaces these services expose, the individual and collective state of resources belonging to these services, and the interaction between these services within a service-oriented architecture. The services are built on Web-service standards, with semantics, additions, extensions and modifications that are relevant to Grids.

The potential range of OGSA services is vast. This definition of OGSA 1.5 is driven by a set of functional and non-functional requirements derived from a

broad set of use cases (Foster et al., 2004<sup>2</sup>) (Von Reich et al., 2004). OGSA defines a set of capabilities to meet these requirements. It presents a refinement of the required functionality into capabilities: Execution Management, Data, Resource Management, Security, Self-Management, and Information services. There is a core set of non-null interfaces, standards and common knowledge/bootstraps that services must implement to be part of an OGSA Grid. This set of common implementations and manifestations to support OGSA is referred to as the infrastructure services. OGSA capabilities share and build on this set of infrastructure services. The following is a brief introduction to OGSA's capabilities:

- Execution Management Services (OGSA-EMS) are concerned with the problems of instantiating and managing, to completion, units of work. There are three broad classes of EMS: resources that model processing, storage, executables, resource management, and provisioning; job management; and resource selection services that collectively decide where to execute a unit of work.
- Those OGSA services are concerned with the management of, access to and update of data resources, along with the transfer of data between resources, are collectively called "Data Services". OGSA Data Services can be used to move data as required; manage replicated copies; run queries and updates; and federate data resources. They also provide the capabilities necessary to manage the metadata that describes this data, in particular, the provenance of the data itself. The heterogeneous nature of the Grid means that many different types of data must be supported. These include, but are not limited to, flat files, streams, relational databases, XML databases, object-oriented databases, catalogues, derived data and even data services themselves. OGSA Data Services provide a set of functional capabilities and properties. Different subsets of the services are needed to implement the different capabilities. Functional capabilities include: data transfer services, storage management services, data-access services, queries, data federation services, location management services, update services, data transformation services, security mapping extensions, resource and services configuration services, data discovery services and provenance services. Properties are

non-functional capabilities of services. Whereas functional capabilities are defined by entries in the service interfaces, non-functional properties are inherent in the design. These properties include performance, availability, legal and ethical restriction and scalability.

- OGSA Resource Management Services perform several forms of management on resources in a Grid. In an OGSA Grid, there are three types of management that involve resources: management of the physical and logical resources themselves, management of the OGSA Grid resources exposed through service interfaces and management of the OGSA Grid infrastructure. Different types of interfaces realize the different forms of management in an OGSA Grid. These interfaces can be categorized into three levels: resource level, infrastructure level and OGSA functions' level. At the resource level, resources are managed directly through their native manageability interfaces. The infrastructure level provides the base management behaviour of resources, forming the basis for both manageability and management in an OGSA environment. Management functionality at the infrastructure level is envisioned to use the Web Services Distributed Management (WSDM) specifications. At the OGSA functions' level, there are two types of management interfaces: a functional interface and a manageability interface. Functional interfaces are provided by some common OGSA capabilities (e.g., OGSA EMS). These capabilities themselves are a form of resource management. Each capability has a specific manageability interface through which the capability is managed (e.g., monitoring of a job manager). The properties provided by OGSA resource management services include scalability, interoperability, security and reliability.
- OGSA Security Services facilitate the enforcement of the security-related policy within a VO. OGSA Security Services provide a set of functional capabilities. These functional capabilities include: authentication, identity mapping, authorization, credential conversion, audit and secure logging, and privacy.
- OGSA Self-Management Services are used to help reduce the cost and complexity of owning and operating an IT infrastructure. In a self-managing

environment, system components—including hardware components, such as computers, networks and storage devices, and software components, such as operating systems and business applications—are self-configuring, self-healing and self-optimizing. One of the main objectives of self-management is to support service-level attainment for a set of services (or resources, depending on the taxonomy)—with as much automation as possible, to reduce the costs and complexity of managing the system. While the self-management capability is a significant part of the OGSA, this work is still at a preliminary stage and hence, only some aspects of self-management are described in this specification (Foster et al., 2006).

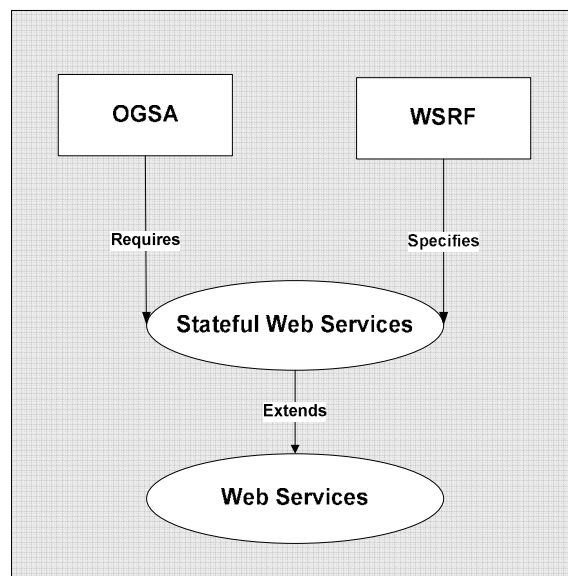
- The ability to efficiently access and manipulate information about applications, resources and services in the Grid environment is an important OGSA capability. The term ‘information’, in defining OGSA Information Services, refers to dynamic data or events used for status monitoring; relatively static data used for discovery; and any data that is logged. The scope of the OGSA Information Service covers publication through consumption. An information service needs to support a variety of QoS requirements for reliability, security, and performance. OGSA Information Services define capabilities, including discovery, message delivery, logging, and monitoring.

## **4.3. WSRF and WS-Notification**

### **4.3.1. OGSA requires stateful services**

OGSA introduced a service-oriented Grid architecture which tailors the Web-services approach to meet some Grid-specific requirements (Foster et al., 2002). Grid service is defined as stateful and dynamic as discussed in section 4.2.3. Web services were chosen as the underlying technology to support OGSA. However, although the Web Services Architecture was certainly the best option, it still did not meet one of OGSA's most important requirements: that the underlying middleware has to be stateful. Unfortunately, although Web services, in theory, are either stateless or stateful, they are usually stateless, and there is no standard way of making them stateful.

A stateless service implements message exchanges with no access or use of information not contained in the input message. Plain Web services are usually stateless. This means that the Web service cannot "remember" information, or keep state, from one invocation to another. Although OGSA requires stateful services, statelessness is generally viewed as good engineering practice for Web-services implementations. Statelessness in the implementation of the service itself tends to enhance reliability and scalability: a stateless Web service can be restarted following failure, without concern for its history of prior interactions, and new copies of a stateless Web service can be created (and subsequently destroyed) in response to changing loads. WS-Resource is an approach to solve this contradiction. The WS-Resource Framework (WSRF) is a set of Web services specifications that define WS-Resource approach. Figure 4.4 illustrates the relationship between Web services, stateful Web services, OGSA and WSRF.



*Figure 4.4 Relationships between OGSA, WSRF, and Web Services*

### 4.3.2. WS-Resource

WS-Resource is an approach to modelling states in a Web-services context (Foster et al., 2004). A WS-Resource is defined as the composition of a Web service and a stateful resource. A stateful resource is defined as having a specific set of state data expressible as an XML document; having a

well-defined lifecycle; and be known to, and acted upon, by one or more Web services. Examples of system components that may be modelled as stateful resources are files in a file system, rows in a relational database, and encapsulated objects, such as Entity Enterprise Java beans. A stateful resource can also be a collection or group of other stateful resources.

A stateful resource is addressed and accessed according to the implied resource pattern, a conventional use of WS-Addressing (W3C, 2006<sup>2</sup>) endpoint reference (EPR). The term ‘implied resource pattern’ describes a specific kind of relationship between a Web service and one or more stateful resources. WS-Addressing standardizes the EPR construct, used to represent the address of a Web service deployed at a given network endpoint. A WS-Addressing EPR is an XML serialization of a network-wide pointer to a Web service. In the implied resource pattern, a stateful resource identifier is used to identify a WS-Resource. It is encapsulated in an EPR and used to identify the stateful resource to be used in the execution of a Web-service message exchange.

The lifetime of a WS-Resource is defined as the period between its creation and its destruction. A WS-Resource can be created through the use of a WS-resource factory. A WS-Resource factory is any Web service capable of bringing a WS-Resource into existence. Bringing a WS-Resource into existence consists of creating a new stateful resource, assigning the new stateful resource an identity, and creating the association between the new stateful resource and its associated Web service. The response message of a WS-Resource factory operation contains a WS-Resource-qualified EPR (an EPR containing a stateful resource identifier is a WS-Resource-qualified EPR), containing a stateful resource identifier that refers to the new stateful resource, though a factory may convey the reference to the new WS-Resource through other means, such as placing the WS-Resource-qualified EPR into a registry for later retrieval. A new WS-Resource can exist for some finite period. After that time, it should be possible to destroy the WS-Resource so that its associated system resources can be reclaimed. A WS-Resource can be destroyed by using the appropriate WS-Resource-qualified EPR to send a destroy request message to the Web service identified by the EPR. The receipt of the response to the

destroy request message represents a point of synchronism between the service requestor and the Web service receiving the destroy request message.

The term ‘resource property’ refers to an individual component of a WS-Resource’s state. The XML document, describing the type of a stateful resource within the WS-Resource composition, is called a WS-Resource properties document. The WS-Resource properties document is described using XML schema (W3C, 2001). Specifically, the WS-Resource properties document is expressed as an XML global element declaration (GED) in some XML namespace. The WS-Resource properties document declaration is associated with the WSDL <portType> definition via the use of a standard attribute, resourceProperties. The state of a WS-Resource, i.e., the values of resource properties exposed in the WS-Resource’s resource properties document, can be read, modified, and queried by using standard Web services messages. The base functionality is to retrieve the value of a single resource property using a simple Web services request/response message exchange. It is also possible to use a standard message exchange to execute an arbitrary XPath (W3C, 2006<sup>5</sup>) expression against the resource properties document.

### **4.3.3. WSRF and WS-Notification family of specifications**

The WSRF is a set of Web services specifications that define a rendering of the WS-Resource approach in terms of specific message exchanges and related XML definitions. The WSRF allows WS-Resources to be declared, created, accessed, monitored for change, and destroyed via conventional Web-service mechanisms, but does not require that the Web-service component of the WS-Resource that provides access to the associated stateful resources be implemented as a stateful message processor. Three related WS-Notification specifications define interfaces and behaviours that allow clients to subscribe to change in state, thus providing for push-mode access to state components.

Initial work on the WS-Resource Framework has been performed by the Globus Alliance and IBM (Czajkowski et al., 2004<sup>2</sup>). These documents were

submitted to the OASIS standards group in March 2004. The OASIS WSRF Technical Committee ([http://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=wsrf](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf)) was formed to work on WS-ResourceProperties (OASIS, 2006<sup>8</sup>), WS-ResourceLifetime (OASIS, 2006<sup>7</sup>), WS-ServiceGroup (OASIS, 2006<sup>6</sup>), and WS-BaseFaults (OASIS, 2006<sup>2</sup>) specifications. The four WSRF specifications define how to represent, access, manage, and group WS-Resources:

- **WS-ResourceProperties** defines how WS-Resources are described by XML Resources Property documents that can be queried and modified.
- **WS-ResourceLifetime** allows a requestor to destroy a WS-Resource either immediately or at a scheduled future point in time.
- **WS-ServiceGroup** describes how collections of Web services and/or WS-Resources can be represented and managed. It creates and uses heterogeneous by-reference collections of Web services. This specification can be used to organize collections of WS-Resources, for example to build registries, or to build services that can perform collective operations on a collection of WS Resources.
- **WS-BaseFault** defines a base fault type for use when returning faults in a Web services message exchange.

The WSRF specifications are compliant with the WS-Interoperability (WS-I) Basic Profile (Ballinger et al., 2006). It means that any WS-I compliant Web services client can interact with any service that supports WSRF specifications.

In a dynamic Grid environment, it is critical that components can request and receive timely notification of changes in one another's states. WS-Notification is a family of related white papers and specifications that define a standard Web services approach to notification, using a topic-based publish/subscribe pattern. WS-Notification defines a general, topic-based Web service system for publish and subscribe (pub/sub) interactions that build on the WSRF. The Event-driven, or Notification-based, interaction pattern is a commonly used pattern for inter-object communications. The term 'notification pattern' refers to the interaction pattern that involves registration of consumers and subsequent

dissemination of events. This notification pattern is increasingly being used in a Web services context. In the notification pattern, a Web service, or other entity, disseminates information to a set of other Web services, without having to have prior knowledge of these other Web services. The goals of WS-Notification are to standardize the roles, terminology, concepts, message exchanges and the WSDL needed to express the notification pattern and to provide a language to describe Topics. The OASIS WSN Technical Committee ([http://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=wsn](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn)) was formed to work on WS-BaseNotification (OASIS, 2006<sup>4</sup>), WS-Topics (OASIS, 2006<sup>12</sup>), and WS-BrokeredNotification (OASIS, 2006<sup>5</sup>) specifications.

- **WS-BaseNotification.** This defines the Web services interfaces for Notification Producers and Notification Consumers. It includes standard message exchanges to be implemented by service providers that wish to act in these roles, along with operational requirements expected of them. This is the base document on which the other WS-Notification specification documents depend.
- **WS-Topics.** This defines a mechanism to organize and categorize items of interest for subscription known as ‘topics’. These are used in conjunction with the notification mechanisms defined in WS-BaseNotification specification. WS-Topics defines three topic expression dialects that can be used as subscription expressions in subscribe request messages and other parts of the WS-Notification system. It further specifies an XML model for describing metadata associated with topics. The WS-Topics specification should be read in conjunction with the WS-BaseNotification specification and the “Publish-Subscribe Notification for Web Services” white paper (Graham et al., 2004).
- **WS-BrokeredNotification.** This defines the Web services interface for the Notification Broker. A Notification Broker is an intermediary which, among other things, allows publication of messages from entities that are not themselves service providers. It includes standard message exchanges to be implemented by Notification Broker service providers, along with operational requirements expected of service providers and requestors that participate in brokered notifications. This work relies upon

WS-BaseNotification specification and WS-Topics specification, as well as the “Publish-Subscribe Notification for Web Services” white paper.

#### **4.3.4. WSRF and OGSi**

The predecessor of WSRF, the Open Grid Services Infrastructure (OGSI) specification version 1.0 (Tuecke et al., 2003), was released in July 2003, by the OGSi Working Group of the GGF. Since development started on OGSi in early 2002, the Web services world has evolved significantly. Specifically, a number of new specifications and use patterns have emerged that simplify and clarify the ideas expressed in OGSi. It is necessary to mention how the new WSRF and WS-Notification specifications derive from and relate to the OGSi specification.

OGSi defines a set of conventions and extensions on the use of WSDL and XML Schema to enable stateful Web services. It introduces the idea of a stateful Web services and defines approaches for creating, naming, and managing the lifetime of instances of services; for declaring and inspecting service state data; for asynchronous notification of service state change; for representing and managing collections of service instances; and for common handling of service invocation faults.

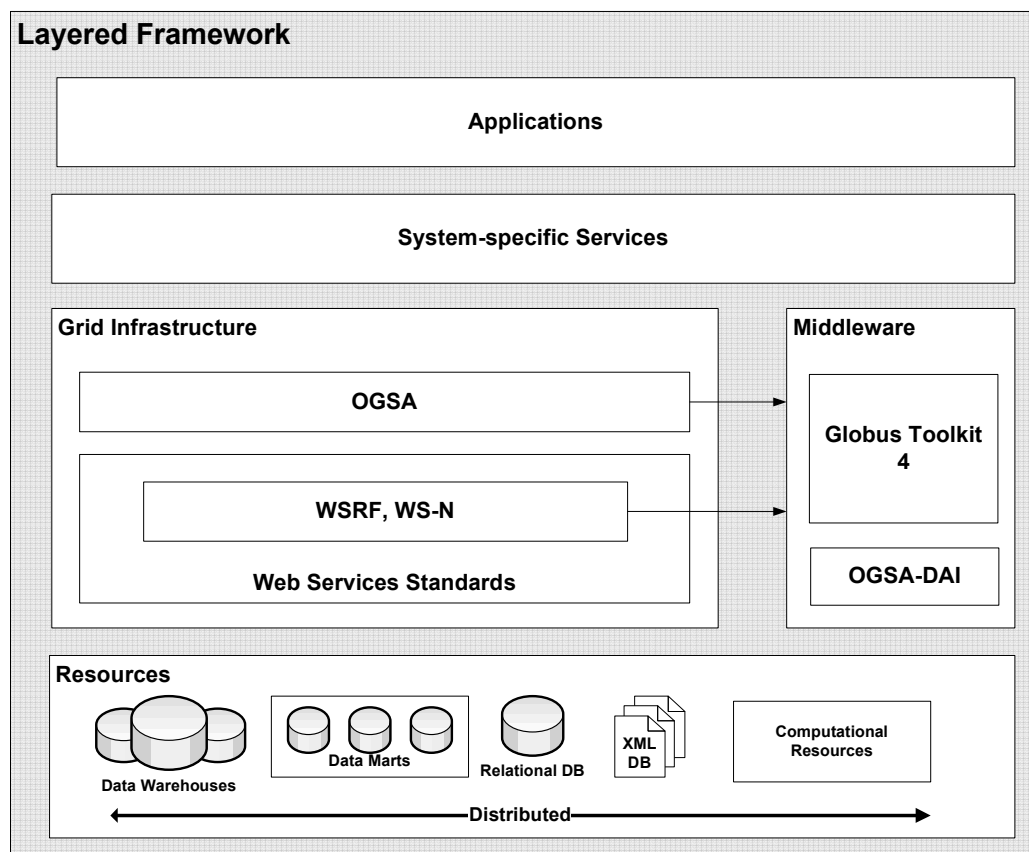
The WSRF was proposed as a refactoring and evolution of OGSi aimed at exploiting new Web services standards, specifically WS-Addressing, and at evolving OGSi based on early implementation and application experiences. The WSRF retains essentially all of the functional capabilities present in OGSi, while changing some of the syntax (e.g., to exploit WS-Addressing) and also adopting a different terminology in its presentation. In addition, OGSi is considered a heavyweight specification with too much definition in one specification; the WSRF partitions OGSi functionality into a set of distinct, composable specifications (plus the related WS-Notification specifications).

The “Refactoring and Evolution” document (Czajkowski et al., 2004<sup>1</sup>) explains the relationship between OGSi and the WSRF and the related WS-Notification family of specifications; explains the common requirements that both address,

and compares and contrasts the approaches taken to the realization of those requirements, and describes how OGSI constructs map to WS-Resource Framework constructs.

## 4.4. System Framework

Web services family of standards, OGSA, WSRF are essential to build an OGSA-based Grid system. Web services were chosen as the underlying technology to meet the requirements of OGSA. WSRF introduces an approach to model stateful resources in a Web services framework. WS-Notification specifications use standard approaches to notification using a topic-based publish and subscribe pattern. WS-Notification defines notification mechanism for the components in a Grid system to request and receive timely notification of changes in one another's states.



*Figure 4.5 System Framework*

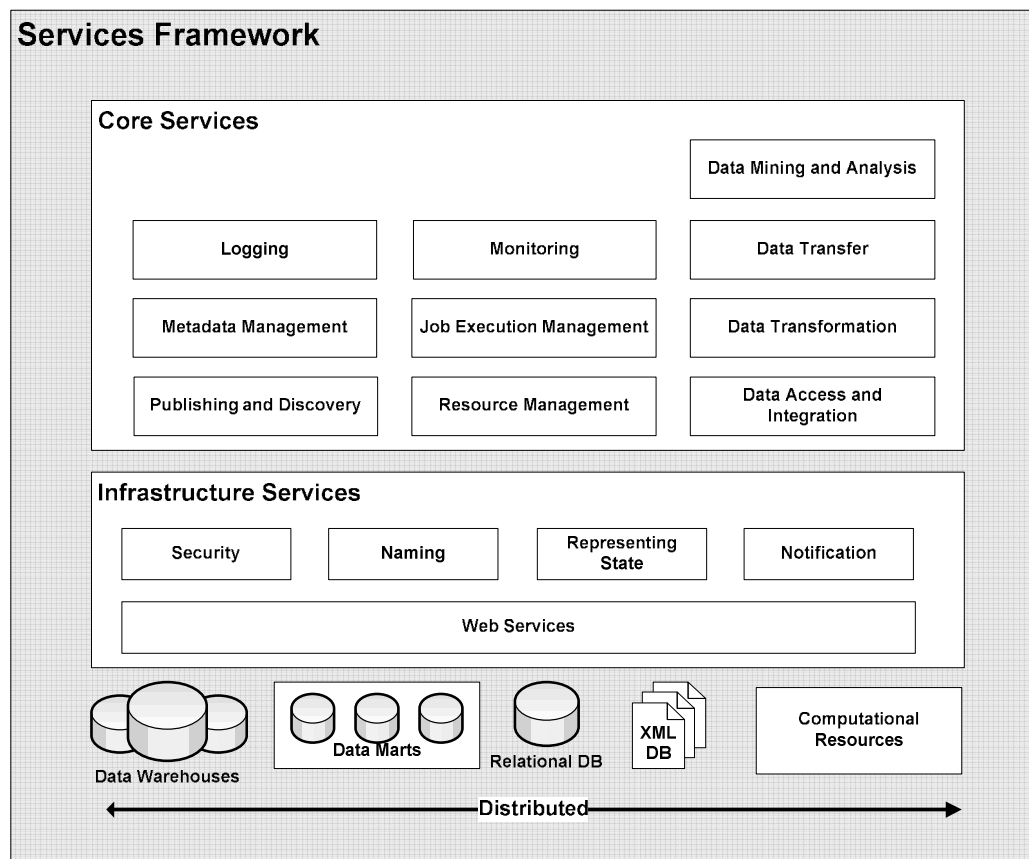
The Grid-enabled, distributed data warehouse system is designed as an OGSA-based Grid system. It requires a set of services to provide various capabilities based on OGSA, and its supporting standards and specifications. OGSA introduces a service-oriented Grid architecture, based on Web services. It represents everything as Grid services, which are stateful Web services with standard interfaces and protocol bindings. In addition, OGSA defines a set of services that provide key capabilities in a Grid system. Based on OGSA, the capabilities required by the system are represented as Grid services. Figure 4.5 illustrates the layered framework for the system. In this framework, OGSA is the core standard to develop the proposed system. In addition, it identifies all supportive standards, specifications, and implementations tools. This framework is divided into four main tiers: resource, Grid infrastructure, system-specific services and applications.

- **Resources.** This tier couples a variety of resources that are available in the pre-defined VO. It primarily consists of data resources and computational resources. Data warehouses are the main data resources that are fundamental to data-mining and data analyses. A data warehouse is aggregated from a set of databases. These databases could be relational databases, XML databases, object-oriented databases or even flat files. Furthermore, additional single databases are possibly required for some specific purposes. For example, it is necessary to build information databases (of extraneous metadata) for logging events, monitoring data provenance, etc. A metadata repository may be represented as an XML database for storing resource locations and data-mapping information. Computational resources are primarily data-analysis and data-mining components deployed within each institution.
- **Grid infrastructure.** This tier provides infrastructure for building high-level capabilities (e.g., data analysis) required by the system. It consists of two parts: the standards and specifications provide essential capabilities with standardized interfaces and protocol binding; the open source middleware for realising these standards and specifications. Web services comprise the underlying technology to the OGSA-based Grid system. The resources in the resource tier are represented as

WS-Resources. WS-Notification provides the approach for notification of changes in one another WS-Resource's states. OGSA offers a set of basic capabilities, represented as Grid services, in order to fulfil the general requirements of an OGSA-based Grid system. Globus Toolkit version 4 (GT4) and OGSA-DAI are open source middlewares to implement these standards and specifications for developing the proposed system. GT4 is a realization of OGSA, including a complete implementation of WSRF specifications. The toolkit addresses issues of security, information discovery, resource management, data management, communication, and portability. GT mechanisms are in use at hundreds of sites and by dozens of major Grid projects worldwide (Foster, 2002<sup>1</sup>). OGSA-DAI (Antonioletti et al., 2005) allows data resources, such as relational or XML databases, to be accessed via Web services. The WSRF version of OGSA-DAI is compatible with the GT's implementation of WSRF. Open Grid Services Architecture Distributed Query Processor (OGSA-DQP) (Alpdemir et al., 2003) is an extension of OGSA-DAI that provides a service-based distributed query processor which is an implementation of service-based distributed query processing based on OGSA-DAI data services. It supports the evaluation of queries over collections of potentially remote relational data services. In practice, it is used by the system to support queries over OGSA-DAI Data Services and over other services available on the system, thereby combining data-access with data-analysis operations.

- **System-specific services.** This tier primarily defines high-level services that are built on infrastructure-level services. This tier would mainly contain data-analysis and data-mining services that would be used to facilitate HIV/AIDS-specific research. The services defined in this tier should also be compliant with OGSA specifications (capabilities).
- **Applications.** This tier comprises the user applications that operate within the pre-defined VO. Through such applications, a set of data-mining and data-analysis tools invoke the services within the second and third tier to complete some analysis tasks and present meaningful results to users.

These four layers represent the services groups within different levels of the Grid-enabled, distributed data warehouse system. The infrastructure tier is a critical part of the proposed system. It defines the standardized services based on OGSA. These services, therefore, virtualize resources and handle interoperability and security issues in the proposed system. Based on this tier, more high-level services can be built to provide more domain-specific capabilities for the system, according to the system requirements. Figure 4.6 shows the service framework, which contains all required services in the Grid infrastructure tier. These services are divided into two levels: infrastructure services and core services.



*Figure 4.6 Service Framework*

#### 4.4.1. Infrastructure services

Some common components, including naming, representing state, notification and security, exist at a higher management level and are identified as infrastructure services. As a brief recap: OGSA offers capabilities by defining

services; the services interfaces are defined by WSDL; XML acts as the lingua franca for description and representation, and SOAP acts as the primary message exchange format for all defined services. The infrastructure services are OGSA services and can be described as follows:

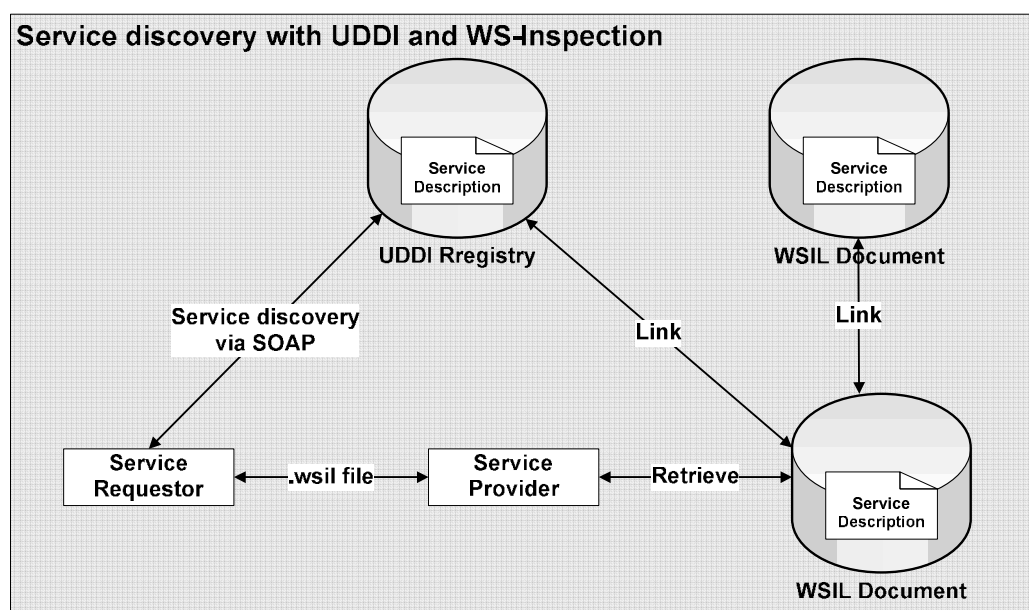
- **Naming service.** According to the OGSA naming service, OGSA-naming uses a three-level convention. The naming service in this system complies with OGSA-naming conventions. Every name-able entity (e.g., a service or resource) in the system is associated with an (optional) human-oriented name, an abstract name, and an address. The human-oriented name is usually human-readable and may belong to a name space. Name spaces are usually hierarchic and usually have syntactic restrictions. Human-oriented names do not have to be unique. The abstract name is a persistent name that does not specify a particular location. The abstract names are globally unique in space and time. The AbstractName element of a WS-Name (Grimshaw et al., 2006) is an example of abstract name. The address specifies the location of an entity. WS-Addressing EPR is an example of an address. The services or resources addressing in the system use WS-Addressing EPRs.
- **Security services.** Security is one key area in a Grid environment. A service request needs to carry appropriate tokens securely for purposes of authentication, authorization and end-to-end message protection. In addition, higher-level protection mechanisms, such as XML encryption and digital signatures and point-to-point transport-level security, are required too. The security services will be discussed in the security chapter.
- **Representing state.** WSRF specification addresses the key issues in the area of state representation and manipulation: the ability to model, access and manage state, and related, activities.
- **Notification.** WS-Notification specification provides the notification mechanisms that support subscription to, and subsequent notification of changes to state components.

