Research and Development of Accounting System in Grid Environment

A thesis submitted for the degree of Doctor of Philosophy

by

Xiaoyu Chen

School of Engineering and Design

Brunel University

November 2009

The piece of work is dedicated to my family and friends who had given peculiar supports and encourages during the period of my three-year researches.

Acknowledgement

There are many people who contributed financial, technical and moral supports that made this thesis possible. First of all, I am grateful for the supervision and guidance that I received from my supervisors, Prof. Akram Khan and Prof. Richard E. Taylor (Stanford University). I really appreciate the three-year studentship funded by Engineering and Physical Sciences Research Council (EPSRC) and Stanford University.

The majority of the work for this thesis was performed in cooperation with research groups outside Brunel. As considerable supports received, I would like to give my gratitude to JISC communities for their generous funding to our review tasks as demonstrated in Chapter 3, and to Dr. Michael A. Pettipher and Mr Tim W. Robinson for their close collaborations on the review tasks. I would also like to thank Dr. John Gordon and Dr. Dave Kant from Rutherford Appleton Laboratory (RAL) for their supports to the WLCG-RUS project discussed in Chapter 4.

I would also give my special thanks to all members and chairs of Usage Record (UR) and Resource Usage Service (RUS) working groups of Open Grid Forum for their supports to my chairship and efforts on specifications.

Finally, my parents definitely deserve my special acknowledgement to their endless supports making all possible.

Abstract

The Grid has been recognised as the next-generation distributed computing paradigm by seamlessly integrating heterogeneous resources across administrative domains as a single virtual system. There are an increasing number of scientific and business projects that employ Grid computing technologies for large-scale resource sharing and collaborations. Early adoptions of Grid computing technologies have custom middleware implemented to bridge gaps between heterogeneous computing backbones. These custom solutions form the basis to the emerging Open Grid Service Architecture (OGSA), which aims at addressing common concerns of Grid systems by defining a set of interoperable and reusable Grid services. One of common concerns as defined in OGSA is the Grid accounting service. The main objective of the Grid accounting service is to ensure resources to be shared within a Grid environment in an accountable manner by metering and logging accurate resource usage information. This thesis discusses the origins and fundamentals of Grid computing and accounting service in the context of OGSA profile. A prototype was developed and evaluated based on OGSA accounting-related standards enabling sharing accounting data in a multi-Grid environment, the World-wide Large Hadron Collider Grid (WLCG). Based on this prototype and lessons learned, a generic middleware solution was also implemented as a toolkit that eases migration of existing accounting system to be standard compatible.

Contents

List of Tables

List of Figures

List of Anonyms

Abbreviation	**Full Notation**
ALICE	A Large Ion Collider Experiment
APEL	Accounting Processor for Event Logs
ARC	Advanced Resource Connector
ATLAS	A Toroidal LHC Apparatu S
BDII	Berkeley Database Information Index
BES	Basic Execution Service
BLAH	Batch Local ASCII Helper
CA	Certificate Authority
CAS	Community Authorisation Service
CE	Computing Element
CERN	European Organisation for Nuclear Research
CIM	Common Information Model
CMS	Compact Muon Solenoid
CLI	Command-Line Interface
CPU	Central Processing Unit
CREAM	Computing Resource Execution and Management
CORBA	Common Object Request Broker Architecture
CRUD	Create, Read, Update, and Delete
DAO	Data Access Object
DAI	Data Access and Integration
DCOM	Distributed Component Object Model
DGAS	Distributed Grid Accounting System
DMTF	Distributed Management Task Force
DOM	Document Object Model
DPM	Disk Pool Manager
DRS	Data Replication Service
EDG	European Data Grid
EDVAC	Electronic Discrete Variable Automatic Computer
EGA	European Grid Alliance
EGEE	Enabling Grid for E-sciencE
EMC	Entity Model Compiler

Abbreviation	**Full Notation**
EMS	Execution Management Service
ENIAC	Numerical Integrator And Computer
EPCC	Edinburgh Parallel Computing Centre
EPR	Endpoint Reference
FQAN	Full Qualified Attribute Name
FTS	File Transfer Service
GIIS	Grid Index Information Service
GLUE	Grid Laboratory for a Uniform Environment
GMA	Grid Monitoring Architecture
GOC	Grid Operational Centre
GOSC	Grid Operation Support Centre
GPE	Grid Programming Environment
GRAM	Grid Resource Allocation Manager
GRIS	Grid Resource Information Service
GridRPC	Grid Remote Procedure Call
GSI	Grid Security Infrastructure
GT	Global Toolkit
GUID	Global Unique Identity
GRUS	Grid Resource Usage System
HLR	Home Location Registry
HPC	High Performance Computing
HQL	Hibernate Query Language
HTTP	HyperText Transfer Protocol
IC	Integrated Circuit
ITU	International Telecommunication Union
JAR	Java Archive
JARM	Job Account Resource Management
JDL	Job Description Language
JISC	Joint Information System Committee
JMX	Java Management eXtensions
JSDL	Job Submission Description Language
JSR	Java Specification Request
JWSDP	Java Web Service Development Pack
LAN	Local Area Network

Abbreviation	**Full Notation**
LB	Logging and Bookkeeping
LCAS	Local Centre Authorisation Service
LCMAPS	Local Credential Mapping Service
LDAP	Lightweight Directory Access Protocol
LFC	LCG File Catalogue service
LFN	Logical File Names
LFS	Load Sharing Facility
LHC	Large Hadrons Collider
LHCb	LHC-beauty
LRMS	Local Resource Management Service
LSI	Large-Scale Integration
LUTS	Logging and Usage Tracing Service
MCS	Market for Computation Service
MDS	Monitoring and Directory Service
MIMD	Multiple Instruction Multiple Data
MISD	Multiple Instruction Single Data
MPI	Message Passing Interface
MPP	Massive Parallel Processing
MVC	Model-Viewer-Controller
NDGF	Nordic Data Grid Facility
NIST	National Institute of Standards and Technology
NGS	National Grid Service
NUMA	Non-Uniform Memory Access
OASIS	Organisation of Advanced Standards for the Information Society
OGF	Open Grid Forum
OGSA	Open Grid Service Architecture
ORM	Object-Relational Mapping
OSG	Open Science Grid
PBS	Portable Batch System
POJO	Plain Old Java Object
PVM	Parallel Virtual Machine
RAL	Rutherford Appleton Laboratory
RBAC	Role-Based Access Control
RDBMS	Relational DataBase Management System

Abbreviation	**Full Notation**
RDF	Resource Description Framework
RFIO	Remote File Input/Output
RFT	Reliable File Transfer
RMI	Remote Method Interface
RPC	Remote Procedure Call
RSL	Resource Specification Language
RURF	RUS Usage Record Format
RUS	Resource Usage Service
R-GMA	Relational Grid Monitoring Architecture
RGOC	Regional Grid Operation Centre
RSS	Resource Selection Service
SAML	Security Assertion Markup Language
SAX	Simple API for XML
SE	Storage Element
SGAS	SweGrid Accounting System
SID	Service Interface Definition
SIMD	Single Instruction Multiple Data
SISD	Single Instruction Single Data
SLA	Service Level Agreement
SLM	Service Level Manager
SMP	Symmetric Multi-Processing
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
TLS	Transport Level Security
UAS	UNICORE Atomic Service
UDDI	Universal Description, Discovery and Integration
UI	User Interface
UML	Unified Modelling Language
UR	Usage Record
UNICORE	Uniform Interface to COmputing REsources
UNIVAC	UNIversal Automatic Computer
VDT	Virtual Data Toolkit
VLSI	Very-Large-Scale Integration
VO	Virtual Organisation

Abbreviation	**Full Notation**
VOMS	Virtual Organisation Management System
WAN	Wide Area Network
WAR	Web Archive
WLCG	World-wide LHC Computing Grid
WSDL	Web Service Description Language
WMS	Workload Management Service
WSN	Web Service Notification
WSRF	Web Service Resource Framework
XACML	eXtensible Access Control Markup Language
XIO	eXtensible Input/Output
XML	eXtensible Markup Language
XNJS	enhanced Network Job Supervisor
XOM	XML Object Model

List of Publications

M. A. Pettipher, A. Khan, T. W. Robinson, and X. Chen, “Review of Accounting and Usage Monitoring (final Report)”, *JISC Final Report*, Jul. 2007.

X. Chen and A. Khan, “Aggregative accounting service enabling economic modelling for commercial grid”, *Conf. on Grid technology for financial modelling and simulation*, Feb. 3-4, 2006, Palermo, Italy.

X. Chen and A. Khan, “Development and Performance of Resource Usage Service in WLCG”, *Conf. on IEEE Nuclear Science Symposium*, Oct. 2006. pp.603-606.

X. Chen and A. Khan, “Development of Multi-Grid Resource Usage Service in LCG”, *Conf. of International Symposium on Grid Computing (ISGC) 2007*, Mar. 26-29, 2007, Taiwan.

X. Chen, R. M. Piro, P. Canal, et. al, “Aggregate Usage Representation Version 1.0”, *OGF Usage Record working group*, Dec. 2006. Available online at: https://forge.gridfourm.org/projects/ur-wg/

X. Chen, “OGSA Resource Usage Service IDL WS-I Rendering”, *OGF Resource Usage Service working group*, Dec. 2007. Available online at: https://forge.ggf.org/sf/sfmain/do/go/artf6090?nav=1&selectedTab=attachments

X. Chen and A. Khan, “GRUS: An Extensive Solution to Resource Usage Service”, *Conf. on IEEE Nuclear Science Symposium*, Dresden, Germany, Oct. 2008.

S. Crouch, D. Fellows, X. “Experiences of Using Usage Record (UR) Version 1.0”, *OGF Usage Record Working Group*, Oct. 2009, Available online at: http://forge.gridforum.org/projects/ur-wg

X. Chen, A. Khan, G. Willis, and L. Gilbert, “Developing Resource Usage Service in WLCG” *IEEE Trans. on Nuclear Science,* Submitted on 25th June, 2010.

Chapter 1

Introduction

The Grid has been recognised as the next-generation distributed computing technology. The basic idea of the Grid is to virtualise heterogeneous resources, including computing power, data storage, application and instruments, across administrative domains as an integrated system. The emergence of Grid technologies is by no means a coincidence but driven by two main factors: supply and demand. On the one side, grand-challenge problems require large-scale collaborations and a great number of computer processing cycles. A typical example would be the Large Hadron Collider (LHC), a facility built to perform particle physics experiments in Geneva. Each experiment involves collaboration of over 3000 physicists from hundreds of world-wide institutions. It is also estimated that individual experiment will generate several PetaBytes of data annually. Thousands of physicists need access to, and analyse immense amounts of experimental data in near real time. On the other side, considerable computational and storage resources are distributed inside individual participant institute, and can potentially supply unprecedented processing and storage capacities over the Internet. The Grid middleware is therefore the bridge of the gap between application-level demands and distributed IT fabrics supplied by using state-of-art distributed computing technologies.

Compared to traditional distributed computing systems, a Grid system requires the assurance of Qualities of Service (QoS) at different levels, including security, performance, responsiveness, etc. In order to ensure system-level QoS, a Grid system need to analyse resource usage status, and take appropriate actions, such as resource reallocation, job migration, or blocking a suspicious user account, to ensure agreed QoS. The major task of Grid accounting service is to meter and log resource usage information of the underlying Grid environment. Accounting data can be also used for Grid economy by providing proofs for charging and billing. This thesis discusses the development of Grid accounting systems in a standard compatible manner to enable interoperability of heterogeneous accounting systems in such multi-Grid environment that consists of multiple Grid infrastructures managed by various middleware software stacks.