### 4.4.2. Core services

The core services provide Fabric-level capabilities to meet the requirements of the Grid-enabled, distributed data warehouse system. These services are compliant with OGSA Grid services. They are built upon the infrastructure services, which use the standard interfaces and mechanisms to describe operations, exchange messages, name and represent resources and services, and handle security. They can be described as follows:

#### 4.4.2.1. *Publishing and Discovery Services*

An OGSA Grid system is structured according to SOA principles. Since everything (e.g., databases, computational resources, programs, etc.) in an OGSA-based system is represented as services, a service provider needs to publish a description of a service to a service registry, which can be consulted by a service requestor. The services registry has persistent storage for the latest information and is optimized for searches. It contains metadata about the resources. The services registry may be replicated and distributed in multiple locations for scalability, and it need to be updated frequently and simultaneously to adapt to the dynamic Grid environment. Additionally, a discovery service virtualizes the name and location of services on the system.



**Figure 4.7 Service Discovery**

A registry can use standard description models, such as UDDI and WS-Inspection. UDDI is an industry standard for service registration and discovery. A service provider uses UDDI to advertise the services, and a service requestor uses UDDI to find the appropriate services by using SOAP message. A UDDI registry exposes a set of APIs containing inquiry and publishing APIs for services discovery and publication. In WS-Inspection, services are described in WS-Inspection documents. A WS-Inspection document provides a means for aggregating references to pre-existing service description documents, which have been authored in arbitrary number of formats such as WSDL, UDDI or plain HTML. UDDI and WS-Inspection can work either together or separately. Figure 4.7 shows the service discovery with UDDI and WS-Inspection. Furthermore, data discovery requires languages or ontologies for describing the data and a query language that operates over these descriptions. The infrastructure for data discovery—registries, notification, etc. use the same mechanisms.

#### **4.4.2.2. *Resource Management Services***

The resource management is a complex task in a Grid environment. The resource management in the proposed system complies with the OGSA resource management mechanism (or architecture) (Maciel, 2005). OGSA resource management concerns three types of management that involve resources:

- Management of the physical and logical resources themselves
- Management of the OGSA Grid resources, exposed through service interfaces (e.g., resource reservation, job submission and monitoring)
- Management of the OGSA Grid infrastructure, exposed through its management interfaces (e.g., monitoring a registry service)

Different types of interfaces realize these forms of management. These interfaces can be categorized into three levels, shown in the middle column of Table 4.1 (Maciel, 2005).

Several definitions are clarified in the OGSA standard:

- A **manager** initiates management actions; it might be either a management console operated by a human, or a software entity that is able to monitor and control its targets automatically.
- **Manageability** defines information that is useful for managing an entity. Manageability encompasses those aspects of an entity that support management, specifically through instrumentation that allows managers to interact with the entity. The manageability may be provided by the entity itself or by a separate means.
- **Manageability interfaces** are sets of standardized interfaces that allow a manager to interact with an entity in order to perform common management actions on it. Typical management actions include starting the entity, stopping it, and gathering performance data.
- **Manageable entities** are entities that provide manageability interfaces and thus, as the name implies, can be managed. Manageable entities can be: physical (e.g., a node, a network switch, or a disk) or logical (e.g., a process, a file system, a print job, or a service), discrete (e.g., a single host) or composite (e.g., a cluster), and transient (e.g., a print job) or persistent (e.g., a host)

Type of management	Level of interface	Interface
Management of the resources themselves	Resource level	WBEM, SNMP, etc
	Infrastructure level	WSRF, WSDM, etc.
Resource management on the Grid	OGSA functions level	Functional interfaces
Management of OGSA infrastructure		Specific manageability interfaces

***Table 4.1 Resource Management***

At the resource level, the resources are managed directly through their native manageability interfaces. These resources are managed by following the description given by an information model, which defines their properties, operations, events, and their relationships with each other. For example, the

Distributed Management Task Force: Common Information Model (CIM, <http://www.dmtf.org/standards/cim/>) (DMTF, 1999) infrastructure is an approach to the management of systems and networks that applies the basic structuring and conceptualization techniques of the object-oriented paradigm. CIM is a model for describing overall management information in a network/enterprise environment. Web-Based Enterprise Management (WBEM, <http://www.dmtf.org/standards/wbem/>) is a set of technologies developed to unify the management of enterprise computing environments. WBEM is based on CIM. It is composed of CIM, which defines the resource model semantics, and a set of encodings and protocols to access the resource model.

The infrastructure level provides the base management behaviour of resources, forming the basis for both manageability and management. At this level, it uses standardized management behaviours to integrate the vast number and type of resources. The infrastructure level provides: a base manageability model, basic functionality, and generic manageability. The base manageability model, which represents resources as services and allows resources in the system to be manipulated through the standard Web services means for discovery, access, etc. All manageable resources are either Web services or are represented by Web services. WSRF defines the interfaces and behaviours which are the basis for representing resources. Furthermore, WS-Notification defines interfaces and behaviours for event notification. The Web Services Distributed Management (WSDM) specifications define: 1) how management of any resource can be accessed via Web services protocols – Management Using Web Services (MUWS), and 2) management of the Web services resources via the MUWS – Management Of Web Services (MOWS). WSDM MUWS (OASIS, 2005<sup>6</sup>) (OASIS, 2005<sup>7</sup>) provides a foundation for management using Web services. WSDM MOWS (OASIS, 2005<sup>5</sup>) builds on the MUWS specification for the management of Web services. The WSRF and WSN specifications, together with WSDM MUWS, will provide the core functionality for the base manageability interfaces. Basic functionality at infrastructure level includes: the interfaces for capabilities that are common to many resources (e.g., start, stop, etc.); representation of the state graph of a resource, including the states

and transitions, and operations to change the state; describing and discovering relationship among resources; and notification. A generic manageability interface is common to all services implementing OGSA capabilities. This manageability interface has functionality such as introspection, monitoring, and creation and destruction of service instances. WSDM MOWS defines standard manageability interfaces for Web services that should be applicable to OGSA services.

At the functions level, there are two types of management interfaces: the functional interface and the manageability interface. In an OGSA-based Grid, some common OGSA capabilities (e.g., job management) are a form of resource management. Services that provide these capabilities expose them through functional interfaces. Each capability has a specific manageability interface through which the capability is managed (e.g., monitoring of registries, monitoring of a job manager, etc.). This interface could extend the generic manageability interface, adding any manageability interfaces that are specific to the management of this capability.

#### **4.4.2.3. *Job Execution Management Services***

Job execution management services are concerned with instantiating and managing, to completion, units of work. The job execution management in the proposed system complies with OGSA Execution Management Services (OGSA-EMS). EMS services enable applications to have coordinated access to underlying resources services, regardless of their physical locations or access mechanisms. More formally, EMS addresses problems with executing units of work, including their placement, provisioning, and lifetime management. It primarily includes: finding execution candidate locations, selecting execution location, preparing for execution, initiating the execution and managing the execution. EMS defines three classes of services: service container, job management and resource selection services.

A service container contains running entities, whether they are jobs or running services. Containers have resource properties that describe static information, such as what kind of executables they can take—OS version, libraries installed,

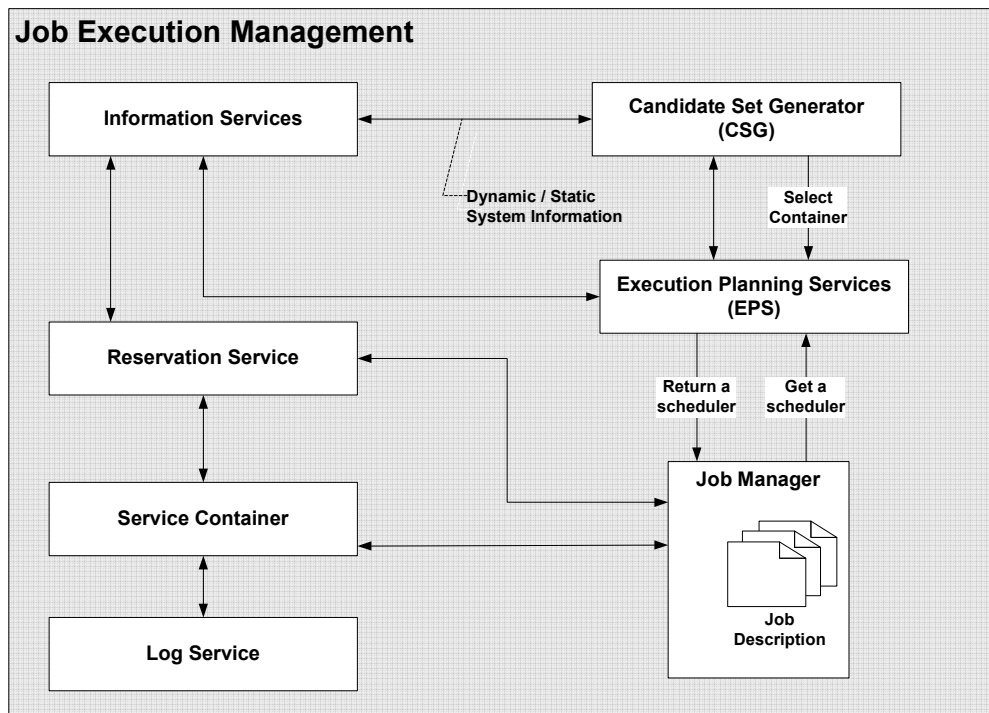
policies, and security environment—as well as dynamic information. A basic container interface may expose only a small set of operations (Grimshaw, Newhouse, Pulsipher & Morgan, 2006). In particular, a container is expected to implement a manageability interface which could be a WSDM managed resource.

The OGSA-EMS definition of a ‘job’ incorporates and extends the notion of a traditional job. The job encapsulates all there is to know about a particular unit of work (i.e., an instance of a running application or a service). A job is the smallest unit that is managed. It represents the manageability aspect of a unit of work. A job implements a manageability interface, which could be a WSDM managed resource. A job is named by an EPR. It is created at the instant that it is requested, even though, at that point, no resources may have been committed. The job keeps track of execution state (e.g., started, suspended, restarted, terminated, and completed), resource commitments and agreements, job requirements, and so on. Many of these are stored in a job document. A job document describes the state of the job, the agreements that have been acquired, its job status, metadata about the user (credentials, etc.), and how many times the job has been started. The job document may be exposed as a resource property of the job.

The Job Manager (JM) is a higher-level service that encapsulates all of the aspects of executing a job, or a set of jobs, from start to finish. A set of jobs may be structured or unstructured. The JM implements a manageability interface which could be a WSDM collection—a collection of manageable entities. The JM is responsible for orchestrating the services used to start a job or set of jobs, by, for example, negotiating agreements, interacting with containers, and configuring monitoring and logging services. It may also aggregate job resource properties from the set of jobs it manages. The JM may be a portal that interacts with users and manages jobs on their behalf.

Resource selection services contain several services: Execution Planning Services (EPS), Candidate Set Generator (CSG), and Reservation services. An EPS is a service that builds “schedules,” where a schedule is a mapping (relation) between services and resources, possibly with time constraints. An EPS will

typically attempt to optimize some objective function, such as execution time, cost, reliability, etc. It first calls a CSG (see below) to get a set of resources, then gets more current information on those resources from an information service, then executes the optimization function to build the schedule. EPS generate a schedule, and a schedule is enacted by JM. CSG determines the set of resources on which a unit of work can execute. CSG generates a set of EPRs of containers in which it is possible to run a job described by a job document. CSGs should be primarily called by EPSs, or by other services, such as JMs, that are performing EPS-like functions. Reservation services manage reservations of resources, revoke reservations, etc. This may not be a separate service, rather an interface to get and manage reservations from containers and other resources. The reservation itself is likely to be an agreement document that is signed. A reservation service presents a common interface to all varieties of reservable resources on the Grid. Reservation services will generally be used by many different services: a JM might create reservations for the groups of jobs which are being managed, or an EPS might use reservations in order to guarantee the execution plan for a particular job. It could also be the case that the creation of reservations will be associated with the provisioning step for a job.



**Figure 4.8 Job Execution Management**

Figure 4.8 is an example of job execution. It illustrates the involved services and the interaction among them. This case use EMS to control the processing of a new job. The JM firstly creates a new job with the appropriate job description. The JM then call an EPS to get a scheduler. The EPS, in turn, calls CSG, which calls information services to determine where the job can be executed, based on binary availability and policy settings. Basically, information services (see section 4.4.2.4) are databases of attribute metadata about resources. The EPS selects a service container, after first checking with the service container that the information is accurate. The EPS returns the schedule to the JM. The JM then may interact with reservation (or deployment and configuration) services to set up the job execution environment. The service container is invoked to start the job. Logging services are used for accounting and audit trails. When the job terminates, the job manager is notified by the container. If the job terminates abnormally, the whole cycle may repeat again.

#### **4.4.2.4. *Information Services***

The ability to efficiently access and manipulate information about applications, resources and services is an important capability in an OGSA Grid. In OGSA specifications, information refers to dynamic data or events used for status monitoring, relatively static data used for discovery and any data that is logged. In practice, an information service needs to support a variety of QoS requirements for reliability, security, and performance. The information services for the proposed system comply with OGSA Information Services and inherit most of the capabilities provided by OGSA Information Services. The approach, defined by Grid Monitoring Architecture, will be used to provide capabilities for information discovery and delivery.

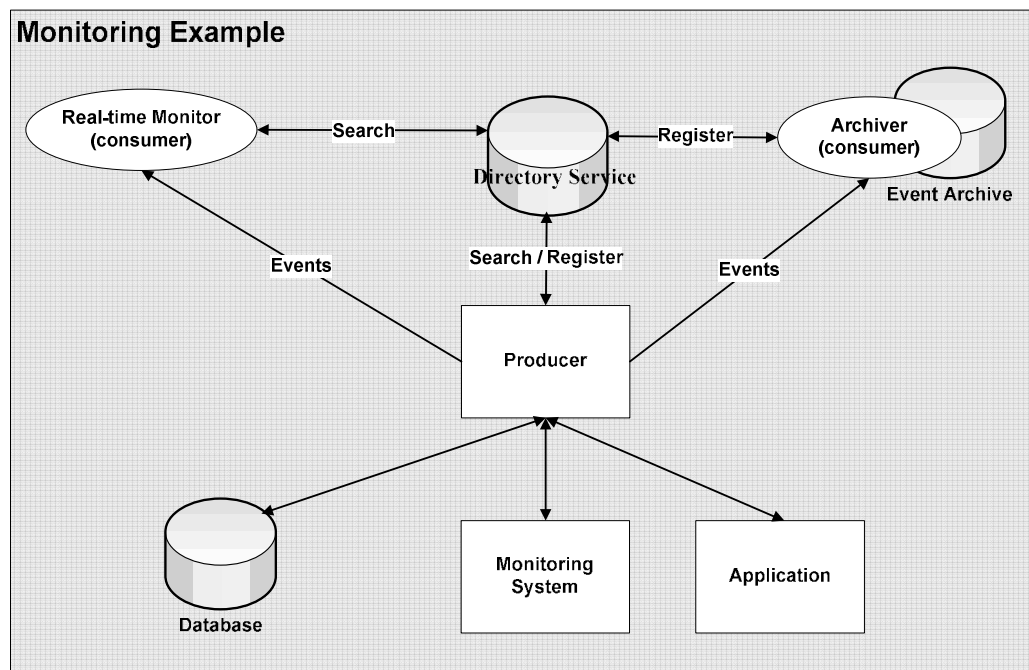
The characterization of an information service depends greatly on factors, such as the demand placed on the source of information (e.g., static versus dynamic, publication rate), its purpose (e.g., discovery, logging, and monitoring) and QoS requirements. Information is made available for consumption, either from the originating producer, or through an intermediary (e.g., logging service or notification broker), acting on behalf of the originating producer. Either one or

more consumers wish to obtain information from one or more producers, or one or more producers wish to send information to one or more consumers. The data model used to implement an information services, or the language used to query for information, are mostly based on XML and XPath/XQuery query languages, and those that use the relational model and the SQL query language. Metadata is associated with information (e.g., events or messages) for describing its structure, properties and usage. For interoperability, a standard event scheme for information services is desirable.

OGSA Information Services define discovery, message delivery, logging, and monitoring capabilities. Discovery service is discussed in section 4.4.2.1. Information producers and consumers interact by exchanging messages. Producers either send messages directly to relevant consumers or make use of an intermediary (message broker) that decouples producers from consumers. Logging service keeps log records in a persistent store for a period of time. It acts as an intermediary between log artifact producers and consumers. Producers write log artifacts sequentially, and consumers may read (but not update) the log records. Information that carries a field for ordering purposes (e.g., a time stamp and sequence number) can be used for monitoring. A monitoring service could be equally used for applications or resources. Some situations (e.g., real-time applications) might impose strict requirement on the monitoring service (e.g., high update rates and high performance). In such a case, a special purpose service might be needed.

For Grid resources in general (including services and applications), the amount of available information about resources could be large, dispersed across the network, and updated frequently. A direct exchange between a producer and consumer is not appropriate or not possible, and searches in this space may have unacceptable latencies. In order to manage such information in a controllable way, it is important to separate information source discovery from information delivery. Grid Monitoring Architecture (GMA) (Tierney et al., 2002) introduces an approach for this separation, based on the Producer-Intermediary-Consumer pattern, which is the basic pattern of decoupling producers from consumers using an intermediary and is widely used.

In the Producer-Intermediary-Consumer pattern, producers put data into an intermediary, and consumers extract data from it. In a general sense, this is the pattern followed by any data store, that is, producers write to and consumers read from a file, RDBMS, ODBMS etc. In addition to supporting producer and consumer interfaces, an intermediary may also support a management interface to control those functions that are not directly associated with reading or writing to the data store.



**Figure 4.9 An Example of Monitoring**

Figure 4.9 shows a sample use of GMA. In GMA, a producer is any component that uses the producer interface to send events to a consumer. A consumer is any component that uses the consumer interface to receive event data from a producer. The GMA defines a directory service to store information about producers and consumers that accept requests. Producers and consumers publish their existence in the directory service. They typically also publish information regarding the types of events they produce or consume, along with the meta-information about accepted protocols, security mechanisms, and so forth. This publication information allows other producers and consumers to discover the types of event data that are currently available, or accept the characteristics of that data, and the ways to gain access to that data.

The directory service is not responsible for the storage of event data itself as it contains only per-publication information about which event instances can be provided or accepted. The event schema may, optionally, be available through the directory service.

A given component may have multiple producer interfaces, each acting independently and sending events. The core interaction functions that may be supported by a producer include: maintaining registration (add/update/remove directory service entry or entries describing events that the producer will send to a consumer), accepting a query request from a consumer, accepting a subscribe request from a consumer, accepting unsubscribe request from the consumer, locating consumer (search the directory service for a consumer), notifying (send a single set of event(s) to a consumer), initiating subscribe (request to consumer to send it events), and initiating unsubscribe (terminate a subscription with a consumer). A given component may have multiple consumer interfaces, each acting independently and receiving events. The core interaction functions that may be supported by a consumer are: locating producer (search the directory service for a producer), initiating query (request one or more events from a producer), initiating subscribe (request establishment of a subscription with a producer), terminating a subscription, maintaining registration (add/update/remove directory service entry or entries describing events that the consumer will accept from the producer), accepting notification (accept a single set of event(s) from a producer), accepting subscribe (accept a subscribe request from a producer), accepting unsubscribe (accept a unsubscribe request from a producer) and locating event schema (search request to the schema repository for a given event type). Many types of consumers are possible, such as a real-time monitor or an archiver. Real-time monitors collect monitoring data in real time used by online analysis tools. An archiver aggregates and stores event data in long-term storage for later retrieval or analysis. Many Grid services may, in fact, be both consumers and producers of monitoring events. These advanced services is the compound producer/consumer, which is a single component that implements both producer and consumer interfaces. Use of these intermediate components can lessen the load on producers of event data that is of interest to many consumers. It is effective to reduce the network

traffic, as the intermediaries can be placed “near” the data consumers. The data used to construct events can be gathered from many sources. Hardware or software sensors that sample performance metrics in real time constitute one type of data source. Another is a database with a query interface, which can provide historical data.

#### **4.4.2.5. *Data Services***

Data services provide the key capabilities that are specific to data-access and operation. OGSA Data Services are concerned with the management of, access to and update of data resources, along with the transfer and replica of data between resources. It provides a broad set of functional and non-functional capabilities that can be defined by entries in the service interfaces. The data services for this system comply with the OGSA Data Services and inherit most capabilities provided by OGSA Data Services. They can be used to move data as required; manage replicated copies; run queries and updates; and federate data resources. The combination of subsets of data services can provide more high level capabilities such as data-collection and data analysis.

A distributed system may contain a variety of data resources. These resources may use different models to structure the data, different physical media to store it, different software systems to manage it, different schema to describe it, and different protocols and interfaces to access it. The data may be stored locally or remotely; may be unique or replicated; may be materialized or derived on demand. Virtualizations are abstract views that hide these distinctions and allow the data resources to be manipulated without regard to them. The data virtualization provides transparency for data-access and processing (see section 3.3.1.1).

A layer of Grid data virtualization services (Raman et al., 2003) provide such transparency and enable ease of data-access and processing. These services support federated access to distributed data, dynamic discovery of data sources by content, dynamic migration of data for workload balancing, parallel data processing, and collaboration. These services virtualize various aspects of the

Grid, and make it appear as a single entity to the end-user applications. Data virtualization services may include data discovery, federation, consistency management service, collaboration, workflow coordination etc. They are described as follows:

- Resources are represented as services in an OGSA Grid. Discovery service (discussed in section 4.4.2.1) virtualizes the name and location of data on the Grid, and forms the basic data virtualization service. Other virtualization services build on top of this name transparency to offer other kinds of transparency.
- A data federation service will analyze each query that it receives to determine how to best answer the query. This may involve the generation of sub-queries against one or more of the data resources making up the federation, applying transformations to the results of those sub-queries, combining those query results in (arbitrarily) complex ways, and then transforming the result into the format requested by the client. It may also determine where intermediate processing is done in order to minimize network traffic. It will allow data sources to be added and removed from the federation, provided they satisfy that service's semantic restrictions. The application specifies its queries in terms of logical domains and predicates; the discovery service maps these onto relevant sources. Thus, the combination of federation and discovery services provide applications with heterogeneity, distribution, and location transparency.
- A data-intensive Grid applications may use Grids for scalability of performance, rather than for integrating data sources. This type of applications typically takes the form of complex workflows of transformation and data-analysis operations, running over large numbers of discrete objects. These operations are often data parallel Grids, which can be used to scale up these applications on demand. A Workflow Coordination Services will automatically spread these operations across Grid nodes, taking responsibility for moving and caching data and functions, recovering from node failures, and so on. Workflow Coordination Services solve workflow parallelizing without changing applications.

- Grid applications often distribute their data across multiple sites. A consistency management service is used to keep the different data pieces consistent with one another. The simplest form of such consistency is referential integrity (Ramakrishnan & Gehrke, 1998), which can be provided by DBMS. Some Grid datasets may also need more sophisticated integrity constraints, which could also vary on an application-specific (user community-specific) basis
- Data-intensive Grid applications involve sharing of data between users at different sites. A collaboration service is used to propagate updates to all users and to resolve conflicts in order to virtualize independent, distributed data updates. The collaboration service must rely on the Grid data sources to maintain version information.

Data services offer data transfer from one location to another. This may be to create a copy of the original data or migrate it completely. QoS can be specified, such as reliable transfer, the maximum bandwidth to use, the time when delivery is required, or delivery guarantees. The base functionality transfers bytes from one source to a single sink. Thin layers above this support transfer to multiple sinks, and allow the preservation of semantic information, such as file hierarchies, byte ordering or encodings. Most services have a security policy decision point before transferring data. An access operation may then check whether the resulting transfer is allowed, depending on the contents of the data, and whether any restrictions apply to that transfer (such as encryption).

Data-access generally concerns the retrieval, insertion or modification of data, which may be available from a variety of infrastructures and in a range of formats. Data-access services include simple access, queries, federation access (discussed above), and update.

- Simple access services operations for reading and writing (logically) consecutive bytes from a data resource. The virtualization interface hides the details of the data location.
- Data-access services provide mechanisms for applying queries against data resources. In simple cases these may run an SQL query over a relational

database, an XML query over an XML database. Other services may implement distributed queries over federated databases or text mining over a set of documents. Synchronous queries return the data in the response to a request, while asynchronous queries expose the derived data as new resources. Services could be data resources. It means they may also deliver the results of a query to a specified set of other services. Query services may optimize a query before sending it to the resource. The resources may further optimize the query and may also handle issues, such as concurrent access to the data. Data services provide a range of mechanisms for updating data resources, depending on the semantics of the data resource, and the nature of the data to be uploaded. Examples include updating a record in a database, or bulk loading data to such resources. Data services may specify some form of transactional behaviour for update operations. When a data resource has replicated versions or is the source for derived data services, the updates may be propagated to the replicated or derived versions. When several clients are updating the same data resource, the various forms of consistency should be maintained by the virtualization interface (consistency management service).

- Various components of the system may transform data. In the most general sense, any service that consumes data of one sort and generates other data could be viewed as a transformation. Data services may themselves transform data from one format to another, or filter it by using built-in functionalities, before moving it or updating it. These transformations may be instigated explicitly by certain operations, or they may be programmed to be triggered automatically in response to certain conditions.

Data services also provide management capabilities to manage data storage, security mapping extension, resource and service configuration, metadata catalogue, and provenance. They are described as follows:

- Storage management services control the provision of storage to applications and other services. They manage quotas, lifetime, and properties such as encryption and persistency.
- Database management systems often implement sophisticated security

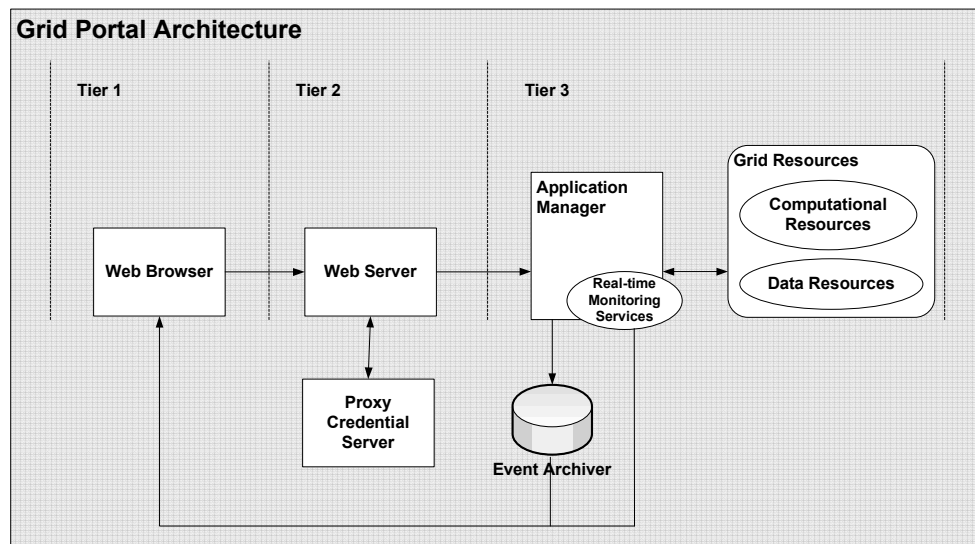
mechanisms. Some of these provide a large range of possible operations and access control at the level of individual records. The data services support the standard OGSA security infrastructure that allows effective use of fine-grained mechanisms in the OGSA Grid.

- Data services can provide functionalities to enable clients to use configuration options provided by data resources. In addition, the services may provide additional operations for configuring the virtualization of the resource provided by the service.
- Metadata catalogues are data services that store descriptions of data held in certain other data services. On the other hand, the metadata for data services may include information about the structure of the data, including references to the schemas that describe the data. For some services this is not practical, as the data resources include many schemas that are modified frequently, and in these cases, schema information will be provided by the services themselves.
- Users of data services may wish to see information about the provenance and quality of the data provided by the services. Provenance is a special form of audit trail that traces each step in sourcing, moving, and processing data. This may be at the level of the whole resource or of its component parts, sometimes to the level of individual elements. This, in turn, requires the services or other processes that generate the data to also maintain the consistency of the provenance information. Complete provenance information can allow the data to be reconstructed by following the workflow that originally created it. Provenance information may be provided by the service itself, or it may be maintained in a metadata catalogue or a logging service.

### **4.4.3. Grid portal**

The Grid end users (e.g. health workers, HIV/AIDS scientists, researchers and medical practitioners), use the Grid to solve domain-specific problems. A Grid portal is a Web-based gateway that provides seamless access to a variety of backend resources. In general, a Grid portal provides end users with a

customised view of software and hardware resources specific to their particular problem domain. It also provides a single point of access to Grid-based resources that they have been authorized to use. For the Grid-enabled, distributed data warehouse system, this will allow HIV/AIDS researchers to focus on their problem area by making the Grid a transparent extension of their desktop computing environment. So far, Grid portal development can be broadly classified into two generations. First-generation Grid portals are tightly coupled with Grid middleware, such as GT. The second generation of Grid portals are those that are starting to emerge and make use of technologies, such as portlets, to provide more customisable solutions. Most Grid portals currently in use belong to the first-generation. This section focuses on the architecture and services of first-generation Grid portals, which will provide a Web-based gateway for the Grid-enabled, distributed data warehouse system.



**Figure 4.10 Grid Portal Architecture**

The first generation of Grid portals primarily used a three-tier architecture (Gannon et al., 2002) as shown in Figure 4.10. It consists of an interface tier of a Web browser, a middle tier of Web servers and a third tier of backend services and resources, such as databases, programs, etc. A user interacts with Web browser through a secure connection. The Web server obtains a proxy credential from a proxy credential server and uses that to authenticate the users. When users complete defining parameters of the task they want to execute, the portal Web server launches an application manager, which is a process that

controls and monitors the execution of Grid tasks. The Web server delegates the user's proxy credential to the application manager so that it can act on the user's behalf. The proxy credential, proxy credential server and proxy credential delegation will be discussed in the next chapter in detail.

The Grid portal generally accommodates the following Grid services:

- **Job management:** A portal provides users with the ability to manage their tasks' execution. For example, a user's application is launched via the Web browser in a reliable and secure way; the statuses of tasks are monitored; and the user can pause or cancel tasks if necessary.
- **Discovery services:** A portal uses a discovery service to find resources and services that are needed and available for a task.
- **Information services:** Dynamic and static data or events can be collected by using real-time monitoring services and events archivers. This data can be retrieve from either real-time monitoring or event archives for monitoring purpose.
- **Authentication:** When users access the Grid via a portal, the portal can authenticate users with their username and password. Once authenticated, a user can request the portal to access Grid resources on the user's behalf.

The GridPort 2.0 (GP2, <http://Gridport.npaci.edu>) and Grid Portal Development Kit (GPDK) (Novotny, 2002) are two representatives of Grid portal toolkit used to facilitate the easy development of application-specific portals. GP2 is a Perl-based Grid portal toolkit, which is a collection of services, scripts and tools, where the services allow developers to connect Web-based interface to backend Grid services. The script and tools provide consistent interfaces between the underlying infrastructure, which is based on Grid technologies, such as GT, and standard Web technologies, such as CGI. GPDK is another Grid portal toolkit that uses Java Server Pages (JSPs) for portal presentation and JavaBeans to access backend Grid resources via GT. Grid portals currently in use include XACT Science Portal (Krishnan et al., 2001), JiPANG (Suzumura, Matsuoka & Nakada, 2001), ASC Grid Portal (Allen et al., 2001) etc.

#### 4.4.4. Models

The previous sections defined the essential services to support the Grid-enabled, distributed data warehouse system based on OGSA. These services provide a rich set of fundamental capabilities. The combinations of subsets of these services are used to provide more high-level capabilities to meet some of the system's requirements (see section 3.3). This section will introduce two models in order to realize data-collection services, and cross-institutional data-access and integration services required by the HIV/AIDS research framework. These two models contain all the components required by these two services, and show the interactions between them.

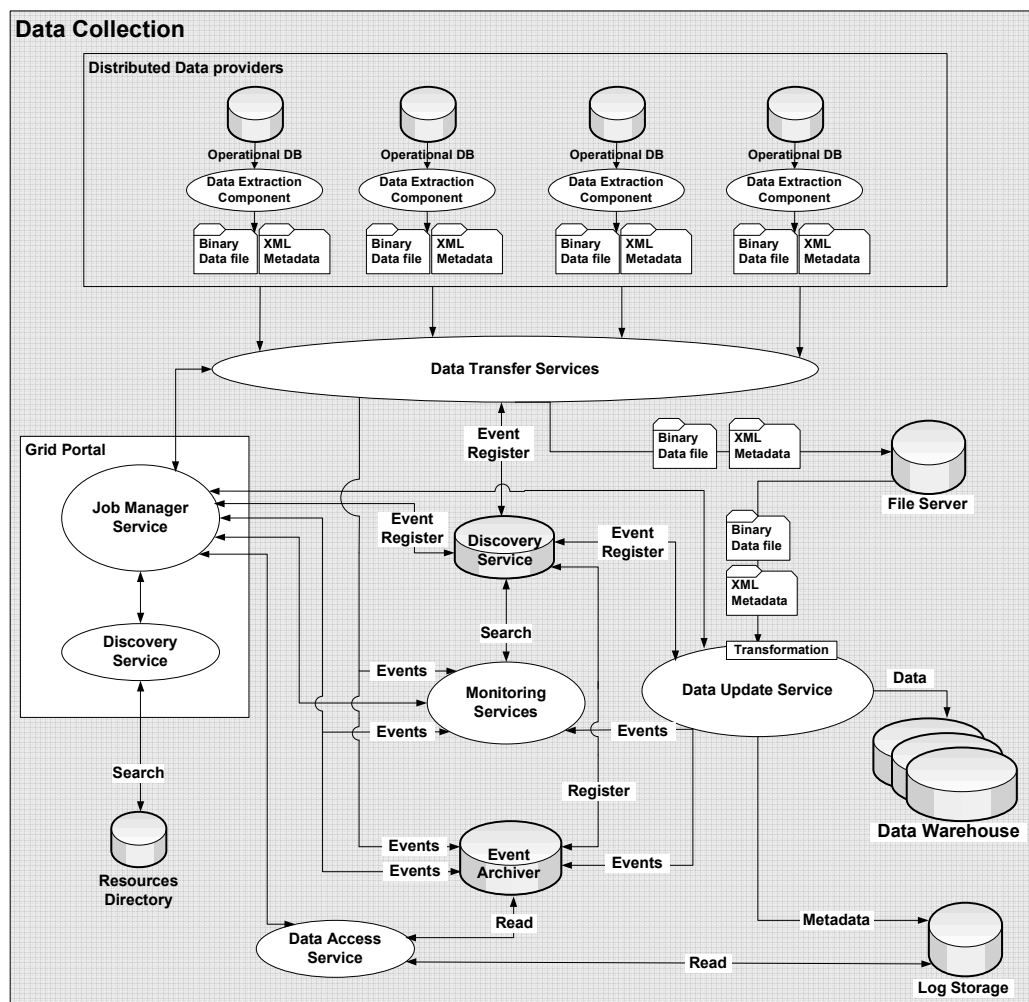


Figure 4.11 Data-Collection Model

#### **4.4.4.1. Data-Collection Model**

As discussed in section 1.1 and 3.3, data-collection is one of the main requirements of a Grid-enabled, distributed data warehouse system. It is required, e.g., to build data warehouses to aggregate HIV/AIDS patients' demographic data. Patients' data are collected from distributed data-providers through ETL processes. The data-collection model uses Grid services to control data transfer, data transformation, and the loading process. Additionally, the main processes are monitored through real-time monitoring services and logged into log storage. Figure 4.11 shows this data-collection model. It contains a number of defined services and illustrates the interactions among them. It can be described as follows:

- The patients' data is collected from physical distributed data-providers. The data extraction components are provided by data-providers according to a data collector's requirements. The required data is extracted and transformed into a specific format, depending on a data schema provided by a data collector. This data schema mainly describes what data (for example, patients' gender, age, race, diagnosis, symptom, treatment, etc.) will be extracted, and in what format (e.g., data type, sequence, etc.). Data schema is stored in a local metadata repository on the data provider's side. When the data extraction process is started, it needs to search its metadata repository to get required information. The extracted data will be exported into a binary data file with an XML metadata file. The metadata file contains the information, such as the data provider's description (e.g., name, location, etc.), file create-date, data volume, data collector's description, etc.
- On the data collectors' side, they need to transfer data and its metadata file to a file server periodically by invoking a data transfer service deployed on the Grid. The data update service is responsible for reading data files and metadata files from the file server, and performing transformation before inserting new data into the data warehouse. The transformation process at this stage performs operations, such as selecting certain columns, translating coded values, deriving new calculated values, summarizing

multiple rows of data, etc. The loading process is essentially inserting data into the data warehouse. The corresponding metadata will be inserted into log storage for future audit purposes. A data-access service can be used to read logged data from the log storage.

- A job manager is used to initiate, schedule, and execute jobs. It invokes data transfer service, data update services, and monitoring services, depending on the job description. Job manager services use discovery services to find the location of these deployed services and control their execution. Discovery services search resource directories to retrieve the location, and other information, of resources (services) and return this information to job manager services. The resource directory contains metadata for describing deployed services and resources. It may have multiple copies distributed at different locations.
- Data transfer, transformation and loading process are monitored during their processing by using real-time monitoring services. Data transfer services, data update services and job manager services register events in a directory service. The directory service is the storage for keeping entries used for searching events and their producers. The monitoring services subscribe all available event data for real-time visualization and performance analysis by searching the events directory. The events are also sent to the event archiver for non-real-time audit and analysis purpose. A data-access service can be used to read archived events.

#### ***4.4.4.2. Data-Access and Integration Model***

Cross-institutional data-access and integration are key capabilities of the Grid-enabled, distributed data warehouse system to facilitate data-mining and data analysis over distributed data warehouses. Data services comprise a rich set of functionalities to provide different capabilities such as federated access, distributed queries, data management, consistency management, metadata management, etc. Different combinations of these capabilities are used for different purposes, according to different requirements. This model only contains a small set of capabilities, which are essential for data-analysis tasks.

Figure 4.12 shows the model used for access and integrating distributed data. This model is described below:

- The model integrates various types of resources: the data-analysis components deployed within each institution, data warehouses, data marts, and single databases. Data-analysis services are treated as computational resources that can be involved in collaborative operations. Data warehouses and data marts are all considered as a groups of databases, which may use different data, different software systems to manage it, and different schema to describe it. On the other hand, a data service is also a kind of data resource. The output data from one data service may be the input of another data service. Whether computational resources, data resources or services, all resources are represented as WS-Resources. Resources publish their description information in a resources directory. A resources directory can have multiple copies distributed at different sites, locations or institutions.
- Data analysis needs to retrieve data from multiple distributed data resources or other data-analysis services. The data services provide a set of capabilities, including federation access, data transfer, data transformation, metadata catalogue, and consistency management. Data-analysis services invoke the subset of data services to complete specific tasks. Data services invoke discovery services to search the resource directory in order to look for available data resources. Discovery services respond to data services with data resources' description information (e.g., location, name, etc). Data services then can use this information to establish connections with these data resources and perform necessary operations.
- Data-analysis services can also invoke other data-analysis services from remote sites to perform some collaborative operations. The Discovery service is used to search available data-analysis services in the pre-defined VO.
- The responsibilities of job manager services and discovery service are as described in the data-collection model.
- This model uses the same monitoring mechanism as the data-collection

### Data Access and Integration

The diagram illustrates the Data Access and Integration architecture, showing the flow of data and services between various components:

- Distributed Data Resources:** This section includes four main data sources: Data Analysis Services, Data Marts, Data Warehouse, and Database. Each source is connected to a corresponding **WS-Resource** (Web Service Resource).
- Data Services:** A central hub containing Data Access, Transfer, Transformation, Federation, Consistency Management, and Metadata Catalogue. It acts as the core for data integration and management.
- Resources Directory:** A central directory that receives **Publish** notifications from WS-Resources and provides **Search** capabilities to the Grid Portal and Monitoring Services.
- Grid Portal:** Contains a **Discovery Service** and a **Job Manager Service**. The Discovery Service interacts with the Resources Directory and Monitoring Services. The Job Manager Service interacts with the Data Services and the Event Archiver.
- Monitoring Services:** Receives **Events** from the Data Services and the Event Archiver. It interacts with the Event Register and the Discovery Service.
- Event Register:** A central register for events, receiving **Register** notifications from the Discovery Service and the Event Archiver. It provides **Search** capabilities to the Monitoring Services.
- Discovery Service:** A service that interacts with the Event Register and the Event Archiver.
- Event Archiver:** Receives **Events** from the Data Services and the Event Register. It provides **Read** capabilities to the Data Access Service.
- Data Access Service:** A service that interacts with the Event Archiver and the Job Manager Service.

## 4.5. Summary

104

standard Web services approach to notification, using a topic-based publish/subscribe pattern.

This chapter defined a layered framework that consists of supportive Grid-related standards and specifications, including Web services, OGSA, WSRF, WS-Notification and GT. Based on these standards and specifications, a feasible system framework is proposed. OGSA offers a set of fundamental capabilities by defining Grid services in a general Grid context. Based on OGSA-services, the services required by the proposed Grid system are identified at different levels. The core services are built on infrastructure services that provide a set of common components, including naming, representing state, notification and security. The core-service set provides capabilities, such as job execution, system monitoring, data-access, data integration, resource management, etc. These services are finally integrated into two proposed models to achieve cross-institutional data-collection, data-access and integration.

## Chapter 5.

### Security

A Grid system is about resource-sharing. The proposed Grid system integrates a distributed and heterogeneous collection of locally managed users and resources, hosted by multiple institutions, which are members of the VO. Each institution may have a different security infrastructure to maintain the trust relationship within a single trust domain; in other words, the institutions in the VO are untrusted to each other. Obviously, establishing trust relationships between VO participants is critical to ensure secure cross-domain interaction. It is also necessary to provide standard security mechanisms that can be deployed to protect local institutions, while simultaneously allowing interoperable secure interaction. VO security is specifically concerned with user authentication and access control mechanisms for enforcing local and VO-wide policies for data-access and resource-usage.

Grid security is typically based on what is known as the Grid Security Infrastructure (GSI), which is now a GGF standard. It consists of a set of components for addressing different security issues (such as authentication, delegation and authorization) in Grid systems. The main advantage of GSI is that the general security issues are solved at infrastructure level rather than application level; the applications need only deal with application-specific policy. Version 4 of GSI (GSI4, <http://www.globus.org/toolkit/docs/4.0/security/>), corresponding to the GT4, integrates with the OGSA security mechanism to allow applications and users to operate in the Grid in a seamless and automated manner.

This chapter will identify the security issues in the Grid-enabled, distributed data warehouse system, and discusses how to use GSI4 components to provide a security solution for the problem-context in question. The envisaged security solution will discuss the use of Transport Level Security (TLS) protocol (Dierks & Rescorla, 2006) with X.509 public key certificate for authentication; X.509 proxy certificate for single sign-on (SSO) and delegation; the MyProxy protocol

(<http://grid.ncsa.uiuc.edu/myproxy/>) as an online credentials repository; and Shibboleth (<http://shibboleth.internet2.edu/>) plus GridShib (<http://grid.ncsa.uiuc.edu/GridShib/>), which integrate X.509 certificates with Security Assertion Markup Language (SAML) (OASIS, 2005<sup>1</sup>) for authorization.

The remainder of this chapter is organized as follows: The first two sections give a brief overview of information security principles and existing security technologies. Section 5.3 discusses the security challenges in the proposed Grid system. Section 5.4 introduces a high-level Grid security architecture which identifies the compulsory Grid security components. Section 5.5 focuses on the mechanisms implemented by GSI for addressing different security issues. Section 5.6 discusses the solution for the proposed Grid system.

## 5.1. A Brief Security Primer

Information Security is a discipline that relies on the synthesis of people, policy, education, training, awareness, procedures and technology to improve the protection of an organization's information assets. The goals of security are threefold:

- **Prevention:** prevent attackers from violating security policy.
- **Detection:** detect attackers' violation of security policy
- **Recovery:** stop an attack, assess and repair damage, and continue to function correctly, even if an attack succeeds.

Prevention is the ideal scenario. Detection occurs only after someone violates the security policy. Recovery implies that the attack has stopped (been stopped) and the system has been fixed.

The three classic security concerns (Whiteman & Mattord, 2003) of Information Security deal principally with data, and are:

- The **confidentiality** of information is the quality or state of preventing disclosure or exposure to authorized individuals or systems.

Confidentiality ensures that only those with the right and privileges to access a particular set of information are able to do so, and that those who are not authorized are prevented from obtaining access.

- **Integrity** is the quality or state of being whole, complete and uncorrupted. The integrity of information is threatened when the information is exposed to corruption, damage, destruction, or other disruption of its authentic state. The threat of corruption can occur while information is being stored or transmitted.
- **Availability** enables users who need to access information to do so without interface or obstruction, and to receive it in the required format. A user means not only a person, but also another computer system.

Additional concerns deal more with people and their actions:

- **Authentication**: Ensuring that users are who they say they are.
- **Authorization**: Making a decision about who may access data or a service.
- **Assurance**: Being confident that the security system functions correctly.
- **Non-repudiation**: Ensuring that a user cannot deny an action.
- **Auditability**: Tracking what a user did to data or a service.

Other security concerns relate to:

- **Trust**: People can justifiably rely on computer-based systems to perform critical functions securely and on these systems to process, store and communicate sensitive information securely.
- **Reliability**: The system does what you want, when you want it to.
- **Privacy**: Within certain limits, no one should know who you are or what you do.

## 5.2. Security Technology

Technology solutions of security, properly implemented, can maintain the confidentiality, integrity, and availability of information in each of its three

states: storage, transmission, and processing. This section will introduce some commonly used security technologies.

### **5.2.1. Firewalls**

A firewall, as part of an information security program, is any device that prevents a specific type of information from moving between the outside world, known as the un-trusted network (e.g., the Internet) and the inside world, known as the trusted network, and vice versa. It could be a hardware or software component added to a network to prevent communication forbidden by an organization's administrative policy.

Generally, there are two types of firewalls: traditional and personal. A traditionally firewall is, typically, a dedicated network device or computer positioned on the boundary of two or more networks. This type of firewall is used to filter all traffic entering or leaving the connected networks. A personal firewall is a software application used to filter traffic entering or leaving a single computer.

All traditional firewalls have the basic task of preventing intrusion on a connected network, but accomplish this in different ways: by working at the network and/or transport layer of the network. A network-layer firewall operates at the network level of the TCP/IP protocol stack. It undertakes IP-packet filtering, not allowing packets to pass the firewall unless they meet the rules defined by the firewall administrator. Application-layer firewalls operate at the application level of the TCP/IP protocol stack, intercepting, for example, all Web/HTTP, Telnet and FTP traffic. They will intercept all packets travelling to or from an application.

### **5.2.2. Intrusion Detection Systems**

Intrusion Detection Systems (IDSs) work like burglar alarms. When the alarm detects a violation of its configuration, it activates the alarm. This alarm can be audible and visual, or it can be a silent alarm that sends a message to a monitoring company. As with firewall systems, IDSs require complex

configuration to provide the level of detection and response desired. IDSs operate as either network-based, as when the technology is focused on protecting network information assets, or host-based, as when the technology is focused on protecting server or host information assets. IDSs use one of two detection methods, signature-based or statistical anomaly-based.

A host-based IDS resides on a particular computer or server, known as the host, and monitors activity on that system. Most host-based IDSs work on the principle of configuration or change management, in which the system records the file sizes, locations, and other attributes of the files, and then reports when one or more of these attributes change, when new files are created, and when existing files are deleted. It can also monitor systems' logs for pre-defined events. Network-based IDSs work differently. They monitor network traffic. When a pre-defined condition occurs, network-based IDSs respond and notify the appropriate administrator. Network-based IDSs require a much more complex configuration and maintenance program than do host-based IDSs. They must match known and unknown attack strategies against their knowledge base to determine whether or not an attack has occurred.

### 5.2.3. Cryptography

Cryptography is the realm of knowledge that deals with creating methods to assure that messages are secretly sent and received. Cryptography is the most commonly used means of providing security. It can be used to address four goals:

- **Message confidentiality:** Only an authorized recipient can extract the contents of a message from its encrypted form.
- **Message integrity:** The recipient should be able to determine if the message has been altered during transmission.
- **Sender authentication:** The recipient can identify the sender, and verify that the purported sender did send the message.
- **Sender non-repudiation:** The sender cannot deny sending the message.

Obviously, not all cryptographic systems or algorithms realize, nor intend to, achieve all of these goals. Cryptosystems are manual or computer-based systems used to encrypt or transform data for secure transmission and storage. The following several sections will introduce several popular cryptosystems briefly.

#### **5.2.3.1. *Symmetric Cryptosystems***

Symmetric cryptography, also known as private key cryptography, is built on symmetric encryption and uses a single key for both encryption and decryption of data. Each participant in the secure communication must possess his or her own set of the identical keys. Secure communication can be accomplished over insecure channels with a symmetric cryptosystem. Because the symmetric encoding and decoding algorithm is public, the level of security generated by a symmetric cipher depends on the key length and the system's ability to protect the key.

IBM's Lucifer algorithm, which was originally based on a key length of 128-bits, was modified to a key length of 56-bits, renamed Data Encryption Standard (DES), and adopted as a standard for encryption of non-classified information. DES consists of two components: an algorithm and a key. The DES algorithm involves a number of iterations of a simple transformation which uses both transposition and substitution techniques applied alternatively. DES is a so-called private-key cipher. Data is encrypted and decrypted with the same key. The DES algorithm is publicly known; thus, learning the encryption key would allow an encrypted message to be read by anyone.

#### **5.2.3.2. *Asymmetric Cryptosystems***

Asymmetric cryptography, also known as public key cryptography, uses a key pair consisting of a public key and private (secret) key. The public key encrypts, but cannot decrypt. Asymmetric encryption is popular because one of the keys can be published and widely distributed, thereby allowing anyone to use another's public key to encrypt data; however, only the person with the

corresponding private key can decrypt the data. This makes encrypted data secure, as long as the private key remains secure.

An example of a public-key cryptosystem is RSA, which was published by Ronald Rivest, Adi Shamir and Leonard Adleman (RSA). The patented RSA algorithm has become the *de facto* standard for public-use encryption applications. RSA provides authentication, as well as encryption, and uses two keys: a private key and a public key. With RSA, there is no distinction between the function of a user's private and public keys. The keys are generated mathematically. The security of the RSA algorithm depends on the use of very large numbers (RSA uses 256- or 512-bit keys).

With both symmetric and asymmetric cryptosystems, there is a need to secure the private key. The private key must be kept private. The stored keys should always be password protected. Another issue, with key-based systems, is that the algorithms that are used are public. This means that the algorithms could be coded and used to decrypt a message via a brute force method of trying all the possible keys. However, such a program would need a significant amount of computational power to accomplish such a process. With keys of sufficient length, the time to decode a message would be unreasonable.

#### **5.2.3.3. *Digital Signatures***

Digital signatures, based on the Digital Signature Standard (DSS), have been widely adopted for authenticating information. The DSS approach uses a hash function to create a message digest, which is then input into the digital signature algorithm with a random number to generate a digital signature. Integrity is guaranteed in public-key systems by using digital signatures. Most digital signatures rely on public-key cryptography to work. In this case, the digital signature function depends upon the sender's private key. The encrypted message containing the digital signature is then verified by the recipient by using the sender's public key. A strong hash function is applied to the message, and the resulting message digest is encrypted instead of the entire message, which makes the signature significantly shorter than the message and saves considerable time.

#### **5.2.3.4. *Digital Certificates***

Digital certificates are electronic documents issued by a reputable third party that certify the identity of a user and the proof of identification associated with the presentation of a public key. A certificate authority (CA) issues, manages, and authenticates signs and revokes a digital certificate containing the user's name, public key, and other identifying information. In contrast to a digital signature, which helps authenticate the origin of a message, a digital certificate authenticates the company that provides the verification of the digital signature's authenticity. Time and data stamps may be included, as a CA validates the identity of a certificate requestor, issues the electronic certificate, and certifies to recipients that the entity presenting the certificate is, in fact, who it claims to be. The ITU-T (International Telecommunication Union Telecommunication Standardization Sector) X.509 version 3 (Housley, Polk, Ford & Solo, 2002) and Pretty Good Privacy (PGP, <http://www.ietf.org/html.charters/openpgp-charter.html>) (Callas, Donnerhacke, Finney & Thayer, 1998) are popular certificates used today.

#### **5.2.3.5. *Public Key Infrastructure***

Public Key Infrastructure (PKI) is an integrated structure of software, encryption methodologies, protocols, legal agreements, and third-party services that enables users to securely communicate across the insecure Internet. Third-party suppliers integrate public key cryptography, digital certificates, and certification authority into an enterprise-wide solution to provide authenticated and secure communications between participants. PKI protects information assets in several ways, including authentication, integrity, privacy, authorization and non-repudiation. A typical PKI solution protects the transmission and reception of secure information by integrating the following components:

- A **certificate authority (CA)** that issues, manages, authenticates, signs, or revokes a digital certificate containing the user's name, public key, and other identifying information.
- A **registration authority (RA)** that operates under the trusted collaboration of the certificate authority and can be delegated day-to-day

certification functions, such as verifying registration information about new registrants, generating end-user keys, revoking certificates, and validating that users possess a valid certificate.

- **Certificate directories** are central location for certificate storage, providing a single access point for administration and distribution.
- **Management protocols** organize and manage the communications between CAs, RAs, and end users. This includes the functions and procedures to register and initialize new users, recover, update, and revoke keys, as well as enable the transfer of certificates and status information among the parties involved in a PKI trust.
- **Policies and procedures** assist an organization in the application and management of certificates, formalization of the legal liabilities and limitations, and actual business practice use.

Certificates are electronic containers for the key values needed for the use of a cryptosystem. The CA manages the housekeeping details of tracking who has been assigned which key, providing a directory of public key values for use across the organization, assisting users in safeguarding their private keys, and helping the organization manage common workplace events that could threaten the safety of keys in use. If the trust relationship is broken, such as when a private key has been comprised or the key holder no longer has authority to manage the key, the certificate may be revoked. The CA periodically distributes a certificate revocation list (CRL), which contains a signed time-stamped listing of all revoked certificates. Key pairs provide encryption and non-repudiation required for secure transaction. The key pair can be generated either by the end user or by the CA.

### 5.3. Grid Security Problems

The Grid-enabled, distributed data warehouse system is a Grid system that integrates distributed heterogeneous resources hosted by multiple institutions in a VO. The dynamic nature of a Grid system makes it difficult to entirely establish trust relationships between sites prior to application execution.