As an introduction, the content of this chapter is intended to establish the context of Grid computing and Grid accounting. Detailed technical issues and solutions are to be discussed in following chapters step by step.

1.1 Evolution

Since the birth of computing, performance has always been one of the leading factors driving the evolution of computing technologies. This section discusses historic progresses of computers and computing technologies that contributed to the emergence of Grid computing.

1.1.1 Computer Generations

As the timeline given in figure 1.1, the history of computer can be traced back to 1940s. The first-generation (1946-1953) computers were characterised by the use of vacuum tubes. A vacuum tube acts as a switch or amplifier by controlling electric currents. For example, the 5th of ten vacuum tubes can be switched on for representation of numeric five. The first electronic computer, Numerical Integrator And Computer (ENIAC)[1], was built at University of Pennsylvania in 1946 using vacuum tubes instead of mechanical switches of the Mark I. The ENIAC was capable of executing 5,000

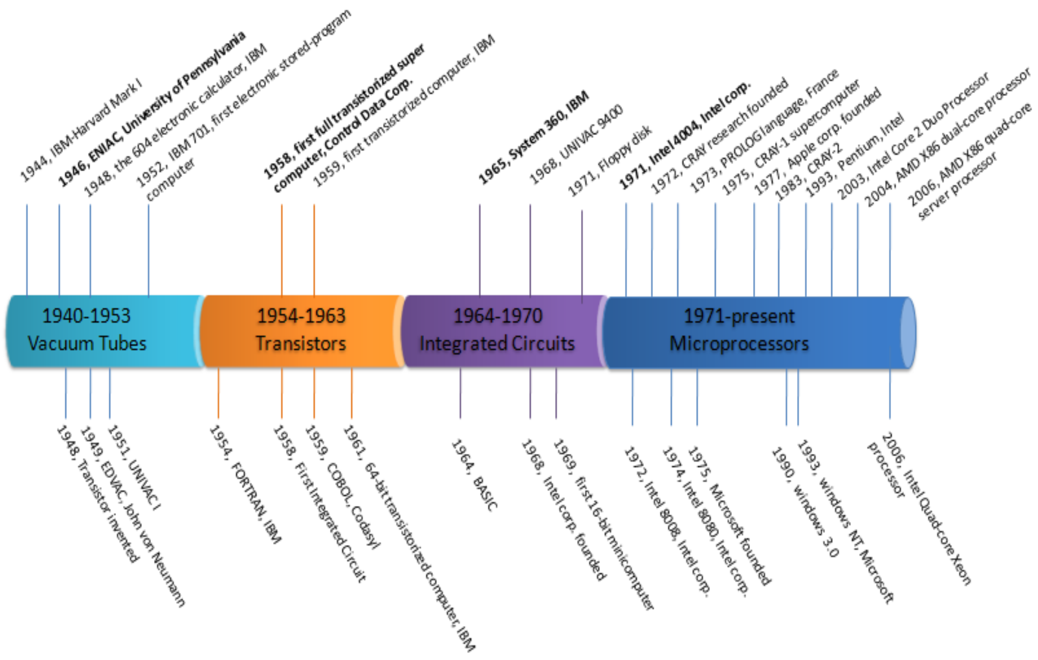

1944, IBM-Harvard Mark I
1946, ENIAC, University of Pennsylvania
1948, the 604 electronic calculator, IBM
1952, IBM 701, first electronic stored-program computer
1958, first full transistorized super computer, Control Data Corp.
1959, first transistorized computer, IBM
1965, System 360, IBM
1968, UNIVAC 9400
1971, Floppy disk
1971, Intel 4004, Intel corp.
1972, CRAY research founded
1973, PROLOG language, France
1975, CRAY-1 supercomputer
1977, Apple corp. founded
1983, CRAY-2
1993, Pentium, Intel
2003, Intel Core 2 Duo Processor
2004, AMD X86 dual-core processor
2006, AMD X86 quad-core server processor
1940-1953
Vacuum Tubes
1954-1963
Transistors
1964-1970
Integrated Circuits
1971-present
Microprocessors
1948, Transistor invented
1949, EDVAC, John von Neumann
1951, UNIVAC I
1954, FORTRAN, IBM
1958, First Integrated Circuit
1959, COBOL, Codasyl
1961, 64-bit transistorized computer, IBM
1964, BASIC
1968, Intel corp. founded
1969, first 16-bit minicomputer
1972, Intel 8008, Intel corp.
1974, Intel 8080, Intel corp.
1975, Microsoft founded
1990, windows 3.0
1993, windows NT, Microsoft
2006, Intel Quad-core Xeon processor

Figure 1.1: The timeline of computer evolution including selected events of each generation.

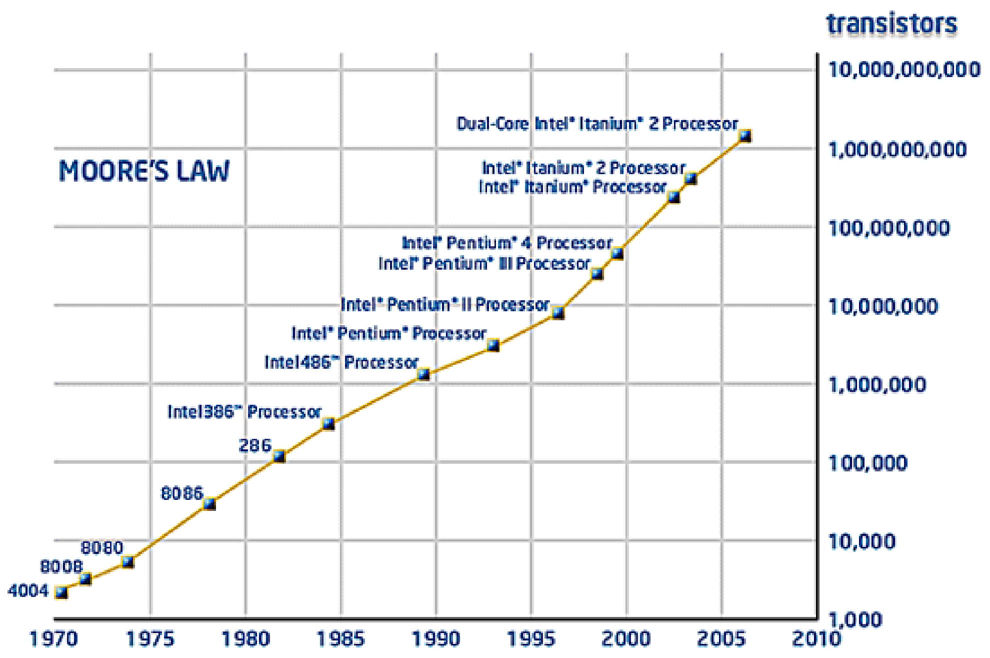


Figure 1.2: The Moore's Law predicated the number of transistors integrated in a single chip doubles very two years. From[2]

operations per second. Other vacuum-tube computers include Electronic Discrete Variable Automatic Computer (EDVAC)[3] and UNIversal Automatic Computer (UNIVAC)[4]. Considering thousands of integrated vacuum tubes that give off so much heat, these early computers were unreliable due to broken vacuum tubes.

Although the transistor was invented early in 1947, it was not widely used in computers until late 1950s. The replacement of vacuum tube with transistor makes computer smaller, faster and more reliable. The first full transistorized super computer was built at Control Data Corp. in 1958, indicating the beginning of "transistors era" as the second-generation computers. Programming on the second-generation computers moved from cryptic binary machine language to symbolic languages, so that programmers can code in high-level natural programming languages, such as early-version FORTRAN and COBOL.

The invention of the Integrated Circuit (IC) formed the basis for third-generation (1964-1970) computers. The size of computers became significantly smaller and faster by integrating in-cooperated transistors within a semiconductor chip. The first integrated circuitry computer, IBM 360, was built by IBM in 1965. It is capable of processing over 6,000 operations per second. In the meantime, advanced storage technologies contributed a new computer design with an internal memory. External storage devices, magnetic tapes and floppy disks, enable data input directly into the computers rather than using punch cards.

The development of Large-Scale Integration (LSI) and Very-Large-Scale Integration (VLSI) was the hallmark of the fourth-generation (1971-present) computers. A VLSI allows integration of millions of transistors into a single IC chip, and makes the fourth-generation computers smaller in size and faster in processing speed. Another revolutionary technology contributed in fourth-generation computers was the invention of microprocessor that incorporates almost all functions of a Central Processing Unit (CPU) into a single IC. A CPU or processor contains only one core, the part of the processor that actually processes an instruction at one time. In 1965, the co-founder, Gordon E. Moore, envisioned that the number of transistors on a chip would double every two years. The Moore's Law [5], as described in figure 1.2, was proposed based on empirical observations. The predication of the Moore's Law remains accurate so far and can be best demonstrated by the multi-core technologies. Since 1990s, Intel initiated and led the multi-core technology until present, by integrating multiple symmetric or identical cores within a single processing unit such that multiple instructions can be processed in parallel at same time. The multi-core technologies, such as Dual-Core and Quad-Core processors, have been widely used in modern commodity computers, making it possible to cope with complex problems across application domains by leverage parallel processing capacities of a single computer.

1.1.2 High Performance Computing

"It can't continue forever. The nature of exponentials is that you push them out and eventually disaster happens"

–Gordon Moore, April 13, 2005

According to Moore's Law, it is expected the number of transistors integrated within a single chip would reach over 15 billion, pushing computer engineering into the molecular and atomic era. Although relevant research[6][7][8][9] have been undertaken for decades, there is no guarantee that development of these advanced technologies will be applied to computer engineering in the coming decades. The effectiveness of Moore's law will eventually come to the ultimate limit in the next decade.

Computer engineers, however, never give up their ambition to pursue higher performance. The concept of High Performance Computing (HPC) was firstly suggested by Charles Babbage in the 19th century in order to solve the "Grand-Challenge" problems by employing multiple processing units or processors in parallel. Such "Grand-Challenge" problems as applied fluid dynamics, ecosystem simulation and weather forecasting are too complex to be solved in a reasonable amount of time using a single processor.

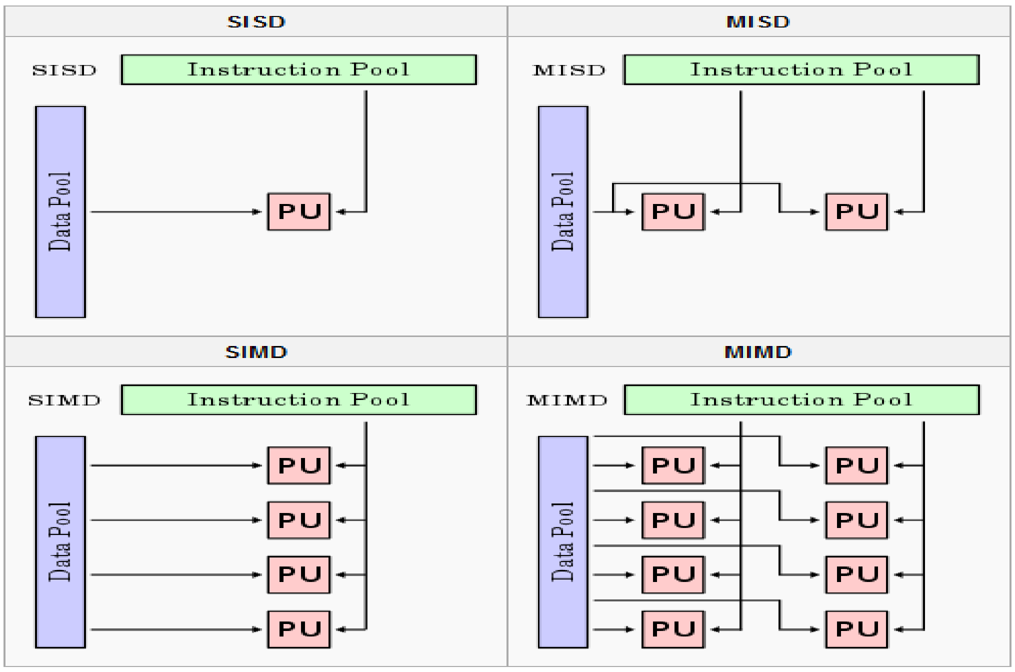


Figure 1.3: Flynn's Taxonomy classifies computer architecture into four types according to the number of instructions and data stream to be processed concurrently. From [10]

Flynn's Taxonomy

According to the Flynn's taxonomy [11] proposed by Michael Flynn in 1966, computer architecture is classified into four types (figure 1.3) based upon two dimensional factors: the number of concurrent instructions and the number of data streams operated concurrently. Traditional computer architecture falls into the Single-

Instruction-Single-Data (SISD) classification, involving a single processing unit that exploits no parallelism in either instruction or data stream. A Single-Instruction-Multiple-Data (SIMD) computer enables data parallelism by execution of the same instruction upon different data streams concurrently. Multiple-Instruction-Simple-Data (MISD) architecture is an uncommon architecture generally used mainly for fault-tolerance through agreed results of execution of different instructions set upon same data stream. Multiple-Instruction-Multiple-Data (MIMD) architecture employs multiple processors simultaneously executing different instructions on different data streams.