As discussed in section 3.2, the user population in the pre-defined VO is large and dynamic. Participants in VOs, such as ones for scientific collaboration, will include members of many institutions and will change frequently. A job may acquire, start processes on, and release resources dynamically during its execution. While these processes form a single, fully connected, logical entity, low-level communication connections (e.g., TCP/IP sockets) may be created and destroyed dynamically during program execution. Resources may require different authentication and authorization mechanisms and policies. These may include Kerberos, plaintext passwords, TLS, and Secure Shell (SSH , <http://www.openssh.com>). An individual user will be associated with different local name spaces, credentials, or accounts, at different sites, for the purposes of accounting and access control. At some sites, a user may have a regular account. At others, the user may use a dynamically assigned guest account, or simply an account created for the collaboration. In brief, the security solution for a Grid system is about coordinating diverse access control policies and operating securely in a heterogeneous environment. In this section, the security challenges will be discussed in the context of the proposed Grid system.

### 5.3.1. Terminology

Firstly, some Grid-based security terminology needs to be clarified:

- In Grid systems, a **subject** is generally a user, a process operating on behalf of a user, a resource, or process acting on behalf of a resource.
- An **object** is a resource that is being protected by the security policy.
- A **trust domain** is a collection of both subjects and objects governed by a single administration and a single security policy.
- A **credential** is a piece of information that is used to prove the identity of a subject (such as a password and a certificate).

Generally, subjects provide data from their own data warehouse, data marts, and near-line operational data sources as Grid resources. Typically, a Grid portal would provide a brokered interface to Grid applications. A Grid application

may be responsible for invoking specific data services for completing user-group-specific tasks.

### **5.3.2. Security requirements**

Based on the characteristics of pre-defined VO and the system design discussed in previous chapters, the security requirements for the Grid-enabled, distributed data warehouse system are summarized as follows:

- This system integrates distributed users with data resources, which are managed locally by their owners. Each institution can be thought as a trust domain. Operations that are confined to a single trust domain are subject to local security policy. Operations across multiple trust domains require multiple authentications that allow a user, the processes and the resources used by those processes, to verify each other's identity.
- Inter-domain access requires, at a minimum, a common way of expressing the identity of a security principal, such as an actual user or a resource. Hence, it is imperative to employ a standard for encoding credentials for security principals.
- As both global and local subjects exist, a remote user can have a global user name, used to access the services portal, and also a local user name, defined by a local trust domain. For persisting credentials, a global subject (identity and role) can be mapped to a local subject (identity and role). Local security policy will dictate the permission sets of virtual local (mapped) and true local subjects.
- User credentials (such as passwords, private keys, etc.) must be protected during interaction either across different trust domains or within a single trust domain. A secure transmission protocol is required to ensure privacy and integrity when these credentials are transported through the network.
- One user's request may involve many processes on many distributed resources. It is necessary that a user only sign-on once for a long-lived program or process without further authentication. A program or process should be allowed to act on behalf of a user and be delegated a subset of the

user's rights. Processes running on behalf of the same subject within the same trust domain may share a single set of credentials.

- VO resources are located within multiple institutions. Inter-domain access mechanism should be provided. Access to local resources will typically be determined by local security policy that is enforced by local security mechanisms. Each institution retains ultimate control over the (local) policies that control access to its resources. The inter-domain security used for the proposed Grid must be able to interoperate with, rather than replace, the diverse intra-domain access control technologies.

To summarize, the security solution for the Grid-enabled, distributed data warehouse system needs to allow computations to coordinate diverse access control policies and to operate securely in a heterogeneous and dynamic environment.

### **5.3.3. Security policy**

Based on the system security requirements, a security policy that addresses requirements for SSO, interoperability with local policies, and dynamically varying resource requirements, will be discussed in this section. The policy focuses on authentication of users, resources and processes and supports user-to-resource, resource-to-user, process-to-resource, and process-to-process authentication.

The security policy can be examined as follows:

- The envisaged Grid environment consists of multiple trust domains. This policy states that the Grid security policy must integrate a heterogeneous collection of locally administered users and resources. The Grid security policy must focus on controlling the inter-domain interactions and the mapping of inter-domain operations into local security policy.
- Operations that are confined to a single trust domain are subject to local security policy only. No additional security operations or services are imposed on local operations by the Grid security policy. The local security policy can be implemented by a variety of methods, including

firewalls, Kerberos and SSH.

- Both global and local subjects exist. For each trust domain, there exists a partial mapping from global to local subjects. Each user of a resource will have two names: a global name and a potentially different local name on each resource. A site might map global user names to: a predefined local name, a dynamically allocated local name, or a single group name.
- Operations between entities located in different trust domains require mutual authentication (or two-way authentication) which refers to two parties authenticating each other suitably. In technology terms, it refers to a client or user authenticating themselves to a server and that server authenticating itself to the user in such a way that both parties are assured of the other's identity.
- An authenticated global subject mapped into a local subject is assumed to be equivalent to being locally authenticated as that local subject.
- All access-control decisions are made locally on the basis of the local subject.
- A program or process is allowed to act on behalf of a user and be delegated a subset of the user's rights. This policy element is necessary to support the execution of long-lived programs that may acquire resources dynamically without additional user interaction. It is also needed to support the creation of processes by other processes.
- Processes running on behalf of the same subject within the same trust domain may share a single set of credentials.

## **5.4. A Grid Security Architecture**

In this section a high-level Grid security architecture (Foster et al., 1998) is introduced, as illustrated in Figure 5.1. Two types of proxy are defined: a user proxy and a resource proxy. Four related protocols are defined: user-proxy-creation protocol, resource-allocation protocol, resource-allocation-from-a-process protocol and mapping-registration protocol.

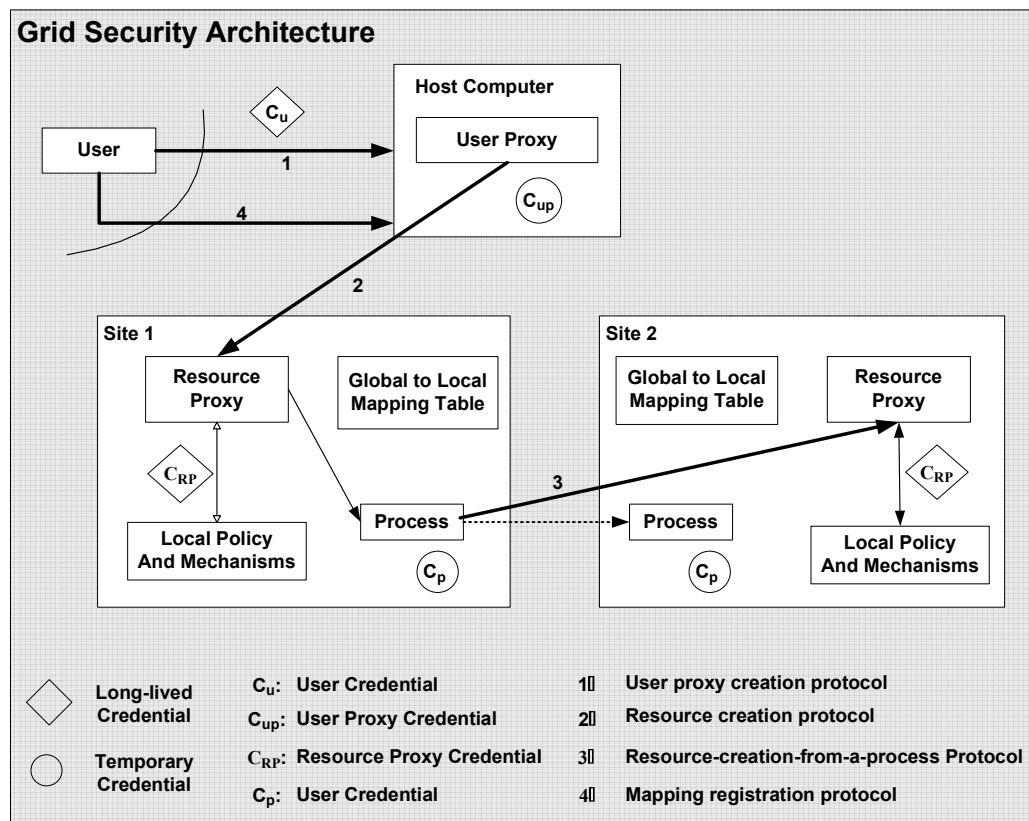
A user proxy is a session-manager process given permission to act on behalf of a user for a limited period of time. Once the user proxy has been created, the user may be disconnected in order to eliminate the need to have the user's credentials available for every security operation. It reduces the possibility of the credentials being compromised during operations. Additionally, the lifetime of the user proxy credentials is under control of the user. A resource proxy is an agent used to translate between inter-domain security operations and intra-domain (local) mechanisms. It is allocated by the user proxy, and is responsible for scheduling the access to a resource and for mapping a computation onto that resource.

When a principal logs on to the Grid system, it creates a user proxy by using the user-proxy-creation protocol. The user proxy then allocates a resource and creates processes by using the resource-allocation-protocol. A process may allocate additional resources by using the resource-allocation-from-a-process protocol. The mapping-registration protocol can be used to define a mapping from a global subject to a local subject. The following should be noted in this regard:

- User proxy credentials should be signed by the user's long-lived credentials and contain all information (user-id, local host name, etc.) required for authentication. The integrity of user proxy credentials is protected by local security policy.
- A user proxy requiring access to a resource first determines the identity of the resource proxy for that resource. It then issues a request to the appropriate resource proxy. If the request is successful, the resource is allocated, and a process created on that resource. The request can fail because the resource is not available, or because of authentication failure or authorization failure.
- The resource-allocation protocol is used to issue a request to a resource proxy from a user proxy. The user proxy and resource proxy authenticate each other. The resource proxy checks if the user who signed the proxy's credentials is authorized by local policy to make the allocation request. At this time, the verification may require accessing a mapping table maintained

by the resource proxy (for mapping the user's credentials onto a local user-id). A single resource allocation request may result in the creation of multiple processes on the remote resources. All such processes are created with the same credentials.

- It is a common case that the resource allocation is initiated dynamically from a process created by a previous resource-allocation request. The user proxy decides whether to honour the request through authentication between user proxy's credentials and process credentials. The resulting process handle is signed by the user proxy and returned to the requesting process.



**Figure 5.1 Grid Security Architecture**

This approach uses a user proxy to interact with the resource proxy to achieve SSO and delegation. Authentication occurs between a user proxy and a resource proxy. Consequently, the SSO leverages the existing trust relationship between a user and a resource that was established when the user was initially granted access to the resource. The user proxy and process

authenticate each other when a resource allocation request is issued by a process. The resource-allocation request is successful only when the user is authorized by the resource on the basis of local policy.

## **5.5. Grid Security Infrastructure**

The architecture discussed in the previous section defined protocols in abstract terms, rather than in terms of specific security technologies. Hence, these protocols can be implemented by using any modern security technologies and mechanisms. Grid Security Infrastructure (GSI), as it appears in the GT, is essentially an implementation of the architecture discussed in previous section. It is based on a Public Key Infrastructure (PKI) with CAs and X.509 certificates. It provides: a public-key system; mutual authentication through digital certificates; credential delegation and SSO. GSI defines a set of protocols, libraries, and tools that allow users and applications to securely access resources. It acts as Grid security middleware to provide infrastructure-level security functionalities for addressing security issues in a Grid environment. In the following sub-section, the mutual authentication, dynamic delegation and authorization mechanisms used by GSI will be introduced.

### **5.5.1. Authentication**

Authentication between two entities (users and resources) on remote Grid nodes means that each party establishes a level of trust in the identity of the other party. An authentication protocol sets up a secure communication channel between the authenticated parties, so that subsequent messages can be sent without repeated authentication steps, although it is possible to authenticate every message. The identity of an entity is, typically, some token or name that uniquely identifies the entity.

VOs need a reliable means for identifying requestors, but participant independence complicates authentication across multiple sites. Without an integrated authentication, VOs have used a variety of ad hoc schemes to achieve

resource-sharing, such as giving users an account at each institution with distinct login names and passwords. This multiplicity of mechanisms and passwords makes access difficult, discouraging information sharing and collaboration. It also hinders the creation of software that securely spans resources at multiple institutions or that allows secure collaboration between users at multiple institutions.

#### **5.5.1.1. *Kerberos and SSH***

Kerberos (Neuman, Yu, Hartman & Raeburn, 2005) and SSH are two widely used approaches for multi-site authentication. However, they do not meet VO authentication requirements.

Kerberos is used alone or under the distributed computing environment. It authenticates users through a secure transaction with a centrally maintained key server. Kerberos achieves inter-organizational, or cross-realm, authentication by designating trustworthy key servers in other organizations. Kerberos meets many of the basic requirements for VO authentication. However, Kerberos requires that all cross-domain trust be established at the domain level, meaning that organizations have to agree to allow cross-domain authentication, which can often be a heavy-weight administrative process (Neuman & Ts'o, 1994).

SSH is a widely used login technology (Daniel, Silverman & Byrnes, 2005) and meets a number of VO authentication requirements. It is based on public-key authentication technology, uses link encryption to protect user credentials, and is easily deployed. It provides basic remote login and file copy capabilities without a lot of complexity. SSH provides a strong system of authentication and message protection, but has no support for translation between different mechanisms or for creation of dynamic entities.

#### **5.5.1.2. *Using PKI for Authentication***

The GSI authentication mechanism is based on PKI (Thompson, Olson, Cowles, Mullen & Helm, 2003). GSI provides libraries and tools for authentication

and message protection that use standard X.509 public key certificates with the TLS protocol.

X.509 v3 Structure	
Version	Certificate:
Certificate serial number	Data:
Algorithm identifier Algorithm Parameters	Version: 3 (0x2) Serial Number: 1 (0x1) Signature Algorithm: md5WithRSAEncryption Issuer: C=ZA, ST=Western Cape, L=Cape Town, O=Thawte Consulting cc, OU=Certification Services Division, CN=Thawte Server CA/Email=server-certs@thawte.com
Issuer name	Validity Not Before: Aug 1 00:00:00 1996 GMT Not After : Dec 31 23:59:59 2020 GMT
Validity Not before/Not after date	Subject: C=ZA, ST=Western Cape, L=Cape Town, O=Thawte Consulting cc, OU=Certification Services Division, CN=Thawte Server CA/Email=server-certs@thawte.com
Subject name	Subject Public Key Info: Public Key Algorithm: rsaEncryption RSA Public Key: (1024 bit) Modulus (1024 bit): 00:d3:a4:50:6e:c8:ff:56:6b:e6:cf:5d:b6:ea:0c: 68:75:47:a2:aa:c2:da:84:25:fc:a8:f4:47:51:da: 85:b5:20:74:94:86:1e:0f:75:c9:e9:08:61:f5:06: 6d:30:6e:15:19:02:e9:52:c0:62:db:4d:99:9e:e2: 6a:0c:44:38:cd:fe:be:e3:64:09:70:c5:fe:b1:6b: 29:b6:2f:49:c8:3b:d4:27:04:25:10:97:2f:e7:90: 6d:c0:28:42:99:d7:4c:43:de:c3:f5:21:6d:54:9f: 5d:c3:58:e1:c0:e4:d9:5b:b0:b8:dc:b4:7b:df:36: 3a:c2:b5:66:22:12:d6:87:0d Exponent: 65537 (0x10001)
Subject public key information Algorithm Parameters Key	X509v3 extensions: X509v3 Basic Constraints: critical CA:TRUE
Issuer unique identifier (optional)	Signature Algorithm: md5WithRSAEncryption 07:fa:4c:69:5c:fb:95:cc:46:ee:85:83:4d:21:30:8e:ca:d9: a8:6f:49:1a:e6:da:51:e3:60:70:6c:84:61:11:a1:1a:c8:48: 3e:59:43:7d:4f:95:3d:a1:8b:b7:0b:62:98:7a:75:8a:dd:88: 4e:4e:9e:40:db:a8:cc:32:74:b9:6f:0d:c6:e3:b3:44:0b:d9: 8a:6f:9a:29:9b:99:18:28:3b:d1:e3:40:28:9a:5a:3c:d5:b5: e7:20:1b:8b:ca:a4:ab:8d:e9:51:d9:e2:4c:2c:59:a9:da:b9: b2:75:1b:f6:42:f2:ef:c7:f2:18:f9:89:bc:a3:ff:8a:23:2e: 70:47
Subject unique identifier (optional)	
Extensions Type Criticality Value	
Signature	

*Figure 5.2 X.509 v3 Certificate Structure*

GSI uses X.509 public key certificates and TLS for authentication for several reasons. It is not only because these are well-known technologies with readily available, well-tested open source implementations. The flexibility of trust model for X.509 certificates was a deciding factor between X.509 certificates and other common authentication mechanisms. It means the trust model of X.509 certificates allows an entity to trust another organization's CA, without requiring that the rest of its organization does so, or requiring reciprocation by the trusted CA.

ITU-T X.509 version 3 certificate is one commonly used identity token for authentication (ITU-T, 2005). It is the public key of a user, together with some other information, rendered unforgeable by encipherment, with the private key

of the certification authority which issued it. Figure 5.2 illustrates the structure of the X.509 public key certificate.

TLS is a cryptographic protocol that provides communications privacy over the Internet. The primary goal of the TLS protocol is to provide privacy and data integrity between two communicating applications. In typical use, only the server is authenticated (i.e., its identity is ensured), while the client remains unauthenticated; mutual authentication requires PKI deployment to clients. It is composed of two layers: the TLS Record Protocol and the TLS Handshake Protocol. At the lowest level, layered on top of some reliable transport protocol (e.g., TCP), is the TLS Record Protocol. TLS allows client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery. TLS has a variety of security measures:

- The TLS protocol exchanges records: each record can be optionally compressed, encrypted and packed with a message authentication code (MAC). TLS numbers all the records and uses the sequence number in the MACs.
- Using a message digest enhanced with a key (so only with the key can you check the MAC) (Krawczyk, Bellare & Canetti, 1997).
- Protection against several known attacks (including man-in-the-middle attacks), like those involving a downgrade of the protocol to a previous (less secure) version or a weaker cipher suite.
- The message that ends the handshake (“Finished”) sends a hash of all the exchanged data seen by both parties.
- The pseudorandom function splits the input data in half and processes each one with a different hashing algorithm (MD5 and SHA), then XORs them together. This provides protection if one of these algorithms is found to be vulnerable.

In practice, GSI uses a X.509 public key certificate as an identity token. An X.509 certificate contains a public key, a subject name in the form of a multi-component distinguished name (DN), and a validity period and is signed by a trusted third party, or CA. The associated private key is owned by the

correct remote subject with whom an encryption or digital signature mechanism will be used. The standards' documents refer to these certificates as "public key certificates" or "X.509 certificates". The term "identity certificates" are used to emphasize their use to securely identify an entity in a Grid environment.

GSI uses the X.509 certificates with the TLS protocol to ensure a secure, authenticated connection between two parties. A GSI user generates a public and private key pair and obtains an X.509 certificate from a trusted CA. X.509 certificates are exchanged between entities (users and resources). The certificates are first tested by checking the expiration dates, possible revocation, acceptable key usage, and signature, by a trusted CA. If the certificates pass all these checks, their public keys are then used to build a challenge handshake to prove that each entity that sent a certificate has the corresponding private key. Passing these tests gives each party a level of confidence that it has established a secure connection to the party represented by the certificate presented. The X.509-TLS infrastructure supports multiple, independent CAs. In a Grid, each site may choose which CAs it will accept for binding DNs and public keys. Most of the current Grid tools are built on GSI or HTTP, both of which use X.509 certificates for securely establishing a Grid identity. The veracity of an entity's identity is only as good as the trust placed in the CA that issued the certificate, so the local administrator installs these certificates, which are then used to verify the certificate chains. The assurance provided by using TLS with mutual authentication depends on the correctness of the TLS and certificate validation implementation at all the sites that take part in establishing a secure connection, the diligence of the individual in protecting the private key, and the certificate policy (CP) and certification practice statement (CPS) of the trusted CAs

#### **5.5.1.3. *Grid Certificate Authority***

A Grid CA is defined as a CA that is independent of any single organization and is responsible for signing certificates for individuals allowed to access the Grid resources, hosts or services running on a single host (Thompson et al., 2003).

Grid CA is substantially different from a traditional organizational CA. Organizational CA only issues certificates for members of its organization, and these certificates are used to access resources within this organization. In identity certificates issued by an organizational CA, the DN often contains a number of attributes (e.g., organizational unit, location, and email) retrieved from the organization's directory (such as X.500 and LDAP directory). Since a Grid CA is independent of the organizations to which its subscribers belong, it does not have a way to verify much information about a subscriber or to know when such information changes. The prudent approach for a Grid CA is to put as little information in the certificate as possible. According to several Grid projects, such as CERN CA (<http://globus.home.cert.ch/globus/ca/>) and DOE Science Grids CA (<http://www.doeagrids.org>), a minimal set of information of a DN contains:

- An organization element that identifies the Grid to which the CA belongs.
- A class designator that identifies the certificate as representing a person, host, or service, which is intended to be used when storing and retrieving certificates in the Grid CA's publishing directory
- A common name that reasonably identifies the entity for which the certificate is issued.

Since the operator of a Grid CA does not personally know the persons who are requesting certificates and does not have access to a trusted directory of such users, he/she must rely on registration agents (RAs). These are individuals who are likely to know a subset of subscribers first-hand or second-hand. If the users of a Grid can be grouped by actual or VOs, an RA may be chosen for each such organization and given the responsibility to approve requests from members of that organization only. The rules for establishing member identities should be published by each RA, and the procedures for verifying the identities and certificate requests should be consistent among all the RAs and approved by the CA.

### **5.5.2. SSO and delegation**

The establishment of X.509 public key certificates and their issuing certification authorities provides a sufficient authentication infrastructure for persistent entities in Grids. However, X.509 certificates cannot cover SSO and dynamic delegation requirements in Grids well.

It is often the case that a Grid user needs to delegate some subset of their privileges to another entity on relatively short notice and only for a brief amount of time. For example, a user needing to move a dataset, in order to use it in a computation, may want to grant to a reliable file transfer service the necessary rights to access the dataset and storage, so that it may perform a set of file transfers on the user's behalf. In addition to delegation to persistent services and entities, the requirement exists to support delegation of privileges to services that are created dynamically, often by the users themselves that do not hold any form of identity credential. For example, a user wants to access data or start a sub-job on other resource. The point is that the user wants to delegate privileges specifically to the job and not to the resource as a whole. It is common practice to protect the private keys associated with X.509 public key certificates, for example, by encrypting them with a pass phrase. This poses a burden on users, who need to authenticate repeatedly in a short period of time, and occurs frequently in Grid scenarios when a user is coordinating a number of resources. In a nutshell, Grids need authentication solutions that allow users to create identities for new entities dynamically, in a light-weight manner, to delegate privileges to those entities in a dynamic, light-weight manner, to perform SSO, and that allows for the reuse of existing protocols and software with minimal modifications. Based on these requirements, X.509 Proxy Certificate (Tuecke, Welch, Engert & Thompson, 2004) is defined and standardized, and used in GSI for delegation and SSO (Welch et al., 2004).

#### **5.5.2.1. *Proxy Certificates***

Proxy credentials are commonly used in security systems when one entity needs to grant to another entity some set of privileges. Proxy Certificates allow an

entity holding a standard X.509 public key certificate to delegate some or all of its privileges to another entity, which may not hold X.509 credentials at the time of delegation.

Proxy Certificates use the format prescribed for X.509 public key certificates, with the prescriptions described in this paragraph. The use of the same format as X.509 public key certificates allows Proxy Certificates to be used in protocols and libraries in many places, as if they were normal X.509 public key certificates, which significantly eases implementation. Unlike a public key certificate, the issuer (and signer) of a Proxy Certificate is identified by a public key certificate or another Proxy Certificate rather than a CA certificate. This allows Proxy Certificates to be created dynamically without requiring the normally heavy-weight vetting process associated with obtaining public key certificates from a CA. The subject name of a Proxy Certificate is scoped by the subject name of its issuer to achieve uniqueness. This is accomplished by appending a CommonName relative distinguished name (RDN) component to the issuer's subject name. The value of this added CommonName RDN is statistically unique to the scope of the issuer. The value of the serial number in the Proxy Certificate is also statistically unique to the issuer. The public key in a Proxy Certificate is distinct from the public key of its issuer and has different properties (e.g., its size may be different). All Proxy Certificates must bear a newly-defined critical X.509 extension, the Proxy Certificate Information (PCI) extension. The PCI extension use a framework for carrying policy statements to allow the issuers to express their desire to delegate rights to the Proxy Certificate bearer, and to limit further Proxy Certificates that can be issued by that Proxy Certificate holder. The existing policy language (e.g., XACML (OASIS, 2005<sup>2</sup>)) can be used to express delegation policies. This use of arbitrary policy expressions is achieved through two fields in the PCI extension: a policy method identifier and a policy field. The policy method identifier is an object identifier (OID) that identifies the delegation policy method used in the policy field. The policy field then contains an expression of the delegation policy that has a format specific to the particular method. There are two policy methods that are defined. The PCI extension also contains a field expressing the maximum path lengths of Proxy Certificates that can be issued by the Proxy

Certificate in question. A value of zero for this field prevents the Proxy Certificate from issuing another Proxy Certificate. If this field is not present, then the length of the path of Proxy Certificates, which can be issued by the Proxy Certificate, is unlimited.

The validation of Proxy Certificates is described in RFC 3280 (Housley et al., 2002) and Proxy Certificate RFC3820 (Tuecke et al., 2004). The Proxy Certificates are created with short life spans, typically in the order of hours. There currently exists no implemented method for revocation of Proxy Certificates. It can use the same mechanism with public key certificate to revoke a Proxy Certificate.

#### **5.5.2.2. *Uses for SSO***

Proxy Certificates enable SSO that allows the user to manually authenticate once in order to create a Proxy Certificate which can be used repeatedly to authenticate for some period of time without compromising the protection on the user's long-term private key. This is accomplished by creating a new key pair (composed of a public and private key). The user's private key associated with their long-term public key certificate is accessed to sign the certificate request containing the public key of the newly generated key pair; hence generating a Proxy Certificate. The Proxy Certificate binds the new public key to a new name and delegates some or all of the user's privileges to the new name. The Proxy Certificate and the new private key are then used by the bearer to authenticate to other parties. Generally, the Proxy Certificate private key is stored on a local file system and is protected by only local file system permissions, which allows the user's applications to access it without any manual intervention by the user.

#### **5.5.2.3. *Uses for Delegation***

Proxy Certificates can also be created so as to delegate privileges from an issuer to another party over a network connection without the exchange of private keys. Firstly, two involved parties perform mutual authentication, the initiator using its existing Proxy Certificate and the target entity uses the public key

certificate of its own. After authentication, an integrity protected channel is established. These two steps can be accomplished by using the TLS protocol. After the initiator expresses its desire to delegate by some application-specific means, the target entity generates a new public and private key pair. With the new public key, a signed certificate request is created and sent back over the secured channel to the initiator. The initiator uses the private key associated with its own Proxy Certificate to sign the certificate request, generating a new Proxy Certificate containing the newly generated public key from the target service. The new Proxy Certificate is sent back over the secured channel to the target entity, which places it into a file with the newly generated private key. This new Proxy Certificate is then available for use on the target service for applications running on the user's behalf.

### **5.5.3. Authorization**

In a Grid environment, each institution in a VO typically retains ultimate control over diverse, complex and dynamic policies that govern who can use which resources for which purpose. Access to local resources will typically be determined by a local security policy that is enforced by a local security mechanism. However, the VO will often wish to apply some common policy about how its users access the resources assigned to the VO. A key problem associated with the formation and operation of distributed VO is that of how to specify and enforce community policies. This section primarily introduces a conceptual Grid authorization framework (Lorch et al., 2004), which is built on the Authorization Framework, presented in RFC2904 (Vollbrecht et al., 2004) and the "Generic AAA Architecture", presented in RFC2903 (de Laat, Gross, Gommans, Vollbrecht & Spence, 2000) of the IRTF AAA Architecture Research Group, as well as the Access Control Framework, described in the ISO recommendation (ITU-T, 1996). With regard to this framework, there exist several authorization mechanisms and systems that are used by various Grid and other applications to address the authorization concerns. For example, Community Authorization Service (CAS) is provided by GSI to implement access control in dynamically created overlaid trust domains.

#### **5.5.3.1. *Grid Authorization Framework Concepts***

The term ‘authorization’ may mean the process of issuing a proof of right, the proof of right itself, and the process of making an authorization decision by checking a proof of right. In principle, authorization decisions are made based on authorization information provided by authorities. These authorities must have a direct or a delegated relationship with either the authorization subject (e.g., user or organization member to which the authorization is issued), or with the resource that is the target of the request that prompted the authorization (e.g., owner or administrator of a resource), or with both.

The authorization may involve three basic high-level entities: subject, resource and authority. A ‘subject’ is an entity that can request, receive, own, transfer, present or delegate an electronic authorization so as to exercise a certain right. The subject may be identified as an individual user or as a member of a group of users. A subject may also be a process that acts on behalf of a user and, as such, holds access rights that were delegated to it from the user. The subject may define a set of policies that determine how its authorization is used. A Grid environment consists of a large number of diverse resources. A component of the system provides or hosts services and may enforce access to these services based on a set of rules and policies defined by entities that are authoritative for the particular resource. Access to resources may be enforced by a resource itself or by some entity (a policy enforcement point, gateway) that is located between a resource and the requestor, thus protecting the resource from being accessed in an unauthorized fashion. ‘Authority’ is an administrative entity that is capable of and authoritative for issuing, validating and revoking an electronic means of proof so that the named subject (a.k.a. holder) of the issued electronic means is authorized to exercise a certain right or assert a certain attribute. Right(s) may be implicitly or explicitly present in the electronic proof. A set of policies may determine how authorizations are issued, verified, etc., based on the contractual relationships the authority has established. There are currently three general types of authorities in common use: attribute authority, policy authority and identity authority. Attribute authority issues attributes assertions that a given subject has one or more attribute/value pairs. Policy authority issues

authorization policies with respect to resources and services offered by these resources. These authorization policies contain assertions that a given subject has a certain right with respect to a given service. Identity authority (e.g., the CAs of a PKI) issues certificates that assert a mapping of cryptographic tokens to subject identities. Identity authority enables authentication rather than authorization. Each of these three entities may implement a set of policies that control authorization.

Authorization information, such as policies, attributes, identities and environmental parameters (e.g., time), are utilized and combined when making authorization decisions. Every entity may use policies to determine how a request or response should be handled. Policy defines rules for resource access. Many policies use the concepts of conditions and actions, which have to be evaluated with respect to the actual request, the requesting subject's identity and the attributes this subject holds. Policies may also be expressed in strings that are compared, and if one string (the request) is more specific than another (the policy), then the request is granted. Authorization attributes are statements about properties bound to an entity that implicitly or explicitly define the entities allowed actions on some resource. Attributes can be grouped into descriptive and privilege attributes. Descriptive attributes associate a characteristic with an entity, while privilege attributes define directly applicable access rights of an entity with respect to a resource. An administrative domain is a definition of the scope of authority. In a Grid environment, there are separate domains for identity, subject attributes, resource policy, and community policy authorities. In a simple Grid-use case, the subject is in one administrative domain, its home domain, and the resource is in another (the home domain of the resource). In more advanced scenarios, a community or VO domain is present. A VO domain can provide authorities that perform privilege management for all the members of a VO. Contractual relationships (often involving legal agreements) between the domains of the different subjects, authorities and resources are frequently necessary to enable the acceptance and issuing of authorizations.

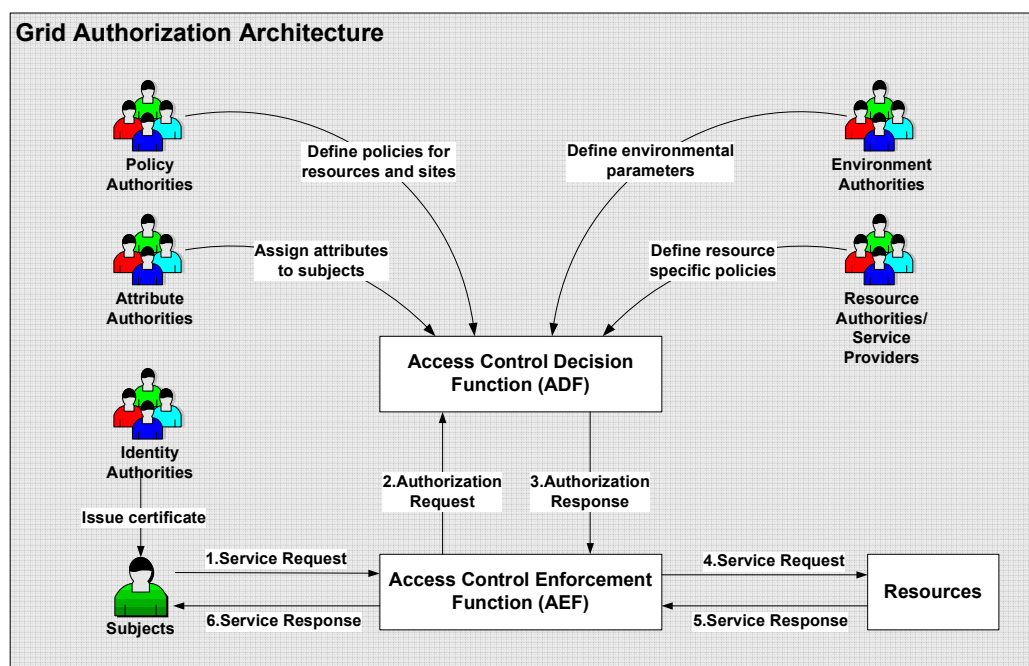
Authorization is commonly divided into three distinct processes. The first is defining an authorization policy at a high level by a person or organization. The second is implementing the high-level policy into a digital representation that can be interpreted by computers. The third is evaluating the digital representation of the policy by a process, which subsequently makes decisions to issue a specific authorization to a subject or takes a specific action. The component performing the evaluation of the executable policy by computing an authorization decision on behalf of the authorities is sometimes referred to as an ‘authorization server’. Typically, an authorization server may make or do (a combination of): an authorization decision that is, typically, the outcome of an evaluation of a policy; an authorization lookup of some entity’s rights that are represented in some form and returned; and delegation of an authorization decision to another authorization server.

RFC2904 defined the authorization sequences, which can be recognized as sequences between the three generic entities. These authorization sequences include: push sequence, pull sequence, agent sequence. With the push sequence, the subject first requests an authorization from an authority (e.g., via an authorization server). The authority may or may not honour the subject’s request. It then may issue and return some message or secured message (token or certificate) that acts as a proof of right (authorization assertion). The assertion may subsequently be used by the subject to request a specific service by contacting the resource. The resource will accept or reject the authorization assertion and will report this back to the requesting subject. With the pull sequence, the subject will contact the resource directly with a request. The resource then must contact its authorization authority. The authorization authority will perform an authorization decision and return a message that obtains the result of an authorization. The resource will subsequently grant or deny the service to the subject by returning a result message. Using the agent sequence, the subject will contact a higher-level agent with a request to obtain a service authorization. This agent will make an authorization decision, based the rules established by the authorization authority, and if successful, it will contact the resource to provision a certain state as to enable the service. These

three sequences are fundamental. They do not cover all possible authorization situations. Sometimes, they are combined together to perform authorization.

### 5.5.3.2. *Grid Authorization Architecture*

An authorization architecture consists of a set of entities and functional components that allow authorization decisions to be made and enforced, based on attributes, parameters and policies that define authorization conditions. Figure 5.3 (Lorch et al., 2004) provides an overview of an authorization system, based on the pull authorization sequence.



**Figure 5.3** *Grid Authorization Architecture*

The subject, resource and authority are three basic involved entities. There are two access control functions defined by ISO-101813 (ITU-T, 1996): Access Control Decision Function (ADF) and Access Control Enforcement Function (AEF). ADF is equivalent to the Policy Decision Point (PDP), defined in RFC2904, and AEF is equivalent to the Policy Enforcement Point (PEP), defined in RFC2904. ADF makes authorization decisions about a subject's access to a service. AEF mediates access to a resource or service. The ADF, AEF, subject and resources may be embedded inside one or more administrative domains in a variety of combinations.

There are three categories of information that may need to be passed between the subject, resource and various attribute authorities: attributes, policy and authorization queries and responses. Attributes, parameters and policies, issued by the corresponding authorities, are made available to the authorization servers. The authorization servers use this information to make authorization decisions upon request by the enforcement functions. An authorization request must be securely bound to a subject and the subject's service request. The authorization response must be securely bound to a request, and when required, also to the response originator.

There are a number of different paths for authorization attributes to get to the ADF (Farrell & Housley, 2002). The subject may get the attributes from the attribute authority, the authority could pass the attributes directly to the relevant ADF, or it could put them in an attribute repository. When the ADF needs an attribute to make an authorization decision, it may get it from the subject, either as part of the original request, or during a negotiation phase, or it may pull it from either a local repository or a repository associated with the attribute authority, the subject or VO. Attributes need to be reliably bound to the holding entities (holder/recipient) as well as the issuing authority. Attributes must be protected to provide for integrity, issuer authoritativeness and issuer non-repudiation. This can either be accomplished by enclosing them in a digitally signed container (e.g., via an X.509 Attribute Certificate or a signed SAML Attribute Assertion), or by issuing them over a secured channel between authenticated and trusted entities, and only storing attributes in trusted and secured repositories. Authorization requests and responses are similar to attributes in that it is necessary to provide for a secure binding.

Policies are typically stored in a repository or provisioned directly to the decision functions by the policy authorities. They may be distributed and stored in the domain of the policy issuer or in a common VO domain. It is imperative to securely establish the authority of the issuer and to protect the integrity of a policy during the transferring of policies. Once policy is written, it must be stored for use by an ADF. Evaluating policy is the heart of the authorization-decision process. The components responsible for expressing,

storing, retrieving and evaluating policy can be thought of as a policy subsystem. The policy expression is usually done by a policy language, which contains the vocabularies to express various policy artifacts. A variety of language primitives, such as XML, can be used as the basis for a policy language. Access policy for resources is written by policy authorities, which generally get their authority from the owner of the resource. Some authorization decisions may be indeterminate, because there are conditions in the policy that the ADF cannot evaluate, involving the current state of the resource. In this case, the conditions may be passed back to the AEF to evaluate, requiring the AEF to understand the policy language that is used to express these conditions. During policies' exchange, the policies must be bound securely to the issuer either by being contained within signed messages or by coming over a secured connection from a secure repository. The participating end points need to agree on a common policy expression language. Policy exchange may also include the exchange of metadata around policy, for example, creation time, policy validity, policy issuer and trust anchors. For policy processing, existing policy systems, including those based on artificial intelligence and neural net paradigms, can be effectively used so long as they understand the policy expression and exchange mechanisms.

#### **5.5.3.3. *Grid Authorization Framework***

Based on the concepts and architecture introduced above, a general Grid authorization framework is defined. This framework consists of several components, including trust management, privilege management, policy management, authorization context, authorization server and enforcement mechanism.

In general, authorization architecture, the assertions about policy and attributes are issued by different type of authorities. Trust management defines these authorities and specifies what they should be trusted to do. Policy and resource authorities both issue policy about resources, but the policy authority operates at a higher level and may issue access control policy for a whole site or VO. It is the root of trust, and will be responsible for defining the domain's trust relationships.

Attribute authorities assign attributes to subjects and may belong to the subject's domain or to a VO. In PKI-based systems, the authority is likely to be represented by a public/private key pair and present its assertions in signed documents or over a secured connection. At the base of a PKI system is the acceptance by all the participating entities of one or more CAs to verify identities. Once a VO or resource domain knows how to represent various authorities, it needs to define which ones are to be trusted and for what purposes. For example, the resource may want the sole say on what authorities it will trust, or it may accept the decisions of a VO policy authority. In some models, the user may provide a pointer to the attribute authority that defines his attributes, and the resource may accept it or not. The AEFs need to know which ADFs to trust for authorization decisions. Trust management also concerns the policy about who can create proxies, which have all or some of the rights of the delegating entity, and who can delegate rights to other entities.

Privilege management covers the definition, assignment, storage, presentation, delegation and revocation of both privilege and descriptive attributes. Privileges can be considered a type of attribute, where an attribute is any characteristic associated with a subject that either implicitly or explicitly defines the subject's allowed actions on some resource. Attributes that explicitly allow some access on a resource are called privilege attributes. Descriptive attributes, such as roles (for role-based access control), clearance level (for mandatory access control), or group membership, may be used by an authorization server interpreting an access policy to grant the user specific actions, and thus implicitly grant access rights. For the management of privilege and descriptive attributes, there are three distinct phases: granting the privilege, using the privilege, and removing the privilege. For privilege attributes, there are two primary actors: the authority granting/removing the privilege, and the subject requesting/using the privilege.

Policy is issued by policy authorities. The creation of policy frequently involves a human entity and is done in advance of the use of a resource. An ADF could query a policy authority in real time, but more typically, policy will be kept in some sort of repository. This could take the form of an access control list (ACL), a database or a collection of signed assertions. Policy management

addresses issues like who can create, modify and delete policy for each resource, how quickly policy can be revoked, and where does the ADF find the policy, i.e., who/what does it trust. For distributed policy management, an ADF needs to know whether it has found all the relevant policy for making an access decision. Additionally, policy management needs to address how to clearly display the current policy to the resource owner or to anyone trying to add to the policy.

The authorization context consists of those properties of the authorization request, which are neither provided via authorization attributes, nor included in authorization policies (specified by or for specific resources or sites), but which are relevant to the decisions made by the authorization server. This includes information about the time, location, transport, and authentication of the service request, and may include an indication of the quality and trustworthiness of this information.

Enforcement of access rights is done by limiting the operations performed on resources on behalf of a subject to those permitted by an authoritative entity. In the Grid context, enforcement functions can either receive the set of authorized operations as part of the service request (push scenario), or by querying an ADF (pull scenario). If the ADF and AEF are remote from each other, they can use authorization request/response protocols. If they are colocated, there are a number of programming interfaces available. Enforcement mechanisms can be characterized in two different groups: application-dependent mechanisms and application-independent mechanisms. Application-dependent enforcement mechanisms are often directly integrated in the application or service and perform enforcement functions before the application attempts to access underlying operating system resources. Application-independent enforcement mechanisms are separate from the service or application and take the approach of running the service in a very constrained execution environment. This permits the running of un-trusted services, supports code migration and the uploading of user-provided executables.

### 5.5.4. GSI security model for OGSA

GSI is the portion of the GT that provides the fundamental security services, according to the mechanisms discussed above. Globus Toolkit version 3 (GT3) and its accompanying GSI (GSI3) is the first implementation of the OGSA. Since GSI3, GSI implements the OGSA security mechanism to allow applications and users to operate in the Grid in a seamless and automated manner (Welch et al., 2003). As discussed in section 4.2., OGSA is built on Web services. It defines standard Web service interfaces and behaviours as well as other capabilities for addressing Grid-specific requirements. The combination of OGSA and Web services security specifications is used to implement OGSA security architecture (Siebenlist et al., 2002).

#### 5.5.4.1. *Web Services Security*

Web services security is an attempt within the Web services community to provide a standard XML vocabulary for defining protocols, message formats and policy languages for application to the entire range of security issues in distributed systems. The strategy for addressing security within a Web service environment requires a comprehensive model that supports, integrates and unifies several popular security models, mechanisms, and technologies (including both symmetric and public key technologies), in a way that enables a variety of systems to securely interoperate in a platform- and language-neutral manner.

In a Web-services environment, the following terms used for Web-services security will be used in this document.

- A **security token** is a representation of security-related information (e.g., X.509 certificate, Kerberos tickets and authenticators, mobile device security tokens from SIM cards, username, etc.).
- A **proof-of-possession (POP) token** is a security token that contains secret data that can be used to demonstrate authorized use of an associated security token. Typically, although not exclusively, the proof-of-possession information is encrypted with a key known only to the

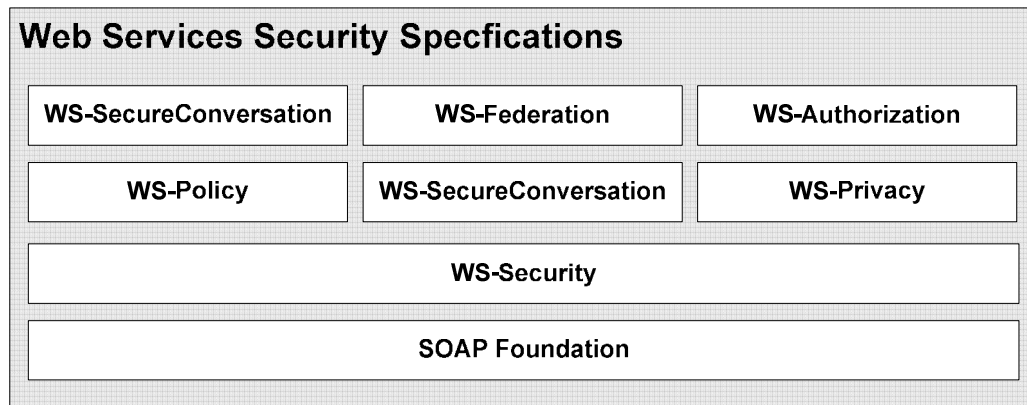
recipient of the POP token.

- A **claim** is a statement about a subject, either by the subject or by a relying party that associates the subject with the claim. Claims can be about keys potentially used to sign or encrypt messages. Claims can be statements the security token conveys. Claims may be used, for example, to assert the senders identity or an authorized role.
- A **signed security token** is a security token that contains a set of related claims (assertions) cryptographically endorsed by an issuer. Examples of signed security tokens include X.509 certificates and Kerberos tickets.
- As SOAP messages are sent from an initial requester to a service, they may be operated on by **intermediaries** that perform actions, such as routing the message or even modifying the message.
- An **actor** is an intermediary or endpoint, which is identified by a URI and which processes a SOAP message. Neither users nor client software (e.g., browsers) are actors.

Web services can be accessed by sending SOAP messages to service endpoints identified by URIs, requesting specific actions, and receiving SOAP message responses (including fault indications). Within this context, the broad goal of securing Web services is to provide facilities for securing the integrity and confidentiality of the messages and for ensuring that the service acts only on requests in messages that express the claims required by policies. TLS can be used to provide transport-level security for Web-services applications. TLS offers several security features, including authentication, data integrity and data confidentiality. However, TLS only enables point-to-point secure sessions. A comprehensive Web-services security solution needs the mechanism to provide end-to-end security.

The WS-security framework (IBM & Microsoft, 2002) illustrated in Figure 5.4 defines seven specifications of security functionality. The specifications build upon foundational technologies, such as SOAP, WSDL, XML Digital Signatures (W3C, 2002<sup>2</sup>), XML Encryption (W3C, 2002<sup>1</sup>), and TLS.

This set includes a message security model (WS-Security) along with a Web service endpoint policy (WS-Policy), a trust model (WS-Trust), and a privacy model (WS-Privacy). The follow-on specifications for secure conversations (WS-SecureConversation), federated trust (WS-Federation), and authorization (WS-Authorization) is built on these initial specifications to establish secure interoperable Web services across trust domains. Each of these specifications is summarized below:



*Figure 5.4 Web Services Security Specifications*

- **WS-Security** (OASIS, 2006<sup>10</sup>) defines how to attach and include security tokens within SOAP messages. It is an enhancement of SOAP messaging to provide quality of protection through message integrity and message confidentiality. It is designed to support multiple security-token formats. Message integrity is provided by leveraging XML Signature in conjunction with security tokens (which may contain or imply key data) to ensure that messages are transmitted without modifications. The integrity mechanisms are designed to support multiple signatures. The message confidentiality is provided by leveraging XML Encryption in conjunction with security tokens to keep portions of SOAP messages confidential. The encryption mechanisms are designed to support additional encryption technologies, processes, and operations by multiple actors. WS-Security also describes a mechanism for encoding binary security tokens. Specifically, the specification describes how to encode X.509 certificates and Kerberos tickets, as well as how to include opaque encrypted keys.
- **WS-Policy** (W3C, 2006<sup>4</sup>) describes how senders and receivers can specify

their requirements and capabilities. It is fully extensible, which means it has no limits on the types of requirements and capabilities that may be described. However, the specification identifies several basic service attributes, including privacy attributes, encoding formats, security token requirements, and supported algorithms. It also defines a generic SOAP policy format, which can support more than just security policies, and a mechanism for attaching service policies to SOAP messages.

- **WS-Trust** (OASIS, 2006<sup>13</sup>) describes the model for establishing both direct and brokered trust relationships (including third parties and intermediaries). This specification describes how existing direct trust relationships may be used as the basis for brokering trust through the creation of security-token issuance services, which build on WS-Security, to transfer the requisite security tokens in a manner that ensures the integrity and confidentiality of those tokens. It also describes how several existing trust mechanisms may be used in conjunction with this trust model
- **WS-Privacy** describes a model for how a privacy language may be embedded into WS-Policy descriptions, and how WS-Security may be used to associate privacy claims with a message. It also describes how WS-Trust mechanisms can be used to evaluate these privacy claims for both user preferences and organizational practice claims.
- **WS-SecureConversation** (OASIS, 2006<sup>3</sup>) describes how a Web service can authenticate requester messages, how requesters can authenticate services, and how to establish mutually authenticated security contexts. It describes how to establish session keys, derived keys, and per-message keys. Finally, it describes how a service can securely exchange context (collections of claims about security attributes and related data). In order to accomplish this, the specification describes, and builds upon, the concepts of security-token issuance and exchange mechanisms, defined in WS-Security and WS-Trust. WS-SecureConversation is designed to operate at the SOAP-message layer so that the messages may traverse a variety of transports and intermediaries.
- **WS-Federation** (Bajaj et al., 2003) defines how to construct federated trust

scenarios using the WS-Security, WS-Policy, WS-Trust, and WS-SecureConversation specifications. For example, it will describe how to federate Kerberos and PKI infrastructures. A trust policy is introduced to indicate, constrain and identify the type of trust that is being brokered. This specification will also define mechanisms for managing the trust relationships.

- **WS-Authorization** describes how access policies for a Web service are specified and managed. In particular, it describes how claims may be specified within security tokens, and how these claims will be interpreted at the endpoint. It is designed to be flexible and extensible with respect to both authorization format and authorization language.

#### **5.5.4.2. *OGSA Security Model***

OGSA security services facilitate the enforcement of the security-related policy within a VO. The nature of the Grid environment requires that the OGSA security-architecture components must support, integrate, and unify popular security models, mechanisms, protocols, platforms, and technologies in a way that enables a variety of systems to interoperate securely. The components must be able to support integrating with existing security architectures and models across platforms and hosting environments. This means that the architecture must be implementation-agnostic, extensible, and integratable. These characteristics mean that the OGSA architecture can be instantiated in terms of any existing security mechanisms (e.g., Kerberos and PKI); incorporate new security services as they become available; and integrate with existing security services. Also, services that traverse multiple domains and hosting environments need to be able to interact with each other, thus introducing the need for interoperability at multiple levels: protocol, policies and identity. In addition, certain situations can make it impossible to establish trust relationships among sites prior to application execution. Given that the participating domains may have different security infrastructures (e.g., Kerberos or PKI), it is necessary to realize the required trust relationships through some form of federation among the security mechanisms.

The security model described in OGSA (Foster et al., 2006) defines the security services as entities with interaction patterns that facilitate the administration, expression, publishing, discovery, communication, verification, enforcement and reconciliation of the security policy. In other words, the security policy enforcement is the ultimate goal, and the security services are designed and deployed to support that goal. This model identified a number of these entities, interaction mechanisms and contexts, and discussed some of their attributes and common relationships.

Based on this model, the functional capabilities and corresponding security services are identified in the OGSA specification:

- **Authentication.** Authentication is concerned with verifying proof of an asserted identity. This functionality is part of the credential validation and trust Services. One example is the evaluation of a user-id and password combination, in which a service requestor supplies the appropriate password for an asserted user-id.
- **Identity mapping.** The trust, attribute and bridge/translation services provide the capability of transforming an identity that exists in one identity domain into an identity within another identity domain. Identity mapping service, via policy, maps the service requestor's identity to an identity that has meaning, for instance, to the hosting environment's local platform registry. The identity-mapping service is not concerned with the authentication of the service requestor; rather it is strictly a policy-driven name-mapping service
- **Authorization.** The authorization service is concerned with resolving a policy-based access-control decision. The authorization service consumes as input, a credential that embodies the identity of an authenticated service requestor and, for the resource that the service requestor requests, resolves, based on policy, whether or not the service requestor is authorized to access the resource. It is expected that the hosting environment for OGSA-compliant services will provide access-control functions, and it is appropriate to further expose an abstract authorization service, depending on the granularity of the access-control policy that is being enforced.

- **Credential conversion.** The trust, attribute and bridge/translation services provide credential conversion from one type of credential to another type or form of credential. This may include such tasks as reconciling group membership, privileges, attributes and assertions associated with entities (service requestors and service providers). For example, the credential conversion service may convert a Kerberos credential to a form that is required by the authorization service. The policy-driven credential-conversion service facilitates the interoperability of differing credential types, which may be consumed by services. It is expected that the credential-conversion service would use the identity mapping service.
- **Audit and secure logging.** The audit service is responsible for producing records that track security-relevant events. The resulting audit records may be reduced and examined so as to determine whether the desired security policy is being enforced. Auditing and subsequent reduction tooling are used by the security administrators within a VO to determine the VO's adherence to the stated access-control and authentication policies.
- **Privacy.** The privacy service is primarily concerned with the policy-driven classification of personally identifiable information (PII). Service providers and service requestors may store personally identifiable information using the privacy service.

#### **5.5.4.3. *GSI Security Model for OGSA***

OGSA introduces both new opportunities and new challenges for Grid security. Emerging Web-services security specifications address the expression of Web-service security policy (WS-Policy, XACML), standard formats for security token exchange (WS-Security, SAML), and standard methods for authentication and establishment of security contexts and trust relationships (WS-SecureConversation, WS-Trust). These specifications can be exploited and extended to create standard, interoperable methods for addressing Grid-security issues.

GT3 provides the first implementation of OGSA mechanisms (Welch et al., 2003). Since GT3, the GSI security model intends to allow applications and users to operate on the Grid in as seamless and automated a manner as possible. It means the security mechanisms should not have to be instantiated in an application, but instead should be supplied by the surrounding Grid infrastructure, allowing the infrastructure to adapt on behalf of the application to meet the application's requirements. It allows for if the application should need to deal with only application-specific policy. GSI uses the following powerful features of OGSA and Web-services security to work toward this goal:

- OGSA security model casts security functions as OGSA services. As introduced in previous section, OGSA defined numerous services, such as authorization, credential conversion services, delegation, etc. This mechanism allows Grid applications to avoid embedding security mechanisms statically in order to adapt to changing requirements.
- The OGSA security model uses a sophisticated container-based hosting environment (such as J2ME and .Net) to handle security for applications and to allow security to adapt without having to change the application. In order to establish trust, two entities need to be able to find a common set of security mechanisms that both understand. The use of hosting environments and OGSA security services enables OGSA applications and services to adapt dynamically and use different security mechanisms.
- A Web service can publish its security policy, along with its interface specification, as part of a WSDL document. This is defined by WS-Policy specification and its related specifications. A published policy can express requirements for mechanisms, acceptable trust roots, token formats, and other security parameters. When an application wishes to interact with the service, it can examine the published policy and gather the needed credentials and functionality by contacting appropriate OGSA security services.
- The WS-Security, WS-SecureConversation, and WS-Trust specifications contain conventions and formats for the communication of various mechanism-specific tokens (e.g., Kerberos tickets and X.509 certificates)

inside SOAP envelopes. The SOAP enveloping standardizes the protocol for security mechanisms and allows mechanisms to be independent of any application protocol. Hosting environments can recognize security-related messages and route them to an appropriate service for handling, and entities in the network can recognize whether and how an interaction is secured.