According to architectural relationship between processors and memories, the MIMD architecture can be further divided into two subtypes, Symmetric Multi-Processing (SMP) and Massive Parallel Processing (MPP). A SMP machine involves two or more identical processors connected via bus to access shared memory. Multiple processing units in a SMP computer therefore have access to all memory spaces with equal latency and bandwidth. In contrast, a Massive Parallel Processing (MPP) MIMD machine is equipped with a large number of processing units, normally over 100, each of which has access to its own physical memory or logically allocated memory spaces, therefore also known as Non-Uniform Memory Access (NUMA) [12] system.

Supercomputer

A supercomputer is a computer with multiple processing units and custom design based upon SIMD or MIMD architecture providing high performance processing capacity, approaching Tera-FLoating point Operation Per Second (TeraFLOPS). Vector or Array computers, formed the basis of most supercomputers throughout 1980s and into 1990s, applied SIMD architecture to execute mathematical calculations on vectorised data set simultaneously. Examples of Vector machines include the early CRAY X-MP [13], Maspar MP-1[14] and the Distributed Array Processor for ATM (ATM DAP) [15]. Modern supercomputers, as top ten supercomputers on the Top 500 list [16], are architected with a cluster of MIMD multiprocessors.

In order to exploit the high-performance of supercomputers, applications are required to be coded differently and divided into pieces that can be executed in parallel. There are parallel programming languages roughly categorised into two classes according to the communication patterns among processes based on underlying memory architecture. For tightly coordinated shared-memory machines, programming languages or libraries, such as OpenMP [17] and Portable Operating System Interface for Unix (POSIX) threads, are mainly used for manipulation and synchronization of shared memory variables. For loosely-coupled memory architecture, communication among multiple processes is realized through message passing APIs. The most commonly used APIs include Messaging Passing Interface (MPI)[18] and Parallel Virtual Machine (PVM)[19].

Cluster Computing

Specialised Supercomputers, built at huge cost to deliver magnitude greater floating point performance, however have been perceived to be too hard, too expensive and of too narrow interests. Rather than developing specialty-class supercomputer architectures, commodity clusters have rapidly grown as HPC systems by harnessing commercial-off-the-shelf (COTS) computer nodes. A commodity cluster comprises computer nodes interconnected by Local Area Network (LAN) within a local administrative domain. It allows flexible system configuration in terms of number of nodes, number of processors and memory capacity. Since 1990s, an increasing number of environments had emerged ranging from commercial SMP servers to self-assembled PC clusters, such as Beowulf [20] cluster.

1.1.3 Distributed Computing

The Internet

Prior to the Internet, communications between computers were prevalently based on mainframe method, simply allowing communications among terminals via local physical connections. In order to enabling interconnection of computers from different local networks, the idea of Packet Switching was proposed by Leonard Kleinrock from Stanford University in the 1960s. Following the successful demonstration of packet switching work at APRANET in 1969, the first packet-switching standard, ITU X.25 [21], was released by the International Telecommunication Union (ITU) based on the concept of virtual circuit. The emergence of the TCP/IP protocol in 1978 enabled unifying different network protocols by using a common inter-network protocol. The Internet was then officially defined as a global system of interconnected computer networks that interchange data by packet switching using the standard TCP/IP protocol. The Internet carries various information, which however could not be shared in a uniform format until the introduction of World Wide Web, or the Web, by Tim Berners-Lee in 1989 [22]. The Web is a huge set of interlinked documents accessible through Hypertext Transfer Protocol (HTTP) [23].

Middleware

The emergence of the Internet and the Web contributed to a large-scale computing model that aims to communicate and coordinate software components on interconnected computers. This type of computing model is defined as distributed computing, allowing a program to be split up into parts that run on multiple computers interconnected under a loosely controlled regime. Two typical paradigms of distributed computing are the client-server (C/S) model and peer-to-peer (P2P) model. The C/S model is the most cited model used in distributed computing with server processes carrying out activities and client processes initializing activities. In P2P model, all distributed processes plays similar roles without clear distinction between client and server. These distributed processes act cooperatively as peers take both the role of client and server depending on initializing or provision of activities.

In order to enable distributed computing in a platform-independent manner, an additional software layer, known as middleware, is required to mask heterogeneity of underlying platform. Middleware-oriented solutions provide a high-level abstraction over underlying networking, hardware, and operating systems. The foundation for communication of distributed parts is the Remote Procedure Call (RPC)[24], which was superseded by the introduction of the object-oriented programming model middleware, such as Java Remote Method Invocation (RMI)[25], Common Object Request Broker Architecture (CORBA)[26] and Distributed Component Object Model (DCOM)[27].

Web Service

Middleware-oriented solutions, however, are normally developed in a language-specific pattern and lack of interoperability. A Java RMI client process, for example, cannot interact with DCOM server processes. Besides, distributed applications relying on middleware are typically used for intranet usage and hard for communication across firewalls. Web service overcomes limits of traditional middleware solution by introducing a stack of Web-oriented standards based upon eXtensible Markup Language (XML), which enable platform- and language-neutral communication via HTTP.

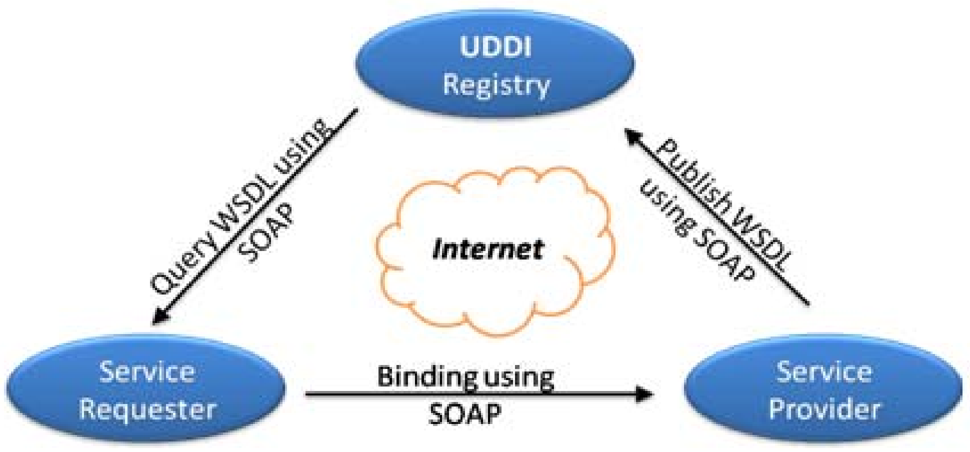


Figure 1.4: Service-Oriented Architecture and Internal Interactions through standard Web service communication protocols

As shown in figure 1.4, abstract Service-Oriented Architecture (SOA) is composed of three major components, service requester, service provider and registry, which communicate with each other through the standard communication protocol, Simple Object Access Protocol (SOAP)[28]. The SOAP defines XML-formatted encoding rules for exchanging structured information between service requesters and providers, as well as binding rules for data transferring upon other networking and application protocols, most notably RPC and HTTP. The registry component maintains a repository of registered services and acts as a coordinator between service requestors and providers. The registry itself can be implemented as a Web service endpoint by exhibiting a set of well-defined interfaces for service query and registration. The Universal Description, Discovery and Integration (UDDI)[29] is such a service definition that specifies standard registration-related interfaces and messaging format. A Web service is required to be self-describable using the Web Service Description Language (WSDL)[30] before getting registered. A WSDL file describes inter-operate contracts of a particular service, such as allowed operations, messaging formats, and enabled networking bindings.

1.1.4 The Grid

"A computational Grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities."[31]

Ian Foster and Carl Kesselman, 1998

The evolution of the Grid was driven primarily by the ever-growing demands on computational power. The Grid was defined as a computational Grid that aims at providing HPC facilities in a cost-effective manner through interconnection of existing computational resources. There are two prerequisites for Grid deployment: reasonable communication latency and tremendous computational resources. As with Gilder's Law, the growth of network bandwidth had been observed faster than computer power at least as much as three times. This law indicates the communication bandwidth via internetwork doubles every six months, if computer power doubles every eighteen months. It has been observed that the bandwidth of Internet backbone had been updated continuously during 1980s and 1990s, from 56 kilobyte/sec to 45 megabyte/sec. In addition, the state-of-art 10G Ethernet technology provides fastest communication network reaching 10

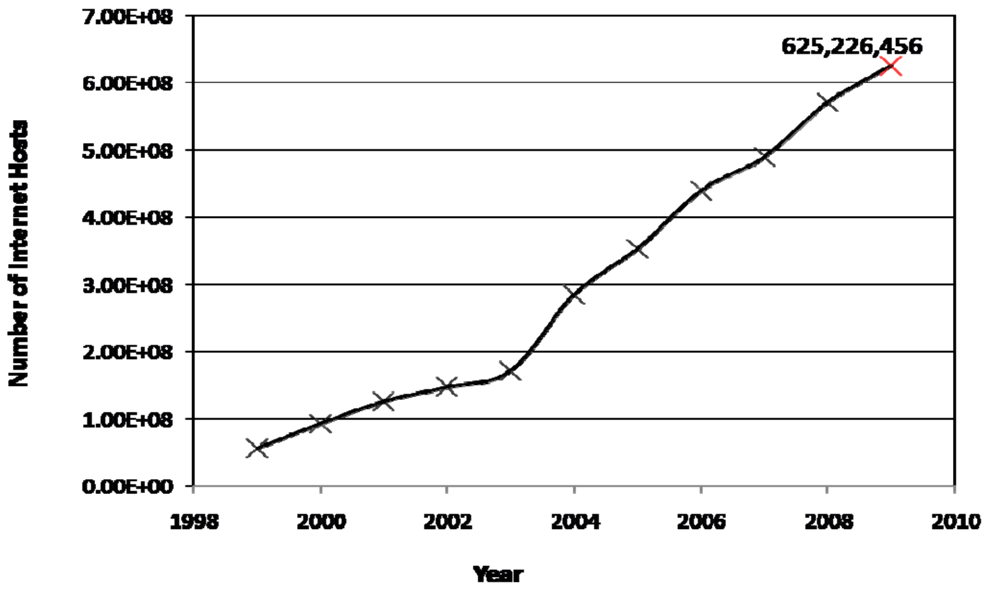


Figure 1.5: Internet Host Statistics. From[32]

gigabyte/sec. On the other hand, the number of computer hosts (as fig. 1.5) connected to the Internet have dramatically increased to over 625 million up to Jan. 2009. These computational resources are becoming potentially large-scale computational resource pool, which provides unprecedented processing power than ever, either through dedicated gigabyte/sec Ethernet for computation-intensive applications or through Wide Area Network (WAN) in pursuit of global collaborations.