## **5.6. Security Solutions**

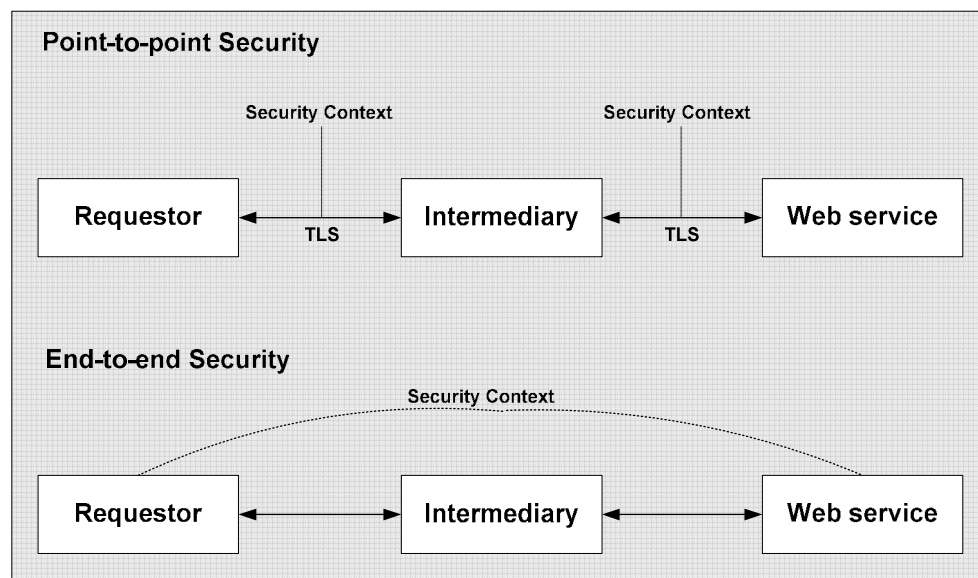
From version 1 in 1998 to the 2 release in 2002 and now the 4 release, GSI has been developing rapidly. In GT1, GSI mainly provided message protection and authentication. In GT2, GSI introduced X.509 Proxy Certificates to support dynamic creation of computing entities and provided CAS to implement access control in dynamically created overlaid trust domains. In GT3, the Grid technology worked with the emerging Web services technology. Security functionalities of GSI3 are defined as OGSA services. In GSI4, which is the latest GSI version, additional Web-services security specifications are implemented.

The security solution for the Grid-enabled, distributed data warehouse system is provided by GSI4. GSI4 Web-services components can be divided into four distinct functions (Welch, 2005): message protection, authentication, delegation and authorization. GSI4 contains both Web-services components and pre-Web-services components. MyProxy is a pre-Web-services component that is used to provide an online credentials repository, and for delegating X.509 Proxy Certificates to the Grid portal. There exist several cross-domain authorization systems for use on the Grid. Authorization in the GT4 is by default, based on ACLs located at each resource. The ACLs specify the identifiers of the users allowed to access the resource. Also, higher-level services (such as CAS) that provide richer authorization policies exist as optional configurations. For the proposed Grid system, the authorization solution is provided by Shibboleth as well as GridShib, which integrate X.509 certificates with SAML to provide cross-domain, attributed-based authorization. This section finally will introduce a security model combining all identified

components to address basic security issues in the Grid-enabled, distributed data warehouse system.

### 5.6.1. Message protection

In an OGSA-based Grid system, resources are represented as services, specifically, as Grid services, which are Web services with well-defined interfaces. Web services use SOAP message to communicate between services. SOAP provides a means of messaging, using XML envelopes to encapsulate payloads, with HTTP, the most commonly used underlying protocol. In GSI4, message protection can be provided either by transporting SOAP messages over TLS, known as transport-level security, or by signing and/or encrypting portions of the SOAP message using the WS-Security standard, known as message-level security. TLS only provides point-to-point secure to ensure privacy and data integrity. GSI4 implements the WS-Security standard and the WS-SecureConversation specification to provide end-to-end message protection for SOAP messages. Figure 5.5 shows the differences between point-to-point security and end-to-end security views.



*Figure 5.5 Point-to-Point and End-to-End Security*

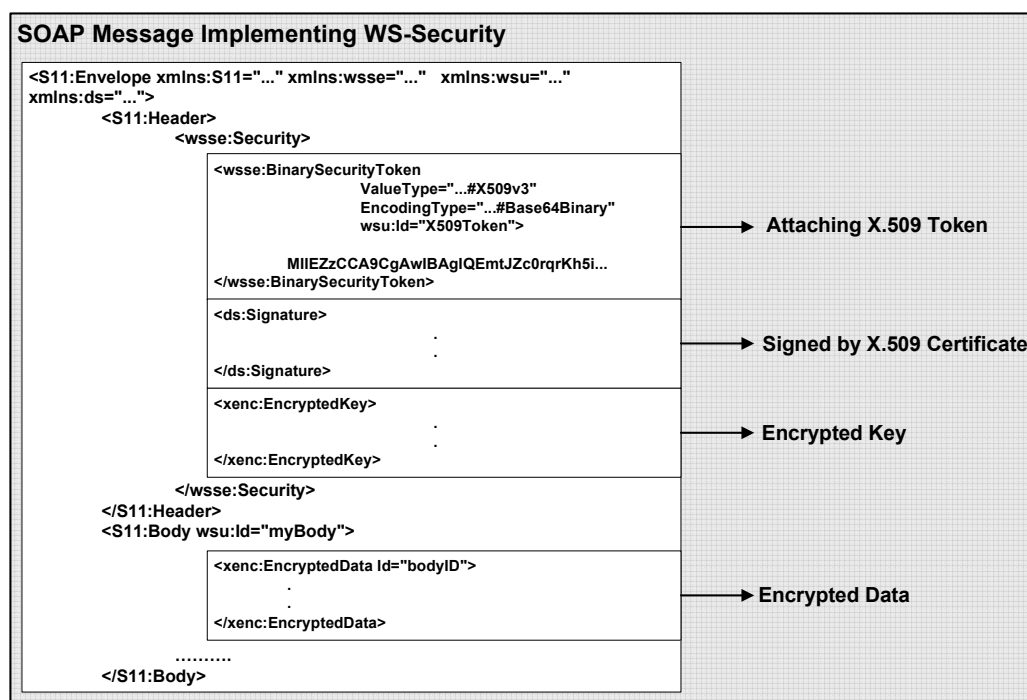
Transport-level security entails SOAP messages conveyed over a network connection protected by TLS. It is normally used in conjunction with X.509 credentials for authentication. The WS-Security standard and the

WS-SecureConversation specification are implemented to provide message protection for SOAP messages. The SOAP specification allows for the abstraction of the application-specific portion of the payload from any security (e.g., digital signature, integrity protection, or encryption) applied to that payload. The WS-Security standard defines a framework for applying security in individual SOAP messages. GSI4 uses these mechanisms to provide security on a per-message basis. WS-SecureConversation allows for an initial exchange of messages to establish a security context, which can then be used to protect subsequent messages in a manner that requires less computational overhead. Both WS-Security and WS-SecureConversation are intentionally neutral to the specific types of credentials used to implement this security.

WS-Security defines how to attach and include security tokens within SOAP messages. It is designed to support multiple security-token formats (OASIS, 2006<sup>9</sup>). Specifically, for the Grid-enabled, distributed data warehouse system, WS-Security attaching a X.509 certificate token (OASIS, 2006<sup>11</sup>) will be used to provide single-message integrity and confidentiality. Figure 5.6 shows a SOAP message structure, which is digitally signed and encrypted by using a X.509 certificate. This example is a SOAP message signed before encryption. Likewise, if a producer wishes to sign a message after encryption, they should first prepend the encryption element (`<xenc:EncryptedKey>`) to the `<wsse:Security>` header, and then prepend the signature element (`<ds:Signature>`).

WS-Security provides the means to secure only a single SOAP message. However, a client and a Web service interact by exchanging series of messages grouped in sessions. While, in principle, WS-Security could secure each separate message in a session, this can become inefficient with regard to bandwidth and processing capacity if X.509 certificates are used in each message. In addition, it is also desirable to guarantee integrity of a whole session, and not just a single message. WS-SecureConversation describes the protocol that allows two or more endpoints exchange long-lived credentials only once, at the beginning of the conversation. If the credentials are accepted, the requestor gets a Security Context Token (SCT) that acts as a lightweight

credential with a reference to a shared secret (symmetric key), only known by the participating nodes and valid for a predefined time period (e.g., duration of a user session or message sequence). Subsequent messages in the conversation only have to carry this token. The SCT, introduced by WS-SecureConversation, points to a secret key, shared between the participants. Its initial key data can be used to repeatedly derive new session keys during a conversation using fast crypto algorithms. An SCT can be established in different ways (OASIS, 2006<sup>3</sup>). For the Grid-enabled, distributed data warehouse system, the X.509 certificates are used to establish a session key.



*Figure 5.6 SOAP Message Implementing WS-Security*

## 5.6.2. Authentication, delegation and SSO

GSI4 uses X.509 end-entity certificates (EECs) to identify persistent entities, such as users and services. X.509 EECs provide each entity with a unique identifier (i.e., a DN) and a method to assert that identifier to another party, through the use of an asymmetric key pair, bound to the identifier by the certificate. Authentication with X.509 credentials can be accomplished either via TLS, in the case of transport-level security, or via signature, as specified by WS-Security, in the case of message-level security.

GSI4 supports delegation and SSO through the use of standard X.509 Proxy Certificates. Proxy Certificates allow bearers of X.509 EECs to delegate their privileges temporarily to another entity. For the purposes of authentication and authorization, GSI4 treats EECs and Proxy Certificates equivalently. GT4 supports a delegation service that provides an interface to allow clients to delegate (and renew) X.509 Proxy Certificates to a service. The interface to this service is based on the WS-Trust specification.

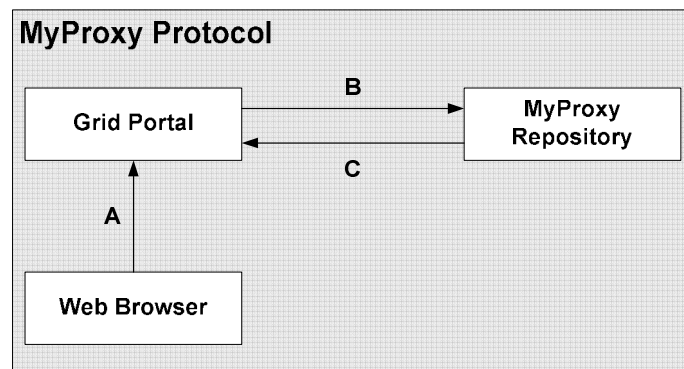
WS-Trust specification defines extension to WS-Security. It provides methods for issuing, renewing, and validating security tokens. For example, it defines Security Token Service (STS), which can be used to create an SCT. It also provides ways to establish, assess the presence of, and broker trust relationships. As defined in WS-Trust, there are a few mechanisms where existing keys are transferred to other parties. Delegation is one of these key transfer mechanisms (OASIS, 2006<sup>13</sup>). It allows one-party transfers the right to use a key without actually transferring the key. For example, a custom token is issued from party A to party B. The token indicates that B (specifically B's key) has the right to submit purchase orders. The token is signed using a secret key known to the target service T and party A (the key used to ultimately authorize the requests that B makes to T), and a new session key that is encrypted for T. A proof-of-possession token is included that contains the session key encrypted for B. As a result, B is effectively using A's key, but does not actually know the key.

### **5.6.3. MyProxy protocol**

MyProxy is an online credentials' repository for Grid systems (Novotny, Tuecke & Welch, 2001) (Lorch, Basney & Kafura, 2004). It is a pre-Web Services component in GSI4. In the Grid-enabled, distributed data warehouse system, a Grid portal, combining a Web server and Grid-enabled software, provides users an interface to access all applications using standard Web browsers. Grid portal requires that the user delegates to the server the right for that server to act on the user's behalf, in order to initiate and monitor operations for that user on Grid resources. GSI supports such delegation, but the standard

Web-security protocols do not. MyProxy bridges this incompatibility between Web- and Grid-security protocols, thus enabling Grid portal to use GSI-protected resources in a secure and scalable manner. It allows long-lived keys to be secured on the remote server, while allowing convenient access to short-lived, proxy credentials as needed.

The MyProxy credentials repository system consists of a repository server and a set of client tools for delegating to and retrieving credentials from the repository. In order to meet the goals of the MyProxy approach, there are two basic steps for using the repository: delegation of proxy credentials to the repository and retrieving the credentials from the repository (as illustrated in Figure 5.7).



**Figure 5.7 MyProxy Protocol**

- a) A user starts by using the myproxy-init client program along with its permanent credentials to contact the repository and delegate a set of proxy credentials to the server, along with authentication information and retrieval restrictions. Authentication information in this process consists of a user identity (ID) and a pass phrase. It is used to authenticate any retrieval operations. This user ID is different from the user's DN; it is actually hand-typed by the user at later times. The user ID identifies the account storing the proxy credentials in repository server. Both can be tested by the repository to ensure they comply with any local policy (for example, the pass phrase must meet a certain length). The only available retrieval restriction that can be placed on delegations by the repository is the maximum lifetime of the proxy credentials. These restrictions are intended to be expanded in future versions.

- b) A user, or service acting on behalf of the user, uses the myproxy-get-delegation client program to contact the repository server and request a delegation of the user's credentials. During this process, the user must provide the ID and pass phrase for verification. After verifying this authentication information and checking the restrictions that the user presented with the delegation, the repository will delegate proxy credentials back to the user or service.
- c) The credentials delegated to the repository normally have a lifetime of a week. This lifetime can be changed to any length of time desired. The credentials delegated to the repository can be destroyed at any time by using the myproxy-destroy client program.

The first step to using MyProxy in a Grid portal is to delegate proxy credentials to the repository by using the myproxy-init client program. Then the user may connect to the Grid portal by using a Web browser and provide the authentication information (user ID and pass phrase) through a Web form at a different time and place. The Grid portal then uses myproxy-get-delegation program to connect to the MyProxy repository and authenticates itself using its own Grid credentials. The user's authentication information (user ID and pass phrase) is also transferred to repository server at the same time for requesting a proxy credential for the user. The repository would delegate a proxy credential for the user back to the portal after all necessary verification (portal's Grid credentials, user's authentication information). The Grid portal then can securely access the Grid resources by using standard Grid applications. The operation of logging out of the Grid portal deletes the user's delegated credentials on the portal; otherwise, the credentials will expire when the lifetime lapses.

MyProxy has been extended to better integrate with existing site infrastructure, and to make it easier for users to bootstrap their X.509 security context. It introduces new developments, including management of trust roots, standards-based integration with site authentication and the ability to act as a CA. These new features enhance the MyProxy usage in the proposed Grid system. They are introduced as follows:

- A user's X.509 security context includes an end entity or proxy credential, one or more trusted CA certificates, and certificate revocation information in the form of CRLs or online certificate status protocol (OCSP) (Myers, Ankney, Malpani, Galperin & Adams, 1999) responses. MyProxy Logon application is used to obtain a user's complete security context from the MyProxy service. The MyProxy administrator maintains a set of trusted CA certificates and configures the server to periodically fetch fresh CRLs. MyProxy Logon fetches the configured CA certificates and CRLs in addition to the user's end entity or proxy certificate and installs them in the local user's environment.
- The MyProxy service can be configured to allow users to logon with existing site credentials, using Pluggable Authentication Modules (PAMs) and/or the Simple Authentication and Security Layer (SASL). Through these mechanisms, users are not required to remember another username and password for the MyProxy service.
- For users that do not already have X.509 credentials to store in the MyProxy repository, the administrator can configure MyProxy to act as an online CA to issue certificates in realtime-based on-site authentication. The administrator must provide a mapping of authenticated usernames to certificate subjects, either in a configuration file or through LDAP (Lightweight Directory Access Protocol). The user authenticates via MyProxy Logon to the MyProxy service, and MyProxy issues a certificate to the user with the subject provided in the mapping file. MyProxy CA provides a lightweight mechanism for sites to distribute X.509 credentials.

#### **5.6.4. Authorization**

Authentication addresses the question, "Is the user who he claims to be?", which is typically a permanent attribute. Authorization addresses the question, "What is the user allowed to do?", which can vary over time. GSI is based on a PKI with CAs and X.509 certificates. PKI provides credentials that can help to solve the first question. Authorization, which is commonly based on a set of user attributes, identities, policies and environmental parameters to make

authorization decisions, however, is not usually solved by PKI: naturally, a X.509 certificate can contain a set of user attributes that are used by the application for determining the user's authorization.

There are a number of authorization systems currently available for use on the Grid as well as in other areas of computing, such as Akenti (Thompson et al., 1999), CAS (Pearlman, Welch, Foster, Kesselman & Tuecke, 2002) (Pearlman, Welch, Foster, Kesselman & Tuecke, 2003), PERMIS (Chadwick & Otenko, 2002), VOMS ("VOMS Architecture v1.1", 2002), Cardea (Lepro, 2003), PRIMA (Lorch & Kafura, 2002) (Lorch et al., 2003), and Shibboleth (<http://shibboleth.internet2.edu/>). Some of these systems are normally used in a push model - they act as services and issue these authorization decisions in the form of authorization assertions that are conveyed to the target resource by the requestor. Others are used in a pull model - they are normally linked with an application or service and act as a policy decision-maker for that application. Akenti, PERMIS and Shibboleth use user attributes to make authorization decisions; VOMS provides user attributes that can be used for authorization.

For the Grid-enabled, distributed data warehouse system, GSI4 provides the X.509 certificates to identify the Grid clients. An X.509 certificate is treated as input of authorization system, provided by Shibboleth, as well as GridShib plug-ins. This approach can integrate X.509 certificates with SAML to provide cross-domain attributed-based authorization. GT4 introduces an SAML/XACML authorization framework that supports multiple security policies (Lang, Foster, Siebenlist, Ananthakrishnan & Freeman, 2006). A Grid system consists of multiple administrative domains, which have their own security policies, such as grid-mapfile, ACL, CAS, SAML authorization decision assertions, and XACML policy statements. This framework tends to be flexible, so that it can be changed easily for different application environments. This framework allows the authorization system for the proposed system to integrate into GT4 authorization framework as an authorization service, so it enables the proposed Grid system to seamlessly accommodate new participants, which use different authorization systems.

#### **5.6.4.1. SAML**

According to OASIS (OASIS, 2005<sup>1</sup>), the SAML standard defines an XML-based framework for describing and exchanging security information between on-line business partners. This security information is expressed in the form of portable SAML assertions that applications, working across security domain boundaries can trust. The OASIS SAML standard defines precise syntax and rules for requesting, creating, communicating, and using these SAML assertions. It provides support for full federation and mapping of identifiers, session management, greater interoperability for attribute exchange and other features.

SAML is defined in terms of assertions, protocols, bindings and profiles. An assertion is a package of information that supplies one or more statements made by a SAML authority. There are three different kinds of assertions: authentication statements, attributes statements, and authorization-decision statements. Authentication statement is typically generated by a SAML authority, called an identity provider, which is in charge of authenticating users and keeping track of other information about them. Attributes statements contain specific identifying attributes about the subject (for example, that user “John Doe” has “Gold” card status). Authorization-decision statements define something that the subject is entitled to do (for example, whether “John Doe” is permitted to buy a specified item). SAML defines a number of generalized request/response protocols, including Authentication Request, Single Logout Assertion Query and Request, Artifact Resolution, Name Identifier Management, and Name Identifier Mapping Protocols. Assertion Query and Request Protocol defines a set of queries by which SAML assertions may be obtained. Name Identifier Mapping Protocol provides a mechanism to programmatically map one SAML name identifier into another, subject to appropriate policy controls. It permits, for example, one service provider to request from an identity provider an identifier for a user that the service provider can use at another service provider in an application integration scenario. SAML bindings detail exactly how the various SAML protocol messages can be carried over underlying transport protocols. For instance, the

SAML SOAP binding defines how SAML protocol messages can be communicated within SOAP messages, whilst the HTTP redirect binding defines how to pass protocol messages through HTTP redirection. A profile of SAML defines constraints and/or extensions in support of the usage of SAML for a particular application.

#### **5.6.4.2. *Shibboleth***

Shibboleth is an attribute-based authorization system that asserts attributes about a user between organizations (Erdos & Cantor, 2001). The current implementation of the specification is Shibboleth 1.3 (released July 2005). More precisely, Shibboleth asserts attributes between the user's home organization and organization's hosting resources that may be accessible to the user. Shibboleth can be conceptually regarded as comprising three components: Handle Service, Attribute Authority (AA) and Target Resource. The Handle Service authenticates users in conjunction with a local organizational authentication service and issues to the user a handle token. When a user requests access to a target resource, he presents his handle token. The resource then presents the user's handle token to the attribute authority and requests attributes regarding the user. The Shibboleth AA retrieves attributes from an organizational authority and provides them in the form of SAML assertions. The target resource includes Shibboleth-specific code to determine the user's home organization and hence, which Shibboleth attribute authority should be contacted for the user to retrieve attributes regarding the user, and to make authorization decisions, based on those attributes. Shibboleth is based in large part on SAML. SAML defines two functional components: an Identity Provider (IdP) and a Service Provider (SP). The IdP creates, maintains, and manages user identity, while the SP controls access to services and resources. An IdP produces and issues SAML assertions to SPs upon request. An SP consumes SAML assertions obtained from IdPs for the purpose of making access-control decisions. Based on Shibboleth Attribute Exchange Profile (Cantor et al., 2005), on the IdP side, a Shibboleth AA produces and issues attribute assertions, while a subcomponent of the SP, called an Attribute Requester, consumes these assertions.

#### **5.6.4.3. *GridShib***

The GSI4 uses X.509 certificates and X.509 Proxy Certificates for authentication. In brief, these certificates allow a user to assert a globally unique identifier (i.e., a DN from the X.509 certificate). SAML can use its attribute queries and assertions to support distributed authorization in support of X.509-based authentication (OASIS, 2005<sup>4</sup>). GridShib is a software product that allows for interoperability between the GT and Shibboleth (Welch, Barton, Keahey & Siebenlist, 2005) (Barton et al., 2006). The complete software package consists of two plug-ins: one for the GT4 and another for Shibboleth. The main purpose of GT4 plug-in is to obtain attributes about a requesting user from a Shibboleth AA and make an access control decision, based on those attributes. GridShib for Shibboleth is a name mapping plug-in for a Shibboleth 1.3 identity provider. Its main purpose is to allow the servicing of attribute queries from Grid SPs, based on the user's X.509 subject DN. With both plug-ins installed and configured, a GT Grid service provider may securely request user attributes from a Shibboleth IdP.

GridShib focuses on using Shibboleth as the AA. It assumes the users have a valid X.509 certificate, containing at least the name of users' institution (IdP). The X.509 certificate is offered to the application, verified, and the IdP information is extracted. Next, the application contacts the IdP, supplies the certificate's ID, and receives in return a SAML assertion containing the user's attributes. The GridShib Profile is an extension of the Shibboleth Attribute Exchange Profile. The primary difference is the use of X.500 DNs to identify principals. The detailed GridShib protocol flow will be discussed with all identified security components in section 5.6.5.

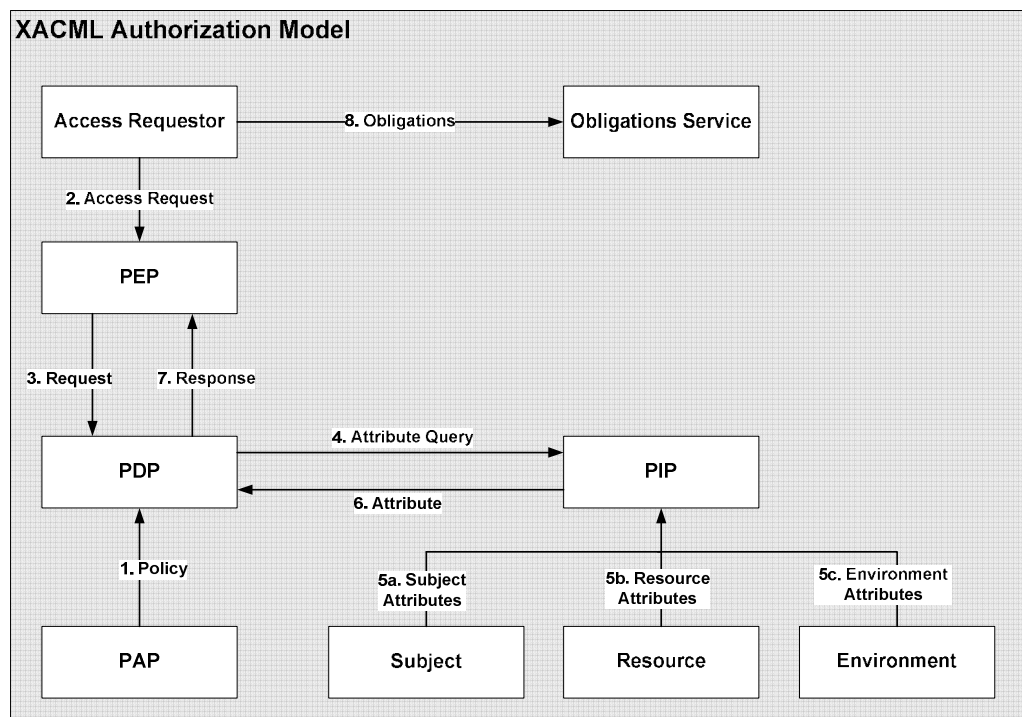
#### **5.6.4.4. *XACML Authorization Framework***

The GT4 authorization framework uses the XACML model. This section gives a brief overview of XACML authorization model.

According to OASIS (OASIS, 2005<sup>2</sup>), XACML defines a core schema and corresponding namespace for the expression of authorization policies in XML

against objects that are themselves identified in XML, and enables the use of arbitrary attributes in policies, role-based access control, security labels, time/date-based policies, indexable policies, “deny” policies, and dynamic policies — all without requiring changes to the applications that use XACML.

Figure 5.8 gives an overview of the XACML authorization model. It mainly contains PEP (Policy Enforcement Point), PDP (Policy Decision Point), PIP (Policy Information Point), and PAP (Policy Administration Point). The PEP intercepts the access requests from users and sends the requests to the PDP. The PDP makes access decisions according to the security policy or policy set written by PAP and, using attributes of the subjects, the resource, and the environment obtained by querying the PIP. The access decision given by the PDP is sent to the PEP. The PEP fulfills the obligations and either permits or denies the access request, according to the decision of PDP.



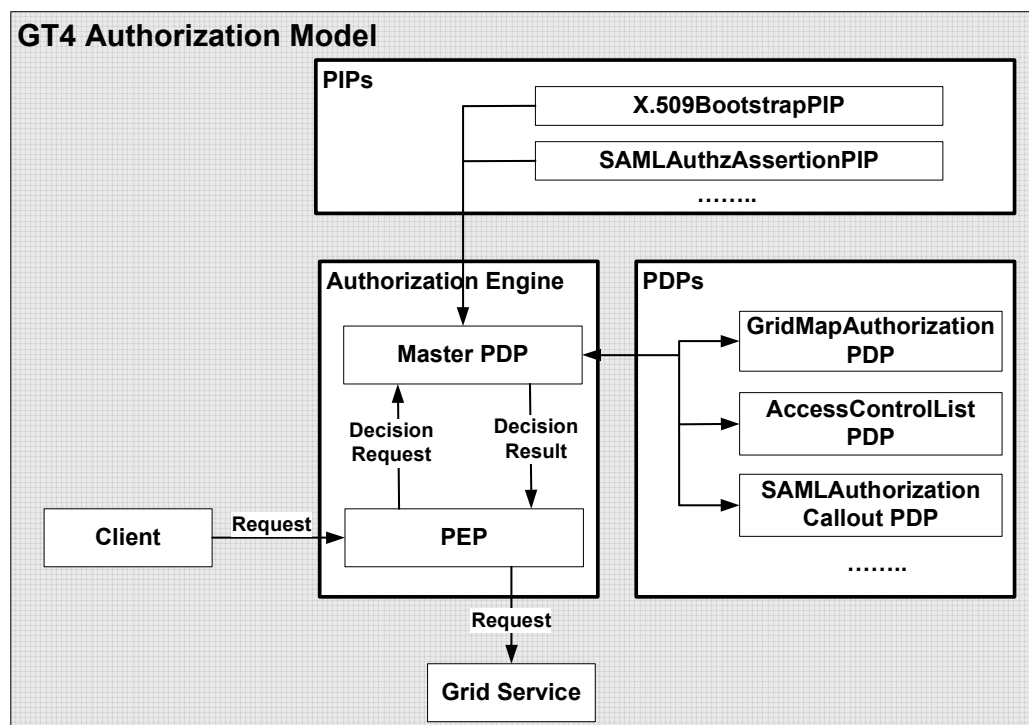
**Figure 5.8 XACML Authorization Model**

#### **5.6.4.5. GT4 SAML/XACML Authorization Framework**

The GT4 authorization framework implements SAML and uses the XACML model. As shown in Figure 5.9, it is composed of a PEP, PDPs, and PIPs.

For each existing authorization policy, the framework constructs a PDP for evaluating that kind of policy. The Master PDP is responsible for coordinating the PDPs to render a final decision. The Master PDP and the PEP are collectively called the authorization engine. The framework provides different kind of PIPs. A subset of PIP, referred to as Bootstrap PIPs, collect information only about the request, such as the peer subject, the requested action, and the resource. An example of one such PIP is the X509BootstrapPIP, which extracts the subject DN of the peer from the X509 certificate.

When a request of the Grid resource comes, the PEP intercepts it and sends a decision request to the Master PDP. The Master PDP collects information needed by calling the Bootstrap PIPs and other PIPs and then invokes the corresponding PDPs with the request and the information collected. The PIPs and the PDPs used are all specified in the security configuration file. When the Master PDP receives the decisions returned by each PDP, it combines the decisions, using a policy combination algorithm, such as deny override or permit override, to render a final decision and returns the decision to the PEP. The PEP then executes the decision, either denying or permitting the request.

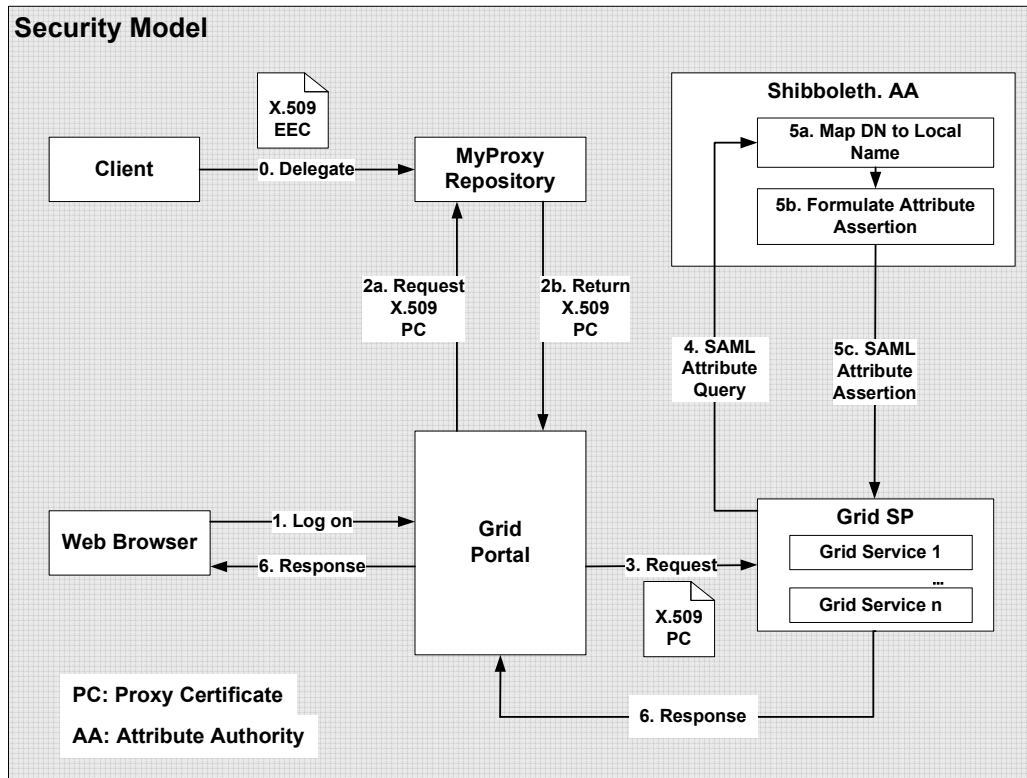


*Figure 5.9 GT4 Authorization Framework*

The PDP is the core of the authorization framework. In order to make the framework support different kind of policies and be scalable, it is necessary to encapsulate the policy into an independent PDP. In Grid systems, there are several frequently used simple authorization policies or mechanisms such as the provided PDPs that implement these existing policies, such as the AccessControlList PDP and the GridMapAuthorizaion PDP. There are also some authorization systems developed by others that can be used in a Grid system, such as Shibboleth, VOMS and PERMIS. Therefore, a SAMLAuthorizationCallout PDP for integrating those authorization systems through the SAML assertions is established.

### **5.6.5. Putting it all together**

The Grid-enabled, distributed data warehouse system requires security functions, including authentication, SSO, delegation and authorization. SOAP messages are conveyed over a network connection protected by TLS in order to ensure transport-level security. Web services security standards and specifications are used to provide end-to-end SOAP message protection. Authentication with X.509 credentials can be accomplished either via TLS, in the case of transport-level security, or via signature, as specified by WS-Security, in the case of message-level security. Delegation and SSO can be achieved through the use of standard X.509 Proxy Certificates. X.509 EEC and X.509 Proxy Certificates represent long-term and short-term credentials respectively; they are treated equivalently for the purpose of authentication. MyProxy provides an online credential repository for X.509 proxy credentials encrypted by user-chosen pass phrases. Shibboleth and GridShib provide authorization. More specifically, a Shibboleth AA of IdP is responsible for name mapping, and GridShib plug-ins are responsible for formulating the attribute assertions.



*Figure 5.10 Security Model*

Figure 5.10 shows the security model that combines all the necessary components. Before explaining the processes defined in this model, the following assumptions are made:

- The Grid client and the Grid SP each possess an X.509 credential.
- The Grid client has an account with a Shibboleth IdP.
- The IdP and the Grid SP each have been assigned a globally unique identifier, called a provider-ID.
- The Grid SP and the IdP rely on the same metadata format and exchange this metadata out-of-band. Specifically, both the IdP and the Grid SP rely on SAML 2.0 metadata (OASIS, 2005<sup>3</sup>) for their trust configuration (i.e., the certificates and public keys of the other entity).

The work flows depicted in Figure 5.10 are described as follows:

**Step 0).** Delegate a X.509 Proxy Certificate to the MyProxy credentials repository server. The client that stores the user's X.509 EEC (long term user's credentials) sends a "Put" request (Basney, 2005) to the repository server, along with a user ID, pass phrase and lifetime through the use of myproxy-init

client program. The repository server accepts the request and generates a new public/private key pair and then sends a certificate request, containing the repository's public key, to the client. The client then sends a X.509 Proxy Certificate containing the public key from the certificate request, signed by its X.509 EEC's private key, followed by the corresponding certificate chain, back to the repository server. The repository then stores this X.509 Proxy Certificate for later retrieval from the Grid portal. The user ID and pass phrase specify the account for storing the proxy certificate and are used to authenticate its retrieval operation. The stored X.509 expires at the lifetime. This process is authenticated via TLS. This is the initialization step.

**Step 1).** Users can log on the Grid portal at a different time and place by using a standard Web browser.

**Step 2).** The Grid portal sends a "Get" request (Basney, 2005), along with the user ID and pass phrase generated in the initialization step, to the MyProxy repository server to retrieve the stored user's X.509 Proxy Certificate through the use of the "myproxy-get-delegation" client program. If the repository's response indicates success, the Grid portal generates a new public/private key pair and then sends a certificate request, containing the Grid portal's public key, to the repository server. The repository server then sends a X.509 Proxy Certificate, containing the public key from the certificate request, signed by the private key of the stored user's X.509 Proxy Certificate, followed by the corresponding certificate chain, back to the Grid portal. This process is authenticated via TLS and also the user ID and pass phrase. It enables the repository to delegate proxy credentials for the user back to the Grid portal, and then the Grid portal can act on the user's behalf to securely access the Grid resources by using standard Grid applications.

**Step 3).** The Grid portal holding the user's X.509 Proxy Certificate can request the Grid service (or resources) on the user's behalf. It authenticates, using the retrieved X.509 Proxy Certificate to the Grid SP. The Grid SP authenticates the request and extracts the client's DN from the credentials.

**Step 4).** The Grid SP formulates a SAML attribute query, whose NameIdentifier element is the DN extracted from the X.509 Proxy Certificate. The Grid SP uses its own X.509 EEC to authenticate to the AA.

**Step 5).** The AA of the IdP authenticates the attribute request, maps the DN to a local principal name, using the GridShib plug-in, retrieves the requested attributes for the user (suitably filtered by normal Shibboleth-attribute release policies), formulates an SAML-attribute assertion, and sends the assertion to the Grid SP.

**Step 6).** The Grid SP parses the SAML attribute assertion, caches the attributes, makes an access control decision, processes the client request (assuming access is granted) and returns a response to the Grid portal. The Grid portal formats the result into standard Web pages and sends it to the Web browser.

## **5.7. Summary**

This chapter focuses on the security issues and corresponding solution for the Grid-enabled, distributed data warehouse system. The proposed system is an OGSA-based Grid system, which consists of multiple, untrusted participants. The nature of VO introduces a number of security challenges that are far more complex compared to the traditional distributed diagram. The primary security requirements include cross-domain authentication, global subject mapping to local subject, credential protection, dynamical delegation, SSO, and cross-domain access control. The security policy within each institution might be changed frequently, so the security issues should be addressed at infrastructure-level in order to release the burden of changing applications. Additionally, the proposed Grid system uses a Grid portal, which provides a Web-based interface, allowing users to access Grid resources by using standard Web browsers at any time and place. Therefore, it is necessary to find a way to bridge the incompatibility between Web and Grid-security protocols.

The security solution is based on the GSI, which is a portion of GT. GSI consists of a number of components focused on different issues. It is based on a PKI with certificate authorities and X.509 certificates. It provides a

public-key system; mutual authentication through digital certificates; credential delegation and SSO. It acts as Grid security middleware to provide infrastructure-level security functionalities for addressing security issues in a Grid environment. The GSI4, which is the latest version of GSI, implements the OGSA security architecture, by combining OGSA and existing or emerging Web-services security standards and specifications. This approach allows the basic security issues to be addressed by Grid infrastructure. Therefore, the Grid applications only need to deal with application-specific security policy without considering the frequent changes of institutions' policies.

The solution is provided by GSI4. The X.509 certificate is used with TLS to provide transport-level message protection. The Web-services security standards and specifications are used to provide end-to-end message-level protection. The X.509 Proxy Certificate is used to achieve dynamic delegation and SSO. The MyProxy protocol acts as an online credentials repository that solves the incompatibility between Web- and Grid-security protocols. Authorization is provided by Shibboleth, as well as GridShib plug-ins, which integrate X.509 credentials with SAML. These components are integrated into a security model, which can be implemented to ensure the proposed Grid system works in a secure way.

## **Chapter 6.**

### **Conclusion**

Grid computing, which has emerged as a state-of-the-art cross-domain approach, is concerned with heterogeneous, distributed resource-sharing for cross-institutional collaboration. It uses standard, open, general-purpose protocols and interfaces to coordinate resource-sharing within a virtual organization (VO) for delivering various services. Combining Grid technologies with appropriate database access and integration technology is essential in the Grid computing paradigm.

Chapter 2 provides a comprehensive introduction to current Grid technologies and database technologies. Grid technology has evolved at a rapid rate. The emphasis of the Grid system has shifted from early meta-computing to distributed global collaboration, a service-oriented approach and information-layer issues. Currently, Grid computing is closely related to the evolution of Web technologies and standards. Current database management technology has already addressed new challenges, like scale, heterogeneity and distribution. However, the major limitations, such as cross-domain federated query and security, do exist. The combination of Grid and database access and integration enables the transparent access of distributed data resources, made possible by using resource virtualization methods.

This project proposes a Grid-enabled, distributed data warehouse system, which provides infrastructure-level services, such as is required to support HIV/AIDS collaborative research. In this case, the Grid is primarily used to: control workflow of data-collection; to coordinate data-access and authorisation across multiple institutions; to integrate various types of data into a single format; and to manage the collaborative data-analysis operations. In the HIV/AIDS example problem-context, patients' data is collected and populated into the collector's data warehouse, which provides the primary type of resource - data. A number of HIV/AIDS-related data-analysis services can be built over the established data warehouse for the purpose of research. The data-analysis

services are considered as computing resources. Data and computing resources, resource providers, and resources consumers comprise a VO that consists of multiple, physically distributed institutions. The proposed system is responsible for providing essential capabilities by defining a rich set of services with standard interfaces and protocols.

This dissertation focuses on two main objectives.

- The first objective involved defining a feasible framework for the Grid-enabled, distributed data warehouse system, according to existing standards, specification, and implementation. The proposed system is designed as an OGSA-based Grid system. There exists a number of standards and specifications (e.g., Web services family, WSRF, and WS-Notification) to support the realization of OGSA. A system framework is proposed through a thorough study of all these standards, specifications and corresponding implementations. Based on this framework, two sub-objectives were identified. The first was to identify core services, providing fundamental capabilities required by the proposed system. The second was to define two component models for (a) data-collection for data warehousing among disparate data providers in a VO and for (b) data-access and data-integration in a VO.
- The second objective involved addressing particular security issues, including authentication, SSO, dynamic delegation and authorization for the proposed system. The security solution was derived based on a literature survey of GSI (Grid Security Infrastructure) and related standards and specifications. GSI concerns general security aspect in Grid environments. Based on the study of the security mechanisms of GSI, and supporting standards and specifications (such as PKI, Web service security, SAML, Shibboleth, etc.), the required components were selected to fulfil the security requirements of the proposed system. One sub-objective in the security area was to define a model that integrates the relevant security components to illustrate how to establish cross-domain trust relationships and to provide secure conversation between VO entities.

Chapter 3 firstly describes the characteristics of a VO environment for the envisaged Grid system and then discusses the essential OGSA (Open Grid Services Architecture) functional capabilities, based on a pre-defined VO. A set of capabilities, including resource virtualization, resource publication and discovery, data-collection, data operation, provenance, resource management, job-execution management, metadata, monitoring and security, are identified for the proposed system. The combination of these capabilities can provide advanced capabilities, such as data-collection, data-sharing coordination and collaborative operations.

Based on the discussion in chapter 3, chapter 4 focuses on the first main objective and its sub-objectives. A layered system framework is defined in order to identify all supportive standards and specifications used to design and implement the proposed system. OGSA is a *de facto* standard for building a Grid system. It is the core standard in this framework. WSRF, WS-Notification and Web service family are supportive standards and specification to realize OGSA. OGSA introduces a service-oriented Grid architecture, which tailors a Web services approach to meet some Grid-specific requirements. It defines what is called Grid service to represent resources (either data or computational resources) in a VO by using stateful Web services with standard interfaces and protocol binding. A Grid service is essentially a stateful Web service. Giving Web services the ability to keep state information, while still keeping them stateless, seems like a complex problem. WS-Resource is an approach defined by WSRF to model stateful resources in a Web-services context. The state information is kept in a stateful resource instead of Web service. WS-Notification specifications use standard approaches to notification, using a topic-based publish-and-subscribe pattern. Based on the OGSA, WSRF, Web services-standard, the infrastructure services and core services were defined to offer capabilities to the Grid-enabled, distributed data warehouse system. Infrastructure services provide the foundation for building core services. They consist of a number of common components for naming, representing state, notification and security. The core services were defined to meet the primary requirements of the proposed Grid system. Core services contain a large set of services to provide a variety of capabilities, such as data-access, data integration,

resources management, system monitoring, job execution, etc. These services are compliant with OGSA Grid services. It means they can be implemented by using OGSA-based development toolkits, such as GT and OGSA-DAI. Based on all identified services, two models were created to demonstrate how to integrate these services to achieve data collecting from distributed providers and data-sharing across multiple, distributed institutions. The models use a Grid portal as a Web-based brokerage point that allows authorized users to access Grid resources via Web browsers.

Web services' standards have evolved at a fast rate. Today there are many specifications that provide Web-service capabilities for resources, events, and management. Some examples are: WS-Transfer, WSRF, WS-Notification, WS-Eventing, WS-Management, WSDM specifications, etc. In 2006, IBM, HP, Intel and Microsoft plan to develop a common set of specifications for resources, events and management that can be broadly supported across heterogeneous systems (Cline et al., 2006). For example, the new WS-EventNotification specification composes with WS-ResourceTransfer to support a state/resource model for managing subscriptions. The existing functionality of WS-Notification, not explicitly defined in WS-EventNotification, can still be layered over its message model and functionality as an extension. The new Web services management specification comprises new WS-ResourceTransfer and new WS-EventNotification. The differences between WS-Management and WSDM specifications are directly due to the differences between WS-Transfer and WSRF, and between WS-Notification and WS-Eventing. The reconciliation of the resource management and event/notification specifications enable reconciliation of many of the functions of the management specifications. This plan outlined the approach to build on existing specifications and defined a set of enhancements that enable this convergence. Future work is based on this convergence in order to allow the proposed system migrate smoothly to the new specifications as they emerge and move through to standardization.

Chapter 5 focused on the second objective: a security solution for particular security issues in the Grid-enabled, distributed data warehouse system. The

Grid-enabled, distributed data warehouse system consists of multiple untrusted institutions. Each institution may have various security policies and use different mechanism to handle intra-domain security issues. The main goals of the envisaged security solution are providing cross-domain mutual authentication, global subject mapping to local subject, credential protection, dynamical delegation, SSO, and cross-domain access control. The security solution is based on GSI, which is a PKI-based infrastructure for addressing general security issues in a Grid environment. For authentication, the X.509 credentials are used either via TLS, in the case of transport-level security, or via digital signatures, as specified by WS-Security, in the case of message-level security. The X.509 Proxy Certificate is used to achieve dynamic delegation and SSO. The MyProxy protocol acts as an online credentials repository that solves the incompatibility between Web- and Grid-security protocols. Shibboleth, as well as GridShib, are used to provide access control. The authorization decision is dependent on users' X.509 credentials and attributes retrieved from an AA by using SAML attribute query and assertion. All these components are finally integrated into a security model, which can be implemented to establish trust relationships and provide secure interaction between untrusted institutions.

A major criticism against the authorization approach discussed in this dissertation is that the user still needs to obtain a valid certificate from a CA. SAML and XACML are two important authorization-related standards. The authorization discussed in this dissertation does not use the full potential of SAML and XACML. The combination of SAML and XACML can enable federated identity management, and federated access management. This approach does not depend on the user have a PKI certificate and provide seamless access control. In chapter 5, GT4 SAML/XACML authorization framework was such an example. Future work in this security area will focus on the study on SAML/XACML framework, related standards and implementation toolkits in order to provide more seamless and scalable access control in the Grid-enabled, distributed data warehouse system.

In summary, this dissertation discussed a flexible framework and security solution for a Grid-enabled, distributed data warehouse system (exemplified in the context of HIV/AIDS collaborative research). The main contribution of this work was exploring a feasible approach, which uses Grid technology to support data-sharing and collaboration over distributed data warehouses (or databases) for the purpose of data analysis (or data-mining) in a VO context.

## References

- Abiteboul, S., Buneman, P., & Suciu, D. (1999). *Data on the web: from relations to semistructured data and XML*. Los Altos: Morgan Kaufmann.
- Allen, G., Daues, G., Foster, I., Laszewski, G., Novotny, J., Russell, M. (Eds.). (2001). *The Astrophysics Simulation Collaboratory Portal: A Science Portal Enabling Community Software Development*. Proceeding of the 10<sup>th</sup> IEEE International Symposium On High Performance Distributed Computing 2001 (HPDC '01), San Francisco, California, USA. CS Press.
- Almond, J., & Snelling, D. (1999). *UNICORE: uniform access to supercomputing as an element of electronic commerce*. Future Generation Computer Systems, 15, 539-548, NH-Elsevier.
- Alpdemir, M. N., Mukherjee, A., Gounaris, A., Paton, N. W., Watson, P., Fernandes, A. A. A. (2003). *OGSA-DQP: A Service-Based Distributed Query Processor for the Grid*. Proceedings of the UK e-Science All Hands Meeting 2003, September 2003.
- Antonioletti, M., Atkinson, M. P., Baxter, R., Borley, A., Chue Hong, N. P., Collins, B. (Eds.). (2005). *The Design and Implementation of Grid Database Services in OGSA-DAI*. Concurrency and Computation: Practice and Experience, 17 (2-4), 357-376, February 2005.
- Armstrong, R., Gannon, D., Geist, A., Keahey, K., Kohn, S., McInnes, L. (Eds.). (1999). *Toward a common component architecture for high performance scientific computing*. In Proceedings of the 8th High Performance Distributed Computing (HPDC99).
- Atkinson, M., L.Chervenak, A., Kunszt, P., Narang, I., W. Paton, N., Pearson, D. (Eds.). (2004). Data Access, Integration, and Management. In Foster, I & Kesselman, C., *The Grid 2: Blueprint for a New Computing Infrastructure* (pp 391-429). San Francisco: Morgan Kaufmann.
- Atkinson, M., Dialani, V., Guy, L., Narang, I., Paton, N. W., Pearson, D. (Eds.). (2003). *Grid Database Access and Integration: Requirements and Functionalities*. Global Grid Forum, DAIS Working Group, GFD-I.13, March 13, 2003.
- Bajaj, S., Della-Libera, G., Dixon, B., Dusche, M., Hondo, M., Hur, M. (Eds.). (2003). *Web Services Federation Language (WS-Federation) Version 1.0*. IBM, Microsoft, RSA, Verisign, July 8 2003.

- Baker, F. (1995). *Requirements for IP Version 4 Routers*. IETF, RFC 1812. Retrieved November 2005, from <http://www.ietf.org/rfc/rfc1812.txt>.
- Ballinger, K., Ehnebuske, D., Ferris, C., Gudgin, M., Karmarkar, A., Liu, C. K. (Eds.). (2006). *Web Services Interoperability Organization (WS-I) Basic Profile Version 1.2*. Working Group Draft 2006-10-03. Retrieved November 2006 from <http://members.ws-i.org/dman/Docs.phx?Working+Groups/WSBasic+Profile/Profile/BP1.2/BasicProfile-1.2-WGD.html?versionID=1>.
- Ballinger, K., Brittenham, P., Malhotra, A., Nagy, W. A., & Pharies, S. (2001). *Web Services Inspection Language (WS-Inspection) 1.0*. IBM DeveloperWorks, November 2001. Retrieved March 2006 from <http://www-128.ibm.com/developerworks/library/specification/ws-wsilec/>.
- Barton, T., Basney, J., Freeman, T., Scavo, T., Siebenlist, F., Welch, V. (Eds.). (2006). *Identity Federation and Attribute-based Authorization through the Globus Toolkit, Shibboleth, GridShib, and MyProxy*. In 5th Annual PKI R&D Workshop, April 2006.
- Basney, J. (2005). *MyProxy Protocol*. Global Grid Forum, GFD-E.054, November 26, 2005.
- Bell, W. H., Bosio, D., Hoschek, W., Kunszt, P., McCance, G., & Silander, M. (2002). *Project Spitfire – Towards Grid Web Service Databases*. Global Grid Forum, DAIS Working Group Informational Document.
- Birrell, A.D., & Nelson, B.J. (1984). *Implementing Remote Procedure Calls*. ACM Transactions on Computer Systems, 2(1), 39-59.
- Box, D. (2003). *Service-Oriented Architecture and Programming (SOAP) - Part 1 & Part 2*. MSDN TV archive, 2003.
- Brittenham, P. (2002). *An Overview of the Web Services Inspection Language*. IBM DeveloperWorks, 01 January 2002. Retrieved June 2006 from [www.ibm.com/developerworks/webservices/library/ws-wsiloer](http://www.ibm.com/developerworks/webservices/library/ws-wsiloer).
- Burbeck, S. (2000). *The Tao of e-business Services*. IBM DeveloperWorks, 01 October 2000. Retrieved July 2006 from <http://www.ibm.com/developerworks/webservices/library/ws-tao/>.

- Butler, R., Engert, D., Foster, I., Kesselman, C., Tuecke, S., Volmer, J. & Welch, V. (2000). *Design and Deployment of a National-Scale Authentication Infrastructure*. IEEE Computer, 33(12), 60-66.
- Buyya, R., Abramson, D., & Giddy, J. (2000). *Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid*. In Proceeding of the 4th International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia2000), Beijing, China.
- Cabrera, F. L., Copeland, G., Feingold, M., Freund, R. W., Freund, T., Johnson, J. (Eds.). (2005). *Web Services Coordination (WS-Coordination) Version 1.0*. IBM, DeveloperWork, August 2005, Retrieved May 2006 from <http://www-128.ibm.com/developerworks/library/specification/ws-tx/>.
- Callas, J., Donnerhacke, L., Finney, H., Thayer, R. (1998). *OpenPGP Message Format*. IETF, Network Working Group, RFC 2440, November 1998. Retrieved August 2006 from <http://www.ietf.org/rfc/rfc2440.txt>.
- Cantor, S. et al. (2005). *Shibboleth Architecture: Protocols and Profiles*. Internet2-MACE, 10 September 2005.
- Chadwick, D.W., & Otenko, O. (2002). *The PERMIS X.509 Role Based Privilege Management Infrastructure*. Proceedings of 7th ACM Symposium on Access Control Models and Technologies (SACMAT 2002).
- Channabasavaiah, K., Holley, K., & Tuggle, E. (2003). *Migrating to a service-oriented architecture, Part 1*. IBM DeveloperWorks, 16 December 2003. Retrieved July 2006 from <http://www-128.ibm.com/developerworks/library/ws-migratesoa/>.
- Clark, D. (2001). *Face-to-Face with Peer-to-Peer Networking*. Computer, 34(1), 18-21.
- Cline, K., Cohen, I., Davis, D., Ferguson, D. F., Kreger, D., & McCollum, R. (2006). *Toward Converging Web Service Standards for Resources, Events, and Management*. HP, IBM, Inter and Microsoft, Joint White Paper V1.0, March 15, 2006.
- Czajkowski, K., Ferguson, D., Foster, I., Frey, J., Graham, S., Maguire, T. (Eds.). (2004)<sup>1</sup>. *From Open Grid Services Infrastructure to WS-Resource Framework: Refactoring & Evolution Version 1.1 (3/05/2004)*. Globus Alliance.

- Czajkowski, K., Ferguson, D., Foster, I., Frey, J., Graham, S., Sedukhin, I. (Eds.). (2004)<sup>2</sup>. *The WS-Resource Framework Version 1.0*. Globus Alliance.
- Daniel, J. B., Silverman, R. E., & Byrnes, R. G. (2005). *SSH: The Secure Shell, The Definitive Guide (2<sup>nd</sup> edition)*. O'Reilly.
- Davidson, S. B., Crabtree, J., Brunk, B. P., Schug, J., Tannen, V., Overton, G. C. (Eds.). (2001). *K2/Kleisli and GUS: Experiments in integrated access to genomic data sources*. IBM Systems Journal, 40(2), 512-531.
- de Laat, C., Gross, G., Gommans, L., Vollbrecht, J., & Spence, D. (2000). *Generic AAA Architecture*. IETF, Network Working Group, RFC 2903, August 2000. Retrieved February 2006 from <http://www.ietf.org/rfc/rfc2903.txt>.
- DeFanti, T., Foster, I., Papka, M., Stevens, R., & Kuhfuss, T. (1996). *Overview of the I-WAY: Wide area visual supercomputing*. International Journal of Supercomputer Applications, 10, 123-130.
- De Roure, D., A. Baker, M., R. Jennings, N & R. Shadbolt, N. (2003). *The Evolution of The Grid*. In Berman, F., Fox, G., & J. G. Hey, A., *Grid Computing: Making The Global Infrastructure a Reality* (pp. 65–100). John Wiley & Sons.
- Dierks, T., & Rescorla, E. (2006). *The Transport Layer Security (TLS) Protocol Version 1.1*. IETF, Network Working Group, RFC 4346, April 2006. Retrieved December 2006 from <http://tools.ietf.org/html/rfc4346>.
- Dinda, P., & Plale, B. (2001). *A unified relational approach to Grid information services*. Global Grid Forum, Technical Report GWD-GIS-012-1.
- DMTF. (1999). *Common Information Model (CIM) Specification, Version 2.2*. Distributed Management Task Force (DMTF) specification. Retrieved August 2006 from <http://www.dmtf.org/standards/documents/CIM/DSP0004.pdf>
- Elmasri, R., & Navathe, S. B. (2000). *Fundamentals of Database Systems (3<sup>rd</sup> ed.)*. Addison-Wesley.
- Erdos, M & Cantor, S. (2001). *Shibboleth-Architecture DRAFT v04*. Shibboleth Project Specification, Internet2/MACE.