> "The real and specific problem that underlies the Grid concept is coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations."[33]
>
> *Ian Foster, Carl Kesselman, Steven Tuecke, 2001*

The concept of Grid was refined in 2001 and highlights advanced features as follows. Rather than computation oriented only, some experimental science projects, such as Particle Physics and Earthquake simulation, requires instrumental resources to be shared, including sensors, detectors, etc. These experimental instruments accompany with compute, storage and others are collectively known as Grid resources. A Grid system is therefore required to address heterogeneity of underlying resource though a set of open protocols and interfaces that address fundamental issues as authentication, authorisation, resource access, discovery, etc. Considering Grid resources may be shared from different administrative domains, it is important for a Grid system to ensure shared resources not be subject to localised control, but are subject to the control at Virtual Organisation (VO) level, which defines a set of resource-sharing rules and conditions of a dynamic of individual or institutions.

However, the Grid computing is only one branch of the evolving distributed computing technologies. In the meanwhile of the evolution of the Grid computing, we have witnessed many other distributed computing technologies, which were driven by different problem scopes, although some underlying technologies and issues are overlapped. The following lists some example distributed computing models and highlights their differences or relations to Grid computing.

Volunteer Computing

Volunteer computing is a type of distributed computing model where computing resources (i.e. processing powers and storage capacities) are provided by one or more computer owners. These resources can be harnessed for one specific application or various applications through a general-purpose middleware solution. The basic idea behind volunteering computing is to use spare processing or storage capacities of computing resources connected to the Internet. In order to participate in a volunteer computing application, computer owners are need to trust the application and agree to install a piece of client-side software, normally lightweight and only active when computer volunteers are free or underutilised. As the SETI@home project [189], a volunteer computing project using internet-connected computers to analyse radio signals and search for signs of extra terrestrial intelligence.

Given its volunteer nature, the volunteering computing differentiates from the Grid computing in following aspects:

- A Grid application owns computing resources shared by one or more organisations, while a volunteer computing application does not has ownership of participating computing resources,
- Grid computing requires delivering QoS at different levels, such as availability, security, etc. These QoS are hard to be ensured in volunteering computing, given the fact the ad-hoc nature of volunteer computing resources.
- Grid computing middleware are general-purpose and provide well-defined APIs for resource sharing across application domains, while volunteering computing middleware are designed for a specific application or a specific application domain.

Autonomic Computing

Autonomic computing was initiated by IBM in 2001, which aimed at developing an intelligent computing system that is capable of self management, and reducing the complexity of system management particularly for large-scale computing environments. An autonomic system is able to monitor, make decisions, and adjust underlying system environment on behalf of system administrators in order to fulfill pre-defined Quality of Service (QoS). IBM defines four core technical features [190] that enable the ability of an autonomic system to adapt to change in accordance with business policies and objectives. These features include:

- Self-optimisation: the ability to automatic monitoring and control of system resources to ensure the optimal functioning with respect to the defined requirements according to dynamic changes;
- Self-healing: the ability to recover from system errors without human intervention;
- Self-configuration: the ability to automatic configuring system components to adopt to changes in the system;
- Self-protecting: the ability to proactively anticipation and protection from arbitrary intrusions

A Grid computing system can make use of autonomic computing facilities to enhance self-manageability and ensure QoS attainments at different service layers. Further details of self-management of Grid computing are given in section 2.3.6.

Utility Computing

The utility computing is analogy to traditional public utility with a metering and paying service running at backend. The basic idea of the utility computing requires low or no initial investment on hardware, while providing pay-and-run facilities through virtualization of computational and storage resources at backend. Utility computing was firstly proposed in the 1960s by John McCarthy, who envisioned that future organisations would simply plug in to a computing grid for computational resources rather than providing their own computing powers, just like connecting to an electrical grid, and paying based on what is used.

It is worth noting that the utility computing is not a specific computing technologies but a vision of next-generation computing. The Grid computing is one of such enabling technologies that enable the vision the utility computing.

Cloud computing

Cloud computing is an emerging distributed computing paradigm that aims at providing an elastic, self-service and pay-per-usage computing facilities. As an emerging concept, the concept of cloud along with its enabling technologies is still confusing many.

According to the definition from National Institute of Standards and Technology (NIST), the Cloud computing is defined as "*a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources, such as networks, servers, storage, application and services, which can be rapidly provisioned and released with minimal management efforts or service provider interaction*"[34].

The recent published review report [196] uses an analogue to Web 2.0 and defines cloud computing as a business model other than a new computing technology. Cloud computing facilities can theoretically deliver any type of computing capacities to end users including the Grid computing. Technically speaking, cloud computing and grid computing differs and relates to each other in many aspects:

- The cloud computing resources are predominately provided by a single organisation and located in a large-scale data centre, in contrast to Grid computing resources shared across multiple, normally geographically distributed organisations.
- The key enabling technology of cloud computing is the virtualisation technology to maximise resource utilisation, while Grid computing is more concerned about load balancing among distributed computing resources.
- Grid computing and cloud computing share high-level technical challenges, such as resource management, data management, security, QoS management, etc.
- By using cloud computing facilities, the management tasks are delegated to the service providers' side, i.e. end users does not need to worry about resource management. The current Grid implementations still require end users to have certain knowledge of the execution environment for capacity planning, resource management and security.
- Cloud computing can potentially provide Grid facilities or using Grid computing technologies at backend.

1.2 e-Science Grid

"e-Science is about global collaboration in key areas of science and the next generation of infrastructure that will enable it."[35]

Dr John Taylor, Director General of Research Councils, 2000

"The term ‘e-Science’ denotes the systematic development of research methods that exploits advanced computing thinking. Such methods enable new research by giving researches access to resources held on widely-dispersed computers as though they were on their own desktops. The resources can include data collections, very large-scale computing resources, scientific instruments and high performance visualization.” [36]

Prof. Malcolm Atkinson, e-Science Envoy

The evolution of computing and networking technology is leading to the revolution in the conduct of scientific research. Scientists from different disciplines started using computing technologies, electronic data storage, and networking to replace or extend traditional efforts. Classic scientific researches, both theoretical and experimental, are using computer-aided simulation to explore new possibilities and achieve new precisions. HPC computers or clusters have been widely deployed at institutions enabling the speedup of simulation and visualization. A group of scientists from the same research fields meet and collaborate online without time and geographic limits.

During the past decades, scientific research had also experienced changes affected by the “e”, abbreviation of the “electronic”, such as e-Social, e-Publishing, and e-Conference. These “e-” technologies somehow did not enable fundamental, but profound, transformation of research methodologies in science, until the advocating of the “e-Science”. The term was coined by John Taylor, the director general of the office of science and technology in the UK, at the time of announcement of a major funding program, the UK e-Science program in 2001. The definition of e-Science was moderately refined by Prof. Malcolm Atkinson with clarified objectives. With e-Science, researchers are expected to have a set of value-added tools, software, and applications to access world-wide experimental results, to access global computing resources for real-time simulation and visualization, and collaboration on a grand project.

All these visions of future sciences depend on an e-infrastructure that provides tools, software and applications enabling coordinated problem solving. During the past five years, there were over 100 projects funded to UK e-Science program for developing an e-infrastructure, including SuperJANET project for constructing high bandwidth academic network, National Grid Service (NGS) project facilitating research activities to access distributed computational and data resources throughout the UK, Open Middleware Infrastructure Institute UK (OMII-UK) providing open source software to e-Research communities, etc. In Europe, the Enabling Grid for E-sciencE (EGEE) project was founded by European Commission in 2004 aiming at providing a Grid-enabled e-Infrastructure for various scientific domains, including earth science, high energy physics, bioinformatics and astrophysics.

1.3 World-wide LHC Computing Grid

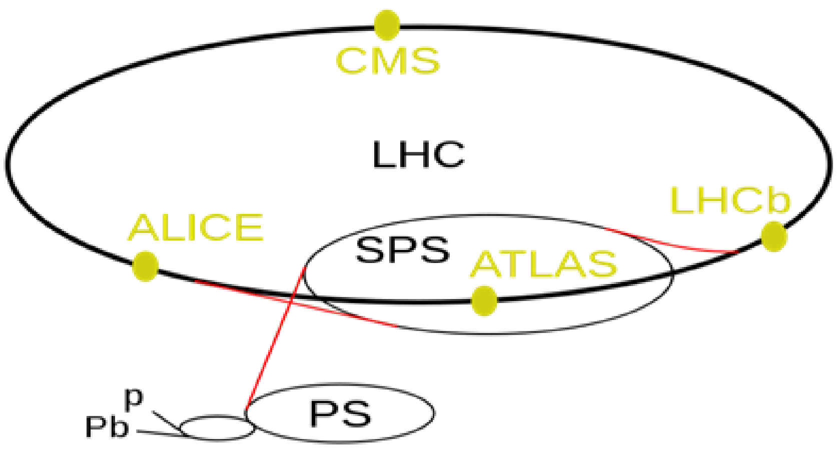


Figure 1.6: LHC layout, four main experimental detectors chained by 27km ring accelerating tunnel. From[41]

The Large Hadron Collider (LHC) is the world’s largest particle accelerator built by the European Organisation for Nuclear Research (CERN). It is intended to test various predications of high-energy physics through collision of opposite particle beams. Four main detectors have been constructed at the LHC to record events triggered by collisions. Two large and general-purpose detectors, the A Toroidal LHC ApparatuS (ATLAS)[37] and Compact Muon Solenoid (CMS)[38], are used to search for signs of new particles clues to the nature of dark matter. Other two medium detectors, LHC-beauty (LHCb)[39] and A Large Ion Collider Experiment (ALICE)[40], focus on investigations of events just after the Big Bang. As LHC layout given in figure 1.6, these detectors are chained and located at four collision points of main circular ring of LHC. Protons are firstly accelerated by linear accelerator to 50 Megaelectron Volt (MeV) before entering into three successive synchrotrons, including the Proton Synchrotron (PS) and Super Proton Synchrotron (SPS). Protons ultimately enter into the LHC main ring, where collisions take place 40 million times per second.

It is estimated there will be huge amount of data to be generated by LHC experiments, approximately 15 PetaBytes a year. These data is intended to be analysed by thousands of scientists around the world. Based on an initial survey[42] on anticipated computing requirements for LHC experiments submitted in early 2001, CERN approved and launched the Worldwide LHC Computing Grid (WLCG) project. One of the objectives of the WLCG project is to develop a Grid infrastructure upon that distributed computing and data storage resources from 140 computer centres in 34 countries. These distributed computer centres are organised into three tiers. When collision triggered, event data are collected by experiment-specific trigger and acquisition systems [43][44][45][46]. Event data are then filtered by local computer farms so that only interesting events are kept into local persistent storage. Four experiments send their filtered raw data simultaneously to the CERN computer centre, so-called Tier-0 centre. When raw data arrives at Tier-0 centre, they are processed initially and backed up on tape at CERN. Subsets or all raw data are then sent out globally to eleven large Tier-1 computer centres that are interconnected by the general-purpose research network with dedicated 10 Gbit/s links. There are more than 150 Tier-2 centres, mainly university and research institutes, which allow physicists to perform analysis and simulations. The WLCG is supported by three major Grid infrastructure projects, Open Science Grid project of US, EGEE project, and Nordic Data Grid Facility (NDGF) project. Each project supplies custom, while interoperable, Grid middleware solution, through open standards and interfaces.

1.4 Grid Accounting

For large-scale complex system as the Grid, resource usage is required to be accurately accounted. Resource usage information is important in the sense of Grid system administration and policy enforcement. In scientific Grid environment, resources are predominately shared for one or more non-profitable research projects. Each project has fixed resource quotas, computational cycles and storage spaces for instance. A Grid system is committed to prevent overexploitation of resources by checking historic resource usage of individual or project against allocated quotas. Resource usage information enriches the understanding of resource utilisation in a Grid system, so that system administrators can determine how to reallocate resource for better system performance, maximising resource utilisation. Resource usage information also helps discover and track abuses or configuration issues of a Grid infrastructure. Commercial Grid allows users to access resources on “pay-per-use” basis. Resource usage information therefore is used as proofs for charging. The Grid accounting is such a service that aims at providing a consistent Grid-wide view of resource usage. Many production Grids have accounting systems deployed for various purposes. The accounting system in UK e-Science Grid, the National Grid Service (NGS) for example, is being developed mainly for resource usage monitoring against individual users. Accounting systems in EGEE/WLCG projects are mainly for statistic usage reporting for Virtual Organisations (VOs) and participating sites.

1.5 Summary

This chapter sets the scene for following detailed discussion. The chapter discussed Grid computing and Grid accounting at high level, including historic driver factors that enables the emergence of Grid computing; its impacts on revolutions of scientific research patterns by giving two example usage scenarios of ‘e-Science’ projects; and a brief introduction of Grid accounting. In following chapters, more technical details are to be discussed. The rest of this thesis is organised as follows. Chapter 2 concentrates on technical aspects of the Grid and reviews of a selection of middleware solutions implemented by production Grid projects. Technical reviews of Grid accounting is presented in Chapter 3. In Chapter 4, an accounting system prototype is presented and shows how a standard-compatible solution contributes to a consistent way to share accounting data in such multi-Grid environment as WLCG with different accounting systems deployed. Chapter 5 demonstrates the implementation of a generic Grid accounting middleware that is used as a toolkit to ease the migration of existing accounting systems to be standards compatible. Finally conclusion and future work are given at chapter 6 as the ending of the whole thesis.

Chapter 2

The Grid

A Grid system integrates heterogeneous and distributed resources across administrative domains a virtual system. Since 1990s, extensive efforts have been put on development Grid middleware and software for diverse research projects. Early Grid middleware solutions were built upon existing Internet protocols and aimed at providing a Grid infrastructure for specific projects. These early adoptions to the Grid are too implementation-specific to be used by others. Based on these initial efforts, the Grid had received increasing attentions and evolved as a standard distributed computing paradigms. In 2001, the first architectural standard, Open Grid Service Architecture (OGSA), was released and formed the basis of constructing interoperable Grid systems. The OGSA standard identifies a set of key functional components of a Grid system based on emerging Web service architecture. Since then, a great number of Grid projects were founded to develop OGSA-compatible middleware and software tools. These production Grids are serving thousands of scientific research projects around the world.

The success of the Grid to date comes from a combination of factors, including early implementation experiences, the emergence of clear architectural principles, standardisations, de factor standard software, etc. This chapter reviews and discusses these factors that combined to make the Grid possible.

2.1 Concept

For a long time, the term Grid was used for a computing Grid that provides unprecedented computational capacities by harnessing inter-connected computers. Based on increasing use cases gathered from both scientific and business applications using Grid technologies, the concept of Grid has been refined as a distributed computing paradigm with following essences [33]:

- *"coordinated resource sharing that are not subject to central control"*
- *"using open, standard, general-purpose protocols and interfaces"*
- *"delivering nontrivial qualities of service (QoS)"*

One of the key objectives of constructing a Grid system is to enable seamless resource sharing across administrative domains. These heterogeneous resources are coordinated to achieve an application-oriented goal in a trustful and controlled manner governed by a set of sharing rules, which are not subject to any specific administrative domain. Such sharing rules include security, user membership, payment, and application-specific policies. A set of individuals and resources governed by same sharing rules forms the so-called Virtual Organisation (VO).

A Grid system is built upon multi-purpose protocols and interfaces that address fundamental issues relating to resource access, resource management, resource introspection, authorisation, etc. A piece of software that implements these protocols and interfaces is known as Grid middleware. It is important these protocols and interfaces are open and standard such that Grid applications can be developed in a consistent manner and migrated from one Grid middleware to the other.

Finally Grid resources are used in a coordinated pattern in order to accommodate requirements for diverse usage modes and deliver various non-functional characteristics, known as Qualities of Service (QoS), such as advanced resource reservation, security semantics, system throughout, responsiveness, etc.

2.2 Architecture

In order to identify general requirements on constructing a Grid system, a layered architecture (figure 2.1) is defined following principles of “hourglass model”. Each layer abstracts a set of core components and protocols, through which actions of high-level applications can be mediated and mapped onto underlying technologies of resources to be shared.

2.2.1 Fabric

The fabric layer, as the base of “hourglass”, provides resources to be shared in a Grid environment. These resources may be either physical entities or local entities, such as distributed file system, computer pool or database systems, which involve internal protocols defined by fabric components and deliver resource-specific QoS. Fabric resources that intended to be accessible through Grid protocols must supply two mandatory mechanisms: introspection and management mechanisms. Introspection mechanisms allows discovery of underlying resource structure, state and capability, while management mechanisms provide control over delivered QoS.

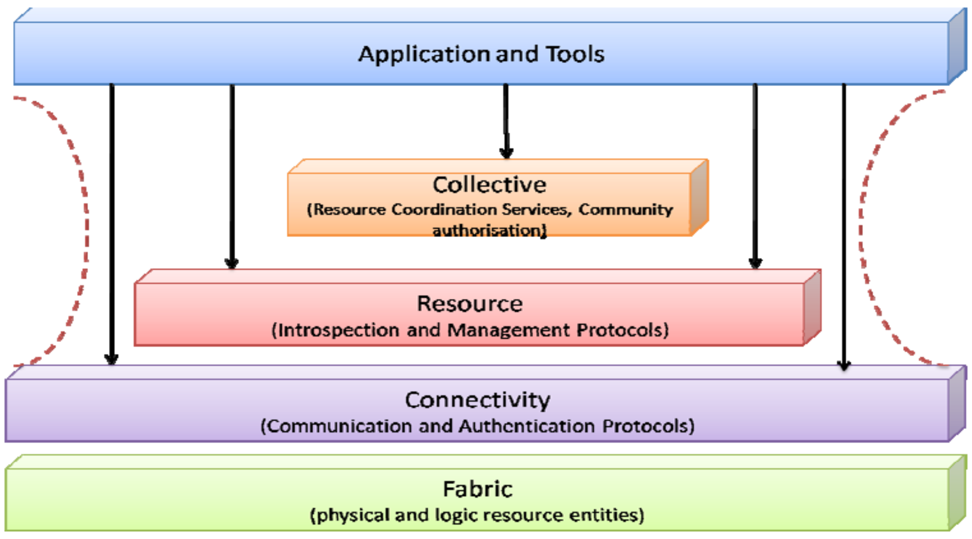

Application and Tools
Collective
(Resource Coordination Services, Community authorisation)
Resource
(Introspection and Management Protocols)
Connectivity
(Communication and Authentication Protocols)
Fabric
(physical and logic resource entities)

Figure 2.1: “Hourglass” architecture identifies requirements on definition of Grid protocols at each layer. From [47]

2.2.2 Connectivity

The connectivity layer defines a set of core communication protocols and authentication protocols required for Grid-specific transactions. Communication protocols are used to transport and route messages among fabric-layer resources involved within a Grid transaction. It is common to assume that these communication protocols are based upon, but not limited to, existing Internet-layered protocols, such as TCP/IP and other application-layer protocols.

Authentication protocols at connectivity layer establish a binary trustful link between communication endpoints by verifying the identity of user and resources. Although there are many security standards built upon Internet protocol suite, they do not satisfy all security problems in a Grid environment. Participants in a Grid environment often need to coordinate multiple resources to accomplish a complex task. By using existing Internet authentication protocols, individual user is required to be authenticated on per resource access basis. It is necessary to have a single sign-on mechanism that ensures user-transparent access to multiple resources coordinated for a single task. Besides, a user may endow a program with ability to execute on behalf the user. A user, for example, submits a job request to an execution service, which need to transfer an input file for the job execution from a remote storage resource. In this case, the remote storage access must be authenticated by verifying job requestor’s identity rather than execution service’s identity. This process is known as delegation.

2.2.3 Resource

Having defined connectivity-layer protocols, users can communicate underlying shared resource in a secure way. As discussed in section 2.1.1, fabric resources to be shared in a Grid environment must provide introspection and management mechanisms. However, these mechanisms are implemented in a resource-specific manner. The role of resource layer is therefore to abstract a set of common protocols that capture the fundamental mechanisms of sharing across many different resource types. Implementations on resource-layer protocols are supposed to call fabric resource functions to access and control local resources.

There are two primary classes of protocols as defined at resource layer: information protocols and management protocols. Information protocols define a set of common interfaces that interrogate local resource introspection mechanisms to obtain information about resource configuration, state, current load, etc. A set of common management protocols are used to negotiate resource access, specify runtime requirements, initiate operations, monitor execution status, and account resource usages. Definition of management protocols should be limited to a small and focused set, which apply to at least a range of resources that share common management requirements therefore forming the bottle neck of the hourglass model. Protocols defined within resource layer differentiate from those of collective layer in that resource-layer protocols target at an individual resource without concerning about coordinated actions across multiple resources.

2.2.4 Collective

While the connectivity and resource layers focus on low-level protocols for introspection and management of a single resource, the collective layer provides protocols and shared services at the Virtual Organisation (VO) level enabling coordinated resource sharing in a Grid environment. Typical collective services include:

- The community authorisation service that maintains and enforces security policies of one or more VOs.
- The directory service that allows VO members to discover the existence and properties of VO resources;
- Resource allocation and brokering service that allocates VO-member requests to one or more appropriate VO resources;
- Data replication service that maintains and manages copies of data among multiple VO storage resources;
- Community accounting service metering, gathering and provisioning VO resource usage information;
- Community monitoring service reporting real-time status of VO resources, mainly for detection of resource failure, intrusion, overload, etc;
- And community economic services that realises Grid economic through pricing and charging VO members according to actual resource usage.

2.2.5 Application

The final layer, application layer, comprises applications that operate within a VO environment. Development of applications may invocate well-defined low layer protocols or APIs. Alternatively, applications may develop sophisticated application-specific protocols and APIs.

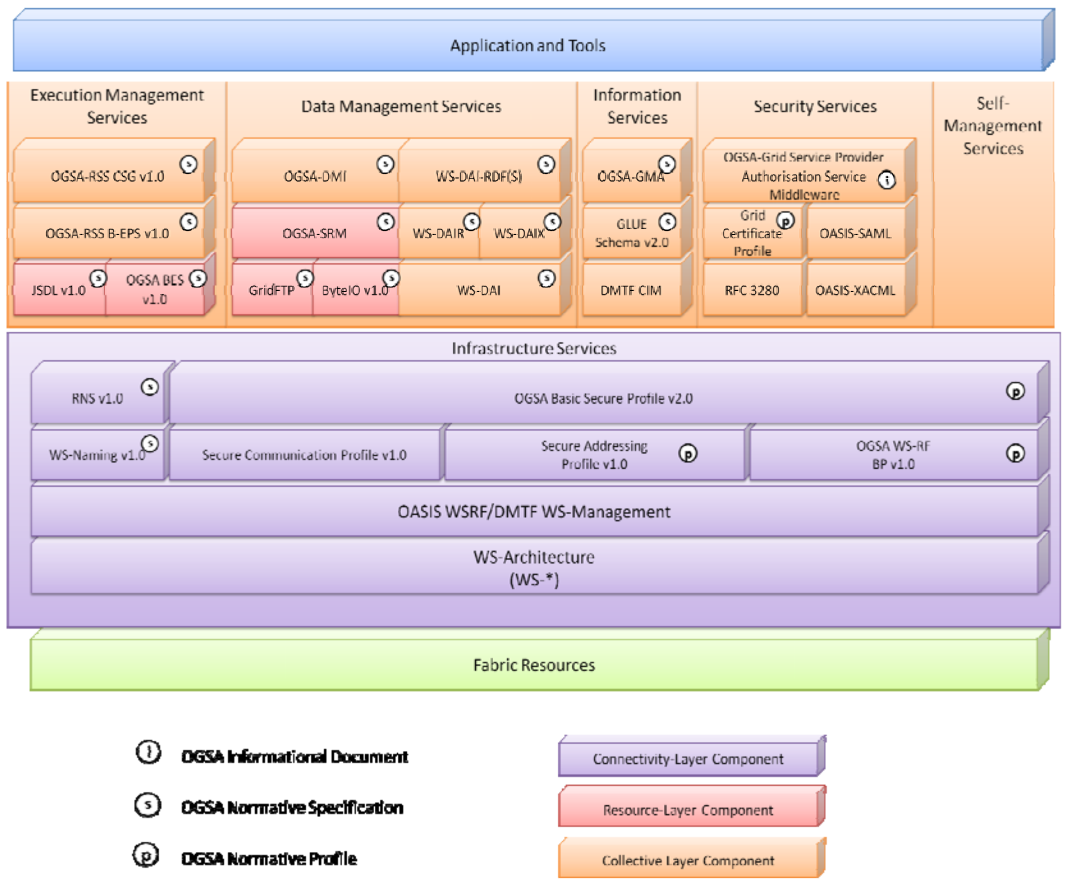


Figure 2.2: OGSA standard stacks and relationships to layered architecture

2.3 Standards

Open Grid Service Architecture is the standard that provides a high-level definition of core capabilities required to support Grid systems and applications. As figure 2.2, these capabilities include execution management, data service, resource management, security, information, and self management so that diverse components can be discovered, managed and integrated as a virtualised system. The OGSA standard [48] was proposed 2001 by the Global Grid Forum, GGF, which merged with Enterprise Grid Alliance (EGA) and formed the current Open Grid Forum (OGF). Thousands of individuals from over 400 organisations in more than 50 countries are currently active in different domain-expert OGF working groups and providing inputs to fulfil OGSA functionalities. There are two types of document inputs being produced by OGF working groups in order to maintain coherence around OGSA and Grid-related standards. The informational documents provide use cases, guidelines and information about OGSA architecture process. OGSA specifications and profiles are a collection of normative documents that define technical details on functional interfaces and protocols as well as their usage to ensure interoperability. The following content of this section discusses details of emerging OGSA specifications and profiles in the context OGSA.

2.3.1 Infrastructure Services

The main goal of infrastructure services is to provide coherent and integrated components that collectively address Grid requirements as demonstrated in section 2.2. A primary assumption is that OGSA systems and applications are built upon the Web Service Architecture (WSA) [49] and aligned with emerging Web-service technical specification in order to ensure interoperability through standard Web service messaging framework (i.e. SOAP)[28] and normative service description (i.e. WSDL)[30]. However, it is clear that currently defined Web service standards are not sufficient to meet all Grid requirements.

Basic Manageability Model

A Grid system requires resources to be shared in a manageable manner. One of the key objectives of OGSA infrastructure services is therefore to provide a basic manageability model that forms the basis for both resource management and management of OGSA environment. The basic manageability model at infrastructure level abstracts core manageability interfaces that are common to all resource/services implementing OGSA capabilities.

In early 2002 OGF proposed the Open Grid Service Infrastructure (OGSI) [193] specification that extends Web service capabilities and introduces the idea of "stateful" Web services, particularly concerned with creating, addressing, managing the lifetime of "stateful" Grid services and notification of service state changes. The OGSI specification is then refactored into a framework of Web service standards in 2004, in particular the family of Web Service Resource Framework (WSRF) [50] and Web Service Notification (WSN) [51], given the fact that the OGSI specification tried to integrate a number of independently reusable Web service functionalities into one specification. These specifications were defined to address specific problems and exploited other Web service standards, the Web Service Addressing (WS-Addressing) [57] for example. The collection of WSRF and WSN standards were originally proposed by OGF and then accepted by Organisation of Advanced Standards for the Information Society (OASIS) as the basis of Web Service Distributed Management (WSDM) [194] standards. In 2006, OGF further proposed a normative profile specification, the OGSA WSRF Basic Profile (WSRF-BP) [53], which aims at addressing interoperability issues of using WSRF specifications for distributed Grid resource management in the context of OGSA.

It is however worth noting that the WSDM specifications received increasing controversial debates mainly because of its Grid nature and incompatibility to WS-* mainstreams. In 2005, a competing specification, the Web Service Management (WS-Management) [52] was proposed by Distributed Management Task Force (DMTF). This specification is defined based on three main WS-* standards, including Web Service Transfer (WS-Transfer) [59], Web Service Enumeration (WS-Enum) [60], and Web Service Eventing (WS-Eventing) [61]. As shown in Table 2.1, these specifications provide functional counterparts of those defined in WSRF and WSN specifications.

In order to enable interoperability of separately developed Grid resources, a future convergence was planned in 2006 to converge WSDM and WS-Management specifications. As shown in Figure 3.2, the plan is to use WS-* standards as basis while defining extensions to support features that defined in the WSRF and WSN specifications, and eventually contribute to the convergence of WSDM and WS-Management specifications. Given the fact that future convergence is more WS-management oriented and based on its three underlying Web service standards, the development of Grid accounting solutions is based on WS-management framework rather than WSRF framework.

Table 2-1: Distributed Web Service Management (WS-RF vs. WS-Management)

Function	WSDM	WS-Management
State Representation	WS-Resource Properties	XML
State Lifecycle Management	WS-Resource Lifetime	WS-Transfer
Collection	WS-Service Group	WS-Enumeration
State Transition Notification	WS-Notification	WS-Eventing
Addressing	WS-Addressing	WS-Addressing
Fault Handling	WS-Base Faults [58]	SOAP Fault [28]

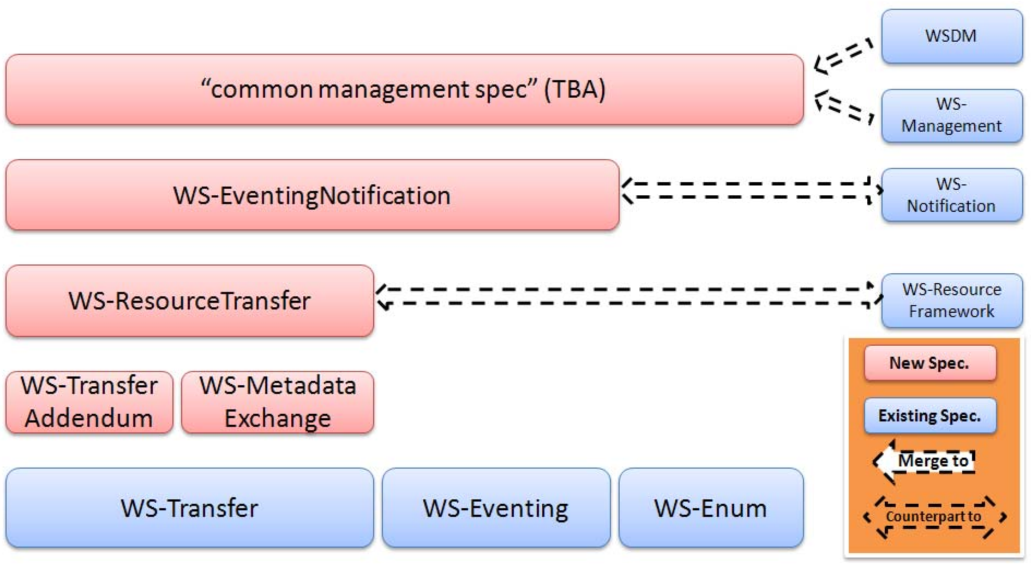


Figure 2.3: Roadmap of convergence of WSDM and WS-Management stacks

Naming

Resources in OGSA environment are represented as services, which are instantiated on demand and assigned a global unique address. The Endpoint Reference (EPR) model defined in the Web Service Addressing (WS-Addressing) specification [57] is used as the architectural construct for an address in OGSA. These addressable EPRs constitute a complex runtime environment of a Grid system. In order to simplify development high-level applications that utilise underlying complex environment, a three-level naming scheme of traditional distributed systems is employed in OGSA. Every named entity is associated with multiple user-defined names, a global unique abstract name, and one or more addresses. Two specifications, the OGSA-Resource Namespace Service (OGSA-RNS)[62] and Web Service Naming (WS-Naming) profile[63], defines standard protocols for resolving and rebinding of a user-defined name to an address by extending the endpoint reference model as defined in Web Service Addressing (WS-Addressing) specification [57].

Security

Another important issue to be solved at infrastructure level is the secure access to shared resources across different administrative domains. Considering there might be different security mechanisms adopted at classic organisations to accommodate specific security requirements, security at OGSA infrastructural layer is therefore required to ensure interoperability among domain-specific security mechanisms. Interoperability can be achieved at two levels ensuring authenticated and confidential communications. Transport Layer Security (TLS) is the commonly used security protocol in distributed computing by providing endpoint authentication and communication confidentiality. Emerging Web service security specifications offers advanced features and addresses secure communication at message level. At message level, authentication and trust relationship can be established using the Web Service Secure Conversation (WS-SecureConversation) [64] and the Web Service Trust (WS-Trust) [65] protocols. During message transfer over the network, data privacy and integrity are ensured by applying standard encryption encoding and security token exchange as defined in specifications of Web Service Security (WS-Security) [66], XML Encryption [67], and XML Digital Signature (XML-DSIG) [68].

Existing Web service security protocols, however, are used to secure stateless Web service transactions. In order to enable secure access to Grid service/resource instances, the OGSA working group proposed a Basic Security Profile (OGSA-BSP) [69] that declares a set of statements on how to ensure security interoperability at Web service resource level in conformance to existing Web service security protocols. This profile links two other profiles, the OGSA Secure Addressing Profile [70] that defines a set of conformance statements for discovery of security requirements of a particular service/resource instance by extending the WS-Addressing [57] schema and the OGSA Secure Communication Profile [71] that facilitates secure communication to Web service resource instances.

2.3.2 Execution Management Services

Execution Management Services (EMS) defines a set of services, which aim at addressing issues related to execution of Units of Work (UoW), ranging from simple batch job to complex workflows. In particular, the issues include, but not limited to, resource provisioning, UoW placement, and UoW lifetime management. As shown in Figure 2.4, the solution of OGSA EMS is decomposed into multiple abstract and reusable services, each of which targets at specific issue. The following gives details of individual service and its roles in the context OGSA EMS.

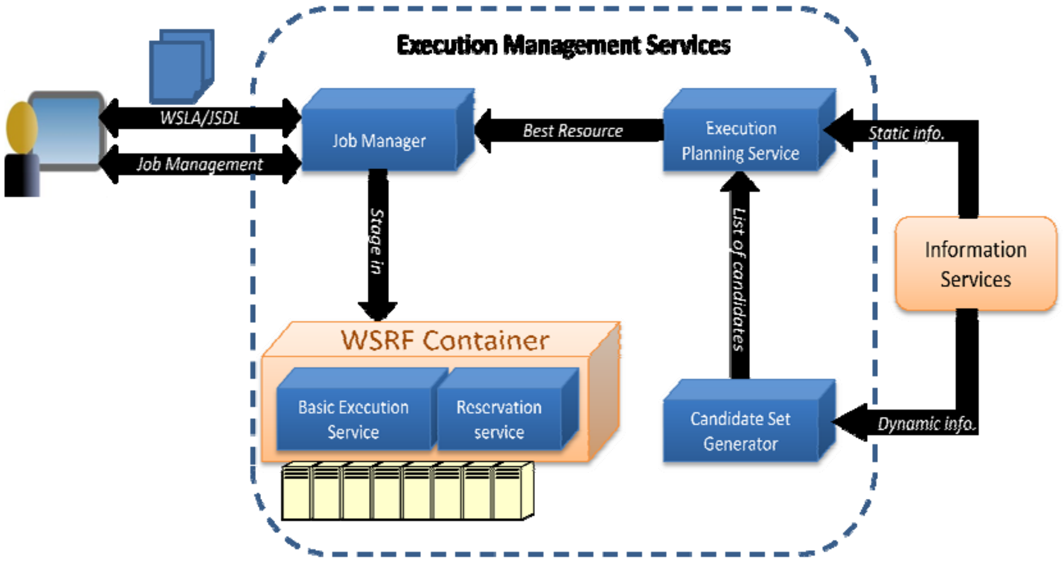


Figure 2.4: OGSA Execution Management Services (EMS) and interactive relations

Job Management

In the context of OGSA EMS, the term, *job*, represents the manageability aspects of a UoW. It is the smallest manageable unit and implements a manageability interface as defined within WSRF-BP [53]. A job has a limited lifetime traversing a set of discrete states (e.g. pending, running, completion, etc). A job can be submitted by end users or spawned by a Grid service with specific runtime requirements, and/or QoS commitments (e.g. reliability, completion deadline, etc). The information related to job submission and job state, along with other metadata (e.g. job owner), is known as the job properties that should be traceable and monitored by clients. OGF defines two specifications related to job submission: the Job Submission Description Language (JSDL) [74] that is a language used to describe the resource requirements of computational jobs for submission to Grid resources, and the emerging Web Service Level Agreement (WSLA) [191], another language specification that is used to describe the job submission with additional agreed QoS terms at service level (such as availability, response time, etc.).

As shown in Figure 2.4, the job manager is defined as the high-level service that provides job manageability facilities. A job manager accepts job submission requests, i.e. JSDL or WSLA instances, and is responsible for orchestrating one or more Grid services necessary to start a job or a set of jobs, for example, negotiating service-level agreements, matchmaking job requests against available resource candidates, optimising resource selection, staging jobs to computational resources and job status monitoring. Job manager may be implemented in various ways. Example job manager implementations include but not limited to:

- a Web portal that allows users to view available Grid resources and perform matchmaking;
- a queuing system that caches job submission requests and distributed them to different resources by applying certain matchmaking algorithm;
- and a workflow manager that receives a number of jobs as a workflow and manage the workflow until completion;

Selection Services

On receiving a job submission request, a job manager is required to determine where to execute a job among a collection of execution resources. The resource selection is a two-stage process involving finding resource candidates and optimise objective functions. Accordingly OGF Resource Selection Service (RSS) working group defines two services related to the resource selection process:

- *Candidate Set Generator (CSG)* [73]: The CSG service is in charge of the selection of a set of computing resource candidates by applying certain match-making algorithms. It mainly deal with low-level technical resource requirements such as CPU type, storage capacity, networking rate. For example, an execution request may specify a list of resource requirements for a target UoW. On receiving the request, the CSG service then returns a list of matched VO resources by interrogating information services (see section 2.3.4).
- *Execution Planning Services (EPS)* [75]: The EPS service takes the match-making results, the outputs of CSG service, and attempts to optimise object function such as execution time, cost, reliability, etc. However both EPS and CSG services do not perform the scheduling process, but returning optimised resources to job manager.

Execution Environment Management

In the context of OGSA, an execution environment consists of every aspect necessary for job execution, particularly including a job container and underlying computational Grid resources. A job container, as its name indicated, contains running jobs, and manages job lifecycles. Example job containers include a queuing service, J2EE hosting environment, batch system, etc. These containers provide common functionalities for creation, monitoring and management of running entities, but in heterogeneous ways and with various interaction interfaces. In order to enable the job manager to interact with various execution environment in a consistent manner, OGF proposed a Basic Execution Service (BES) specification [72] that defines a set of well-defined service interfaces and information models based on the WSRF-BP [53] profile, through which clients can send requests to initiate, monitor and manage computational jobs upon different underlying execution environments.

Besides basic functionalities as specified in the BES specification, a resource provider might also provide optional advanced features. One of such advanced features is the reservation service. State-of-the-art execution environments, such as Portable Batch System (PBS) [192], have advanced reservation facilities implemented to ensure availability of a set of resources to users at a given period. The reservation service proposed in the context OGSA EMS is to define a common interface for creation and management of resource reservations.

2.3.3 Data Services

A variety of data services have proved to be useful to facilitate high-level applications to locate and utilised distributed data resources. These collective services provide primitive mechanisms for management, access, and federation of data resources shared across administrative domains.

Data Resource

A data resource acts as a sink or source of data. There are different types of data resources in a Grid environment, including relational database, XML database, flat files, data stream, etc. Most of data resources are managed by existing systems such as Relational Database Management System (RDBMS) or file systems. Existing data management systems provide similar manageability interfaces mainly for data access and lifetime management but using different manageability interfaces. One of key objectives of OGSA data services is therefore to provide a high-level functional and manageability interfaces upon existing data management systems.

The OGF Data Access and Integration (DAI) working group proposed a stack of standards for data access and management. The Web Service Data Access and Integration Service (WS-DAI)[76] is the core specification that defines a collection of generic service interfaces for uniform data access and manipulation. The WS-DAI specification distinguishes data resources that are managed by external management systems from those to be managed by WS-DAI service. In the case of service managed data resources, the WS-DAI specification recommends implementations to use WS-RF compatible solution for lifetime management. The working group also proposed three additional specifications, the WS-DAIR[77], WS-DAIX[78], WS-DAI-RDF(S) [79][80] which extends core properties defined in WS-DAI core interfaces for realisation of access service to relational, XML, and Resource Description Framework (RDF) [81] data resources respectively.

Storage Resource

Considering some data resources, such as relational database and files, are storage based, it is necessary for OGSA data services to provide functional and manageability interfaces for storage resources as well. Like data resources, most storage resources are managed by existing storage management systems. These storage management systems provide custom solutions to control and provision of raw storage or space in a file system as well as custom file access protocols. OGSA data services are intended to provide an abstract manageability interface for storage management over heterogeneous storage management systems.

The OGF Grid Storage Management (GSM) working group focus on the definition of standard interfaces of a middleware component, the OGSA Storage Resource Manager (OGSA-SRM)[80], which provides dynamic storage resource allocation and file management facilities to storage resources shared in the Grid. File access to storage resources can be achieved through two standard mechanisms as proposed by OGF working groups. The OGF GridFTP working group proposed a standard file access protocol, the Grid File Transfer Protocol (GridFTP) [82], which extends from File Transfer Protocols with enhanced security and performance. Remote files can be alternatively accessed through a set of standard interfaces defined within the OGSA Byte Input/Output (OGSA ByteIO) specification [83], which provides “POSIX-like” file functionalities. These standardisation efforts make it possible to implement advanced data features of OGSA data services. File replication, for example, is an important feature than enhances system performance and fault tolerance in a Grid system. The OGSA Data Movement Interface (OGSA-DMI) [84] is a recent specification proposed by the OGF DMI working group and simplifies data transfer across multiple storage and data resources through a set of standard interfaces.

2.3.4 Information Services

The Grid environment consists of a huge amount of highly distributed and heterogeneous resources, which are coordinated to accomplish complex application goals. The OGSA defines a set of collective capabilities that hide low-level complexity of a Grid environment. Information services exhibit one of such high-level capabilities by providing efficient access to information about resource/services, applications and events in a Grid environment. The information supplied by an information service is intended to be used for various purposes including resource/service discovery, system performance tuning, fault detection, and accounting. There are two main components of information services, the logging service and discovery service. Logging services work at the infrastructure layer and produce dynamic status information of individual resources. Discovery services are likely to be deployed in every Grid system and act as registry maintaining static resource information as well as dynamic information collected from multiple logging service instances.

Rather than defining a single information service to support all usage scenarios, which is impossible, current standardisation efforts on OGSA information services are at highly abstract level without compromising service usability. The OGSA Grid Monitoring Architecture (OGSA-GMA)[85] specification defines essential interactions among three abstract components in a Grid monitoring architecture, the information provider, information consumer and directory. Another important point issue relating to OGSA information services is resource information models. Resource information models describe resource-specific semantics by defining resource-specific properties, operations and relations to other resources. There are many other industry standards for resource modelling, such as the Common Information Model (CIM) [86] defined by DMTF, the resource model proposed by Java Management Extensions (JMX) [87] framework, etc. It is likely that implementations of information services of different Grid projects may apply these standards for resource modelling. In order to enable interoperability between Grid-specific information services, the OGF Grid Laboratory for a Uniform Environment (GLUE) working group defines an abstract information model, known as the GLUE schema[88], as a legacy schema that can be mapped to concrete schema employed by a Grid information service.

2.3.5 Security Services

OGSA security services provide facilities to enforce security policies in a VO. From security perspectives, a VO maintains certain security policies that is outsourced by resource providers and coordinates their resource sharing and usage in a consistent manner. To be more specific, VO-specific security policies pulls together user participants and resource/services from disparate domains into a common trust domain. Compare to traditional means of security administration that involves a centralized policy databases of user credentials, administration of VO security policies in the OGSA environment is complicated by the dynamic nature of VO. A VO needs to establish trusts between users and Grid resource/services. Theses trust domains spans multiple user participants that dynamically join and leaving and multiple resource/services that are dynamically deployed or created over the lifetime of a VO. The establishment of dynamic trust domains of a VO requires a delegation mechanism that allows one entity to grant rights to another (e.g. newly created resource or services) to perform actions on its behalf. Besides, user participants in a Grid environment may need to coordinate multiple resource/services to accomplish a single task. OGSA security services are also required to provide a single sign-on mechanism to ensure that the user is authenticated exactly once and need not to be re-authenticated upon following access to Grid resource/service during a period of time. Access to Grid resource/services must be authorized by security policies specified by resource/service providers as well as those from VOs. OGSA security services need to provide a standard authorization framework that accommodates various access control models and implementations deployed by service providers.

Within OGF, an OGSA security working group has been founded to enumerate and address aforementioned security issues in the context of OGSA. The initial profile specification, Grid Certification Policy (GCP) [89], provides a guidance for the use of attributes and extensions of the Internet X.509 Public Key Infrastructure Certificate [89] to accommodate advanced security requirements such as delegation and single sign-on in OGSA environment. Another security-specific working group, the OGSA Authorisation working group, focuses on addressing interoperability issues among multiple authorisation domains by defining a generic authorisation framework. The recent released informational document, Functional Components of Grid Service Provider Authorisation Service Middleware [90], proposed two OGSA authorisation models from the resource/service providers' point of view, the pull model and push model. By push model, user credentials and authorisation assertion of a VO are attached with request message to service provider. On receiving access requestors, resource/service providers are required to validate assertion and apply local authorisation policies. On the other hand, a resource/service provider is required to call VO authorisation decision point to get user attributes or authorisation assertions before applying local authorisation polices. This model is known as the pull model. Based on proposed authorisation framework, the working groups are working on defining standard authorisation protocols compatible to XML Access Control Markup Language (XACML) [91] and Security Assertion Markup Language (SAML) [92] proposed by OASIS.

2.3.6 Self-management Services

Self management capabilities have received increasing attentions in OGSA. A self-management Grid environment composed of autonomous services (see section 1.1.4) that are self-configurable, self-healing, and self-optimising. One of the major objectives of self-management in a Grid is the support service-level attainment for OGSA resource/service through a conceptual component, the Service Level Manager (SLM). The SLM component is modelled after a generic control loop pattern, which consists of monitoring, analysis and projection, and action phases. A SLM may be used to control and adjust service activities at different levels. Grid system-level SLMs, for example, can be used for improving resource utilisation by dynamically enrolling resources or releasing surplus resource depending on current system load. Although identified as a significant part of OGSA, standardisation efforts self-management services are still at a preliminary stage.

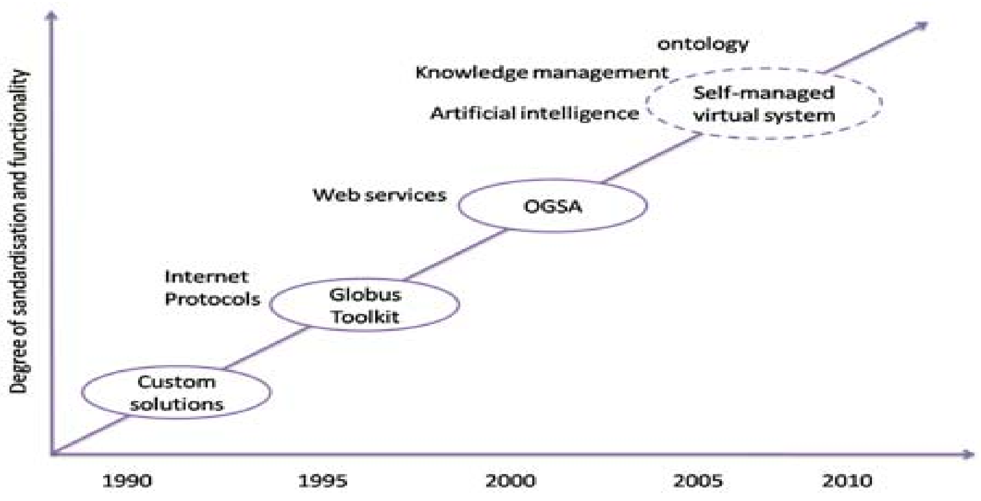


Figure 2.5: Evolution of Grid Middleware Technologies. From [33]

2.4 Middleware

Having identified requirements and capabilities that are fundamental to the success of Grid applications, considerable progresses have been made during past ten years in developing Grid middleware. As illustrated in figure 2.5, the evolution of Grid middleware is divided into four phases. Starting from early 1990s, Grid technologies concentrated on addressing meta-computing [93] issues through linking heterogeneous computational resources in such a way that are transparent to users as a single computer. Middleware development uses various solutions to achieve a limited set of functionalities, security and scalability in particular, therefore not concerning about interoperability. The emergence of Globus Toolkit version 2 (GT2) in 1999 became the first de factor standard and pioneered the creation of interoperable Grid middle. Services and protocols defined within GT2, however, are based on internet protocols and implementation-oriented. It is not possible to have different implementations of Grid middleware until 2002 when a community of standards released based on OGSA profile, which aligns Grid computing with broad Web service protocols. Since then, a great number of standard-compatible Grid middleware released. It is also envisioned that the evolution of OGSA-compatible middleware will eventually lead Grid computing in another stage with enhanced features on autonomy and self management. The section reviews Grid-middleware solutions both OGSA compatible and OGSA non-compatible.

2.4.1 Globus Toolkit

Globus Toolkit (GT) is an open-source toolkit that forms a fundamental technology enabling Grid computing. The project was founded in late 1990s and originated from the US national project, I-WAY [95], which aimed at providing inter-connection between eleven high-speed research networks. Since version 1.0 release in 1998, version 2.0 in 2002 and recent release Web service compatible version 4.0, GT has evolved rapidly as a standard Grid middleware and forms foundation for thousands of Grid projects worldwide in both scientific and industry fields. However early adoptions, such as gLite (see section 2.4.2), are mainly based on GT2, which addresses issues relating to security, resource management, monitoring, discovery and file transfer at resource layer. These projects have various custom solutions developed upon GT2 components to address high-level issues for coordinated resource access and VO management. As presented in figure 2.5,

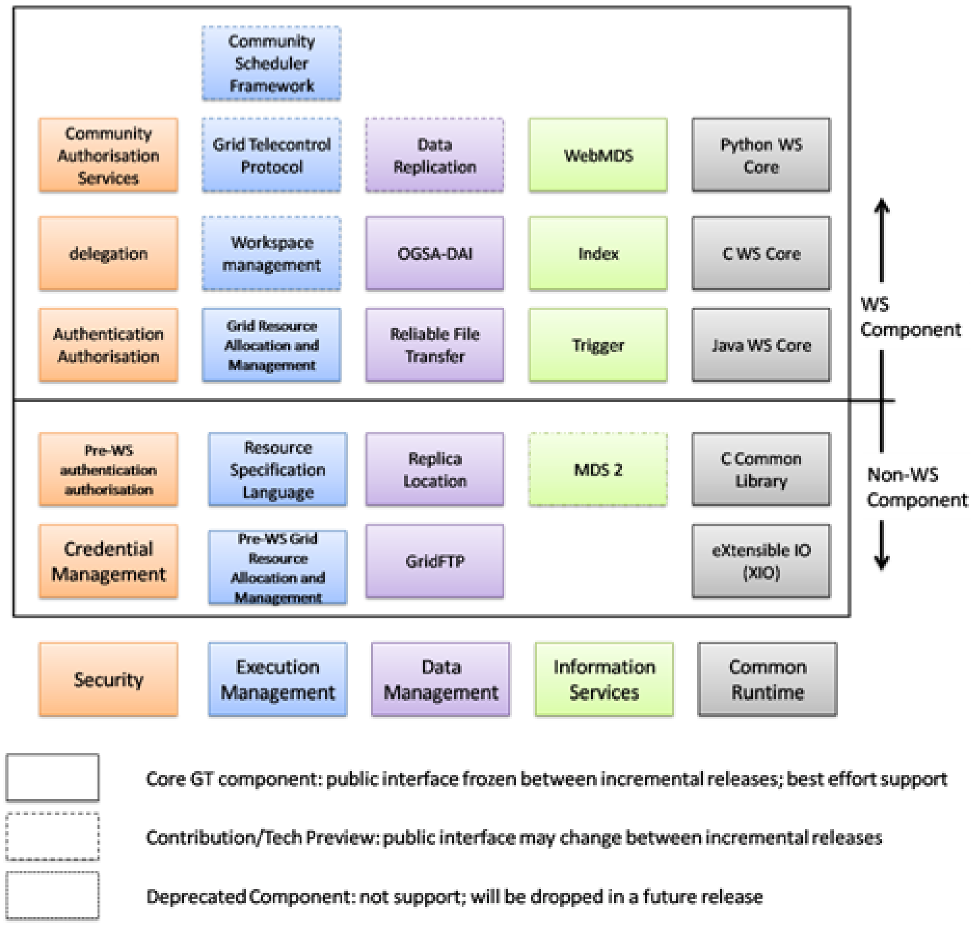

Community Scheduler Framework
Community Authorisation Services
Grid Telecontrol Protocol
Data Replication
WebMDS
Python WS Core
delegation
Workspace management
OGSA-DAI
Index
C WS Core
Authentication Authorisation
Grid Resource Allocation and Management
Reliable File Transfer
Trigger
Java WS Core
WS Component
Non-WS Component
Pre-WS authentication authorisation
Resource Specification Language
Replica Location
MDS 2
C Common Library
Credential Management
Pre-WS Grid Resource Allocation and Management
GridFTP
eXtensible IO (XIO)
Security
Execution Management
Data Management
Information Services
Common Runtime
Core GT component: public interface frozen between incremental releases; best effort support
Contribution/Tech Preview: public interface may change between incremental releases
Deprecated Component: not support; will be dropped in a future release

Figure 2.6: Globus Toolkit Components (Pre-WS vs. WS releases). From [94]

the Web service-based GT4 provides significant improvements in terms of community functionalities and OGSA standard compliance.

Common Runtime Environment

Common runtime environment of GT consists of a set of components that abstract low-level connectivity protocols in a platform independent manner. The pre-Web service release of GT provides two runtime tools, the eXtensible Input/Output (XIO) [96] and C command library. The XIO represents a simple Open/Close/Read/Write (OCRW) interface that provides an abstract layer upon transport protocols, such as TCP and UDP. A common library written in C programming language implements most infrastructure functionalities of GT2, including security, introspection and management facilities for development of custom services in a platform independent manner. GT4 leverages Web service stacks and provides WS-RF compatible runtime environments. There are three-version containers available for service development in Java, C, and Python programming languages.

Execution Management

Execution management in GT is realised by the Grid Resource Allocation and Management (GRAM) [97] component, which defines standard protocols allowing initialising, monitoring, and managing execution of jobs on remote computational resources. However GRAM is not a job scheduler, but abstracts a single protocol for communicating with the Local Resource Management System (LRMS) and allows a client to specify resource requirements using Resource Specification Language (RSL) [98]. The GRAM component also provides operations for monitoring status of execution resources.

The GRAM component is refactored in GT4 and provides standard Web service interfaces for job submission and management, therefore also known as WS-GRAM. Two additional functional sub-components are added in WS-GRAM. The workspace management service functions as a sandbox and dynamically allocates local Unix accounts to execution requestors. A more general protocol, Grid TeleControl Protocol, is also provided in WS-GRAM mainly for instrumentation management, such as management of earthquake engineering facilities and microscopes.

Data Management

Data Management components of GT provides facilities to data access, transfer, and replication. GT's implementation of GridFTP [82] protocol enables secure and high-performance file transfer over Wide Area Network. The Replica Location Service (RLS)[99] acts as a registry of file replicas and provides two-level naming mechanism allowing mapping multiple user-defined logic file names to target physical file location. However the RLS itself does not guarantee either file consistency or filename uniqueness. It is expected high-level services would provide these advanced features.

Data management in GT4 is enhanced by introducing two high-level manageability interfaces for data transfer and replication. The Reliable File Transfer (RFT)[100] service provides Web service interfaces for management and reliability of multiple file transfers using GridFTP protocols. A prototype service, Data Replication Service (DRS)[101], is expected to hide the complexity of the overall processes of data replication by allowing users to identify a set of desired files in the Grid environment, to make local replicas of those files by transferring files from one or more source locations through RFT service, and to register the new replicas in a RLS. A third-party tool developed by UK e-Science program, the OGSA Data Access and Integration (OGSA-DAI) [102], is integrated within GT4 as a data management component providing access and management facilities to other structured data, relational and XML data in particular.

Information Services

Information services in GT are enforced by the Monitoring and Discovery System (MDS) for collection, indexing, discovery of resource/service information in a Grid. MDS implemented in GT2, called MDS2, is based on Lightweight Directory Access Protocol (LDAP) and consists of three hierarchical components: Grid Information Index Service (GIIS), Grid Resource Information Service (GRIS), and Information Providers (IPs). Resource/service providers may have multi-purpose monitoring sensors running on a resource/service to collect information data such as CPU load, system configuration, etc. The IPs provides an abstract interface layer upon local monitoring sensors so that resource-specific data can be collected and published in a consistent manner. The GRIS runs on a resource/service and acts as a modular content gateway for a resource. GRIS instances are registered to a GIIS endpoint, where information data are indexed