- Farrell, S., & Housley, R. (2002). *An Internet Attribute Certificate Profile for Authorization*. IETF, Network Working Group, RFC 3281, April 2002. Retrieved February 2006 from <http://www.ietf.org/rfc/rfc3281.txt>.
- FIPA. (2002). *FIPA Communicative Act Library Specification*. Foundation for Intelligent Physical Agents (FIPA) Technical Report XC00037I, October 2002.
- Foster, I. (2002)<sup>1</sup>. *The Grid: A New Infrastructure for 21st Century Science*. Physics Today, 55 (2), 42-47.
- Foster, I. (2002)<sup>2</sup>. *What is the Grid? A Three Point Checklist*. Retrieved November 2005 from <http://wwwfp.mcs.anl.gov/~foster/Articles/WhatIsTheGrid.pdf>.
- Foster, I. (2004). *Brain Meets Brawn: Why Grid and Agents Need Each Other*. In Proceeding of 3rd International Conference on Autonomous Agents and Multi Agent Systems, New York.
- Foster, I., Frey, J., Graham, S., Tuecke, S., Czajkowski, K., Ferguson, D. (Eds.). (2004)<sup>1</sup>. *Modeling Stateful Resources with Web Services Version 1.1*. Globus Alliance.
- Foster, I., Gannon, D., Kishimoto, H & Von Reich, Jeffrin. J. (2004)<sup>2</sup>. *Open Grid Services Architecture Use Cases*. Global Grid Forum OGSA-WG, GFD-I.029, October 2004.
- Foster, I., Geisler, J., Nickless, W., Smith, W., & Tuecke, S. (1997). *Software Infrastructure for the I-WAY High Performance Distributed Computing Experiment*. In Proceeding of the 5th IEEE Symposium on High Performance Distributed Computing, 562-571
- Foster, I., & Kesselman, C. (1997). *Globus: A Metacomputing Infrastructure Toolkit*. International Journal of Supercomputer Applications, 11(2), 115-128.
- Foster, I., & Kesselman, C. (1998). *The Globus Project: A Status Report*. In Proc. Heterogeneous Computing Workshop, IEEE Press, 1998, 4-18.
- Foster, I., & Kesselman, C. (1999). *Globus: A Toolkit-Based Grid Architecture*. In Foster, I & Kesselman, C., *The Grid: Blueprint for a New Computing Infrastructure* (pp. 259-278). San Francisco: Morgan Kaufmann.

- Foster, I., & Kesselman, C. (1998). *The Grid: Blueprint for a new Computing Infrastructure*. San Francisco: Morgan Kaufmann.
- Foster, I., Kesselman, C., Nick, J., & Tuecke, S. (2002). *The Physiology of the Grid: Open Grid Services Architecture for Distributed Systems Integration*. Global Grid Forum, Open Grid Service Infrastructure WG. Presented at GGF4, February 2002.
- Foster, I., Kesselman, C., Tsudik, G. & Tuecke, S. (1998). *A Security Architecture for Computational Grids*. 5th ACM Conference on Computer and Communications Security, 83-91.
- Foster, I., Kesselman, C., & Tuecke, S. (2001). *The Anatomy of the Grid: Enabling Scalable Virtual Organizations*. International Journal of High Performance Computing Applications, 15 (3), 200-222.
- Foster, I., Kishimoto, H., Savva, A., Berry, D., Djaoui, A., Grimshaw, A. (Eds.). (2005). *The Open Grid Services Architecture, Version 1.0*. Global Grid Forum OGSA-WG. GFD-I.030, 29 January 2005.
- Foster, I., Kishimoto, H., Savva, A., Berry, D., Djaoui, A., Grimshaw, A. (Eds.). (2006). *The Open Grid Services Architecture, Version 1.5*. Global Grid Forum OGSA-WG. GFD-I.080, 24 July 2006.
- Frawley, W., Piatetsky-Shapiro, G., & Matheus, C. (1992). *Knowledge Discovery in Databases: An Overview*. AI Magazine, Fall 1992, 213-228.
- Funderburk, J. E., Kiernan, G., Shanmugasundaram, J., Shekita, E., & Wei, C. (2002). *XTABLES: Bridging relational technology and XML*, IBM Systems Journal, 41(4), 616–641.
- Gannon, D. (Eds.). (2002). *Programming the Grid: Distributed Software Components, P2P and Grid Web Services for Scientific Applications*. Cluster Computing, 5(3), 325-336.
- Gopalan, S.R. (1998). *A Detailed Comparison of CORBA, DCOM and Java/RMI, Object Management Group (OMG)*. Object Management Group (OMG) White Paper.
- Graham, S., Niblett, P., Chappell, D., Lewis, A., Nagaratnam, N., Parikh, J. (Eds.). (2004). *Publish-Subscribe Notification for Web services Version 1.0 03/05/2004*. Globus Alliance.

- Grimshaw, A., Wulf, W. (Eds.). (1997). *The Legion Vision of a Worldwide Virtual Computer*. Communications of the ACM, 40(1).
- Grimshaw, A. (Eds.). (2006). *WS-Naming Specification. Draft, March 2006*. Open Grid Forum, OGSA-NAMING-WG. Retrieved August 2006 from <http://forge.gridforum.org/projects/ogsa-naming-wg/>.
- Grimshaw, A., Newhouse, S., Pulsipher, D., & Morgan, M. (2006). *OGSA Basic Execution Services Version 1.0*. Global Grid Forum OGSA-BES-WG, Draft 16, February 2006. Retrieved August 2006 from <https://forge.gridforum.org/projects/ogsa-bes-wg/document/ogsa-bes-draft-v16/en/1>
- Groth, R. (1997). *Data Mining: A Hands-On Approach for Business Professionals*. Prentice Hall.
- Haas, L. M., Schwarz, P. M., Kodali, P., Kotlar, E., Rice, J. E., & Swope, W. C. (2001). *DiscoveryLink: A system for integrated access to life sciences data sources*. IBM Systems Journal 40(2), 489-511.
- Hand, D., Mannila, H., & Smyth, P. (2001). *Principles of Data Mining*. Cambridge, MA: MIT Press.
- He, H. (2003). *What is Service-Oriented Architecture?*. O'REILLY XML.COM Article Archive. Retrieved 21 July 2006 from <http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html>.
- Horstmann, M., & Kirtland, M. (1997). *DCOM Architecture*. Microsoft, MSDN Library, DCOM Technical Article, 23 June 19997.
- Hoschek, W., Jaen-Martinez, J., Samar, A., Stockinger, H., & Stockinger, K. (2000). *Data Management in an International Data Grid Project*. In Proceedings of the 1st IEEE/ACM International Workshop on Grid Computing (Grid 2000), Bangalore, India, 17-20. Springer-Verlag Press, Germany.
- Housley, R., Polk, W., Ford, W., & Solo, D. (2002). *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. IETF, Network Working Group, RFC 3280, April 2002. Retrieved May 2006 from <http://www.faqs.org/rfcs/rfc3280.html>.
- IBM. (2001). *Web services Object Runtime Framework: implementing Web services with XML Extender, Version 7.2 (beta)*. Retrieved June 2006

from  
<http://www-306.ibm.com/software/data/db2/extenders/xmltext/docs/v72wrk/dxxservl.htm>.

IBM & Microsoft. (2002). *Security in a Web Services World: A Proposed Architecture and Roadmap*. IBM Corporation and Microsoft Corporation, A joint security whitepaper, April 7, 2002, Version 1.0.

ITU-T. (1996). *Information technology – Open Systems Interconnection–Security Frameworks for Open Systems: Access Control Framework*. ITU-T Recommendation X.812, ISO Published Standard: [ISO/IEC 10181-3](#).

ITU-T. (2005). *Information Technology - Open Systems Interconnection - The Directory: Authentication Framework*. ITU-T Recommendation X.509.

Inmon, W. H. (1996). *Building Data Warehouse (2<sup>nd</sup> ed.)*. John Wiley & Sons.

Inmon, W. H., & Hackathorn, Richard. D. (1994). *Using the Data Warehouse (1st ed.)*. John Wiley & Sons.

Jennings, N. R. (2001). *An agent-based approach for building complex software systems*. Comms. of the ACM, 44 (4), 35-41.

Karasavvas, K., Antonioletti, M., Atkinson, M. P., Chue Hong, N. P., Sugden, T., Hume, A.C. (Eds.). (2005). *Introduction to OGSA-DAI Services*. Lecture Notes in Computer Science, Volume 3458, Pages 1-12, May 2005.

Kossmann, D. (2000). *The state of the art in distributed query processing*. ACM Computing Surveys, 32 (4), 422-469.

Krawczyk, H., Bellare, M., & Canetti, R. (1997). *HMAC: Keyed-Hashing for Message Authentication*. IETF, Network Working Group, RFC 2104, February 1997. Retrieved December 2006 from <http://www.ietf.org/rfc/rfc2104.txt>.

Krishnan, S., Bramlay, R., Gannon, D., Govindaraju, M., Indurkar, R., Solminski, A. (Eds.). (2001). *The XCAT Science Portal*. Proceeding of Super Computing 2001 (SC '01), Denver, Colorado, USA. CS Press.

Lang, B., Foster, I., Siebenlist, F., Ananthakrishnan, R., & Freeman, T. (2006). *A Multipolicy Authorization Framework for Grid Security*. Proceedings

of the Fifth IEEE International Symposium on Network Computing and Application, 2006.

Lee, A., Magowan, J., Dantressangle, P., & Bannwart, F. (2005). *Bridging the integration gap*. IBM Developer Works. 16 Aug 2005.

Lenzerini, M. (2002). *Data Integration: A Theoretical Perspective*. PODS 2002, 243-246.

Lepro, R. (2003). *Cardea: Dynamic Access Control in Distributed Systems*. NASA Technical Report NAS-03-020, November 2003.

Li, M., & Baker, M. (2005). *The Grid Core Technologies*. John Wiley & Sons.

Lorch, M., Adams, D., Kafura, D., Koneni, M., Rathi, A., & Shah, S. (2003). *The PRIMA System for Privilege Management, Authorization and Enforcement in Grid Environments*. 4th Int. Workshop on Grid Computing - Grid 2003, 17 November 2003, Phoenix, AR, USA.

Lorch, M., Basney, J., & Kafura, D. (2004). *A Hardware-secured Credential Repository for Grid PKIs*. 4th IEEE/ACM International Symposium on Cluster Computing and the Grid, Chicago, Illinois, April 19-22, 2004.

Lorch, M., Cowles, B., Baker, R., Gommans, L., Madsen, P., McNab, A. (Eds.). (2004). *Conceptual Grid Authorization Framework and Classification*. Global Grid Forum, Authorization Frameworks and Mechanisms-WG, GFD-I.038, Revised November 23 2004.

Lorch, M., & Kafura, D. (2002). *Supporting Secure Ad-hoc User Collaboration in Grid Environments*. 3rd Int. Workshop on Grid Computing, Baltimore, November, 18, 2002.

Maciel, F. B. (2005). *Resource Management in OGSA*. Global Grid Forum, CMM-WG. GFD-I.045, March 1, 2005.

McBrien, P., & Poullovassilis, A. (2002). *Schema Evolution in Heterogeneous Database Architectures, A Schema Transformation Approach*. CAiSE, 484-499.

Myers, M., Ankney, R., Malpani, A., Galperin, S., & Adams, C. (1999). *X.509 Internet Public Key Infrastructure Online Certificate Status Protocol –*

OCSF. IETF, Network Working Group, RFC 2560, June 1999. Retrieved May 2006 from <http://www.ietf.org/rfc/rfc2560.txt>.

Nagy, W. A., & Ballinger, K. (2001). *The WS-Inspection and UDDI relationship*. IBM DeveloperWorks, 01 November 2001. Retrieved July 2006 from <http://www-128.ibm.com/developerworks/webservices/library/ws-wsilluddi.html>

Neuman, B. C., & Ts'o, T. (1994). *Kerberos: An Authentication Service for Computer Networks*. IEEE Communications Magazine, 32 (9), 33-88.

Neuman, C., Yu, T., Hartman, S., & Raeburn, K. (2005). *The Kerberos Network Authentication Service (V5)*. IETF, Network Working Group, RFC 4120, July 2005. Retrieved September 2006 from <http://tools.ietf.org/html/rfc4120>.

Novotny, J. (2002). *The Grid Portal Development Kit*. Concurrency and Computation: Practice and experience, 14(13-15), 1129-1144.

Novotny, J., Tuecke, S., & Welch, V. (2001). *An Online Credential Repository for the Grid: MyProxy*. Proceedings of the Tenth International Symposium on High Performance Distributed Computing (HPDC-10), IEEE Press, August 2001.

OASIS. (2005)<sup>1</sup>. *Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0*. OASIS Standard, 15 March 2005. Retrieved September 2006 from <http://docs.oasis-open.org/security/saml/v2.0/>.

OASIS. (2005)<sup>2</sup>. *eXtensible Access Control Markup Language 2 (XACML) Version 2.0*. OASIS Standard, 1 Feb 2005. Retrieved September 2006 from [http://docs.oasis-open.org/xacml/2.0/access\\_control-xacml-2.0-core-spec-os.pdf](http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf)

OASIS. (2005)<sup>3</sup>. *Metadata for the OASIS Security Assertion Markup Language (SAML) V2.0*. OASIS Standard, 15 March 2005. Retrieved September 2006 from <http://docs.oasis-open.org/security/saml/v2.0/>.

OASIS. (2005)<sup>4</sup>. *SAML Attribute Sharing Profile for X.509 Authentication-Based Systems*. Committee Draft, 1 June 2005. Retrieved September 2006 from

[http://www.oasis-open.org/committees/documents.php?wg\\_abbrev=security](http://www.oasis-open.org/committees/documents.php?wg_abbrev=security).

OASIS. (2005)<sup>5</sup>. *Web Services Distributed Management: Management of Web Services (WSDM-MOWS) 1.0*. OASIS-Standard, 9 March 2005. Retrieved February 2006 from <http://docs.oasis-open.org/wsdm/2004/12/wsdm-mows-1.0.pdf>

OASIS. (2005)<sup>6</sup>. *Web Services Distributed Management: Management using Web Services (MUWS 1.0) Part 1*. OASIS Standard, 9 March 2005. Retrieved February 2006 from <http://docs.oasis-open.org/wsdm/2004/12/wsdm-muws-part1-1.0.pdf>.

OASIS. (2005)<sup>7</sup>. *Web Services Distributed Management: Management using Web Services (MUWS 1.0) Part 2*. OASIS Standard, 9 March 2005. Retrieved February 2006 from <http://docs.oasis-open.org/wsdm/2004/12/wsdm-muws-part2-1.0.pdf>.

OASIS. (2006)<sup>1</sup>. *Reference Model for Service Oriented Architecture 1.0*. OASIS Committee Specification 1, 19 July 2006. Retrieved December 2006 from [http://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=soa-rm](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm).

OASIS. (2006)<sup>2</sup>. *Web Services Base Faults 1.2 (WS-BaseFaults)*. Committee Specification, 9 January 2006. Retrieved May 2006 from [http://docs.oasis-open.org/wsrf/wsrf-ws\\_base\\_faults-1.2-spec-cs-01.pdf](http://docs.oasis-open.org/wsrf/wsrf-ws_base_faults-1.2-spec-cs-01.pdf)

OASIS. (2006)<sup>3</sup>. *WS-SecureConversation 1.3*. Committee Draft 01, 06 September 2006. Retrieved October 2006 from <http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512>.

OASIS. (2006)<sup>4</sup>. *Web Services Base Notification 1.3 (WS-BaseNotification)*. OASIS Standard, 1 October 2006. Retrieved November 2006 from [http://docs.oasis-open.org/wsn/wsn-ws\\_base\\_notification-1.3-spec-os.pdf](http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-os.pdf)

OASIS. (2006)<sup>5</sup>. *Web Services Brokered Notification 1.3 (WS-BrokeredNotification)*. OASIS Standard, 1 October 2006. Retrieved November 2006 from [http://docs.oasis-open.org/wsn/wsn-ws\\_brokered\\_notification-1.3-spec-os.pdf](http://docs.oasis-open.org/wsn/wsn-ws_brokered_notification-1.3-spec-os.pdf).

OASIS. (2006)<sup>6</sup>. *Web Services Service Group 1.2 (WS-ServiceGroup)*. Committee Specification, 9 January 2006. Retrieved May 2006 from

[http://docs.oasis-open.org/wsrf/wsrf-ws\\_service\\_group-1.2-spec-cs-01.pdf](http://docs.oasis-open.org/wsrf/wsrf-ws_service_group-1.2-spec-cs-01.pdf).

OASIS. (2006)<sup>7</sup>. *Web Services Resource Lifetime 1.2 (WS-ResourceLifetime)*. Committee Specification, 9 January 2006. Retrieved May 2006 from [http://docs.oasis-open.org/wsrf/wsrf-ws\\_resource\\_lifetime-1.2-spec-cs-01.pdf](http://docs.oasis-open.org/wsrf/wsrf-ws_resource_lifetime-1.2-spec-cs-01.pdf)

OASIS. (2006)<sup>8</sup>. *Web Services Resource Properties 1.2 (WS-ResourceProperties)*. Committee Specification, 20 January 2006. Retrieved May 2006 from [http://docs.oasis-open.org/wsrf/wsrf-ws\\_resource\\_properties-1.2-spec-cs-01.pdf](http://docs.oasis-open.org/wsrf/wsrf-ws_resource_properties-1.2-spec-cs-01.pdf)

OASIS. (2006)<sup>9</sup>. *Web Services Security Rights Expression Language (REL) Token Profile 1.1*. OASIS Standard, 1 February 2006. Retrieved May 2006 from <http://docs.oasis-open.org/wss/oasis-wss-rel-token-profile-1.1.pdf>.

OASIS. (2006)<sup>10</sup>. *Web Services Security: SOAP Message Security 1.1 (WS-Security 2004)*. OASIS Standard Specification, 1 February 2006. Retrieved May 2006 from <http://docs.oasis-open.org/wss/v1.1/>.

OASIS. (2006)<sup>11</sup>. *Web Services Security X.509 Certificate Token Profile 1.1*. OASIS Standard Specification, 1 February 2006. Retrieved May 2006 from <http://docs.oasis-open.org/wss/v1.1/>.

OASIS. (2006)<sup>12</sup>. *Web Services Topics 1.3 (WS-Topics)*. OASIS Standard, 1 October 2006. Retrieved November 2006 from [http://docs.oasis-open.org/wsn/wsn-ws\\_topics-1.3-spec-os.pdf](http://docs.oasis-open.org/wsn/wsn-ws_topics-1.3-spec-os.pdf).

OASIS. (2006)<sup>13</sup>. *WS-Trust 1.3*. Committee Draft 01, 06 September 2006. Committee Draft01, September 06, 2006. Retrieved October 2006 from <http://docs.oasis-open.org/ws-sx/ws-trust/200512>.

Pearlman, L., Welch, V., Foster, I., Kesselman, C., & Tuecke, S. (2002). *A Community Authorization Service for Group Collaboration*. Proceedings of the IEEE 3rd International Workshop on Policies for Distributed Systems and Networks, 2002.

Pearlman, L., Welch, V., Foster, I., Kesselman, C. & Tuecke, S. (2003). *The Community Authorization Service: Status and Futures*. Computing in High Energy Physics (CHEP03), 2003.

- Pearson, D. (2002). *Data Requirements for The Grid: Scoping Study Report*. UK DBTF working paper, presented at GGF4. Retrieved July 2006 from <http://www.cs.man.ac.uk/grid-db/>.
- Pyle, D. (2003). *Business Modeling and Data Mining (1<sup>st</sup> edition.)*. San Francisco: Morgan Kaufmann.
- Rajasekar, A.K., & Moore, R. W. (2001). *Data and Metadata Collections for Scientific Applications*. European High Performance Computing conference, Amsterdam, Holland.
- Ramakrishnan, R., & Gehrke, J. (1998). *Database Management Systems*. McGraw Hill.
- Raman, V., Narang, I., Crone, C., Haas, L., Malaika, S., Mukai, T. (Eds.). (2003). *Services for Data Access and Data Processing on Grids*. Global Grid Forum, DAIS Working Group, GFD-I.14, February 9, 2003.
- Schulte, R.W., & Natis, Y. V. (1996). *Service Oriented Architecture*. Gartner 12 April 1996.
- Sheth, A. P., & Larson, J. A. (1990). *Federated database systems for managing distributed, heterogeneous, and autonomous databases*. ACM Computing Surveys, 22 (3).
- Siebenlist, F., Welch, V., Tuecke, S., Foster, I., Nagaratnam, N., Janson, P. (Eds.). (2002). *Global Grid Forum Specification Roadmap towards a Secure OGSA*. Global Grid Forum, Draft 1.3, Revised July 19, 2002.
- Smarr, L., & Catlett, C. (1992). *Metacomputing*. Communication of ACM, 35, 44-52.
- Srinivasan, R. (1995). *RPC: Remote Procedure Call Protocol Specification Version 2*. IETF, Network Working Group, RFC 1831, August 1995. Retrieved July 2006 from <http://www.ietf.org/rfc/rfc1831.txt>.
- Stonebraker, M., Aoki, P.M., Litwin, W., Pfeffer, A., Sah, A., Sidell, J. (Eds.). (1996). *Mariposa: A Wide-Area Distributed Database System*. VLDB Journal 5(1), 48-63
- Stonebraker, M., & Moore, D. (1996). *Object-Relational DBMSs: The Next Great Wave*. Morgan Kaufmann.

- SUN. (2002). *JDBC 3.0 Specification (Final Release)*. Retrieved from 20 March 2006 from <http://java.sun.com/products/jdbc/>.
- Suzumura, T., Matsuoka, S. & Nakada, H. (2001). *A Jini-based Computing Portal System*. Proceeding of Super Computing 2001 (SC '01), Denver, Colorado, USA. CS Press.
- Thompson, M. (Eds.). (1999). *Certificate-based Access Control for Widely Distributed Resources*. Proceedings of 8th Usenix Security Symposium, 1999.
- Thompson, M. R., Olson, D., Cowles, R., Mullen, S., & Helm, M. (2003). *CA-based Trust Issues for Grid Authentication and Identity Delegation*. Global Grid Forum, Grid Certificate Policy Working Group, GFD-I.17, June 2003.
- Tierney, B., Aydt, R., Gunter, D., Smith, W., Swany, M., Taylor, V. (Eds.). (2002). *A Grid Monitoring Architecture*. Global Grid Forum, Performance Working Group, GFD.I.7, Revised 16-January-2002.
- Tuecke, S., Czajkowski, K., Foster, I., Frey, I., Graham, S., Kesselman, C. (Eds.). (2003). *Open Grid Services Infrastructure (OGSI) Version 1.0*. Global Grid Forum, OGSI-WG. GFD-R-P.15 (Proposed Recommendation), June 27, 2003.
- Tuecke, S., Welch, V., Engert, D. & Thompson, M. (2004). *Internet X.509 Public Key Infrastructure (PKI) Proxy Certificate Profile*. IETF, Network Working Group, RFC3820, June 2004. Retrieved February, 2006 from <http://www.rfc-archive.org/getrfc.php?rfc=3820>.
- Vollbrecht, J., Calhoun, P., Farrell, S., Gommans, L., Gross, G., de Bruijn, D. (Eds.). (2000). *AAA Authorization Framework*. IETF, Network Working Group, RFC 2904, August 2000. Retrieved February 2006 from <http://www.ietf.org/rfc/rfc2904.txt?number=2904>.
- VOMS Architecture v1.1*. (2002). Europe Union, Data Grid Project, Authorization Working Group, Draft May 09, 2002.
- Von Reich, Jeffrin. J. (Eds.). (2004). *Open Grid Services Architecture: Second Tier Use Cases*. Global Grid Forum OGSA-WG, Draft, March 2004.
- W3C. (2001). *XML Schema Part 0: Primer*. W3C, Recommendation, 2001, Retrieved June 2006 from <http://www.w3.org/TR/xmlschema-0/>.

- W3C. (2002)<sup>1</sup>. *XML Encryption Syntax and Processing*. W3C Recommendation 10 December 2002. Retrieved September 2006 from <http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/>.
- W3C. (2002)<sup>2</sup>. *XML-Signature Syntax and Processing*. W3C Recommendation 12 February 2002. Retrieved September 2006 from <http://www.w3.org/TR/2002/REC-xmlsig-core-20020212/>.
- W3C. (2003). *Simple Object Access Protocol (SOAP) version 1.2*. W3C Recommendation 24 June 2003. Retrieved July 2006 from <http://www.w3.org/TR/soap12>.
- W3C. (2004)<sup>1</sup>. *RDF Primer*. W3C Recommendation 10 February 2004. Retrieved May 2006 from <http://www.w3.org/TR/2004/REC-rdf-primer-20040210/>.
- W3C. (2004)<sup>2</sup>. *Resource Description Framework (RDF) Concepts and Abstract Syntax*. W3C Recommendation 10 February 2004, Retrieved May 2006 from <http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/>.
- W3C. (2004)<sup>3</sup>. *Web Services Architecture*. W3C Working Group Note 11 February 2004. Retrieved 20 January 2006 from <http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/>.
- W3C. (2006)<sup>1</sup>. *Extensible Markup Language (XML) 1.0 (Fourth Edition)*, W3C Recommendation 16 August 2006. Retrieved December 2006 from <http://www.w3.org/TR/2006/REC-xml-20060816>.
- W3C. (2006)<sup>2</sup>. *Web Services Addressing 1.0 – Core*. W3C Recommendation 9 May 2006. Retrieved July 2006 from <http://www.w3.org/TR/2006/REC-ws-addr-core-20060509>.
- W3C. (2006)<sup>3</sup>. *Web Services Description Language (WSDL) Version 2.0*. W3C Candidate Recommendation 27 March 2006. Retrieved July 2006 from <http://www.w3.org/TR/wsdl20/>.
- W3C. (2006)<sup>4</sup>. *Web Services Policy 1.2 - Framework (WS-Policy)*. W3C Member Submission 25 April 2006. Retrieved September 2006 from <http://www.w3.org/Submission/2006/SUBM-WS-Policy-20060425/>.
- W3C. (2006)<sup>5</sup>. *XML Path Language (XPath) 2.0*. W3C Candidate Recommendation 8 June 2006. Retrieved July 2006 from <http://www.w3.org/TR/2006/CR-xpath20-20060608/>.

- W3C. (2006)<sup>6</sup>. *XQuery 1.0: An XML Query Language* , W3C Candidate Recommendation 8 June 2006. Retrieved December 2006 from <http://www.w3.org/TR/2006/CR-xquery-20060608/>.
- Welch, V. (2005). *Globus Toolkit Version 4 Grid Security Infrastructure: A Standards Perspective*. Globus Alliance, Globus Security Team, Version 4 updated September 12, 2005.
- Welch, V., Barton, T., Keahey, K., & Siebenlist, F. (2005). *Attributes, Anonymity, and Access: Shibboleth and Globus Integration to Facilitate Grid Collaboration*. In 4th Annual PKI R&D Workshop, April 2005.
- Welch, V., Foster, I., Kesselman, C., Mulmo, O., Pearlman, L., Tuecke, S. (Eds.). (2004). *X.509 Proxy Certificates for Dynamic Delegation*. 3rd Annual PKI R&D Workshop, 2004.
- Welch, V., Siebenlist, F., Foster, I., Bresnahan, J., Czajkowski, K., Gawor, J. (Eds.). (2003). *Security for Grid Services*. Twelfth International Symposium on High Performance Distributed Computing (HPDC-12), June 2003.
- Westphal, C., & Blaxton, T. (1998). *Data Mining Solutions: Methods and Tools for Solving Real-World Problems*. Wiley.
- Whiteman, M. E., & Mattord, H. J. (2003). *Principles of Information Security*. Thomson.
- Wooldridge, M. (1997). *Agent-based software engineering*. IEE Proc. Software Engineering, 144, 26-37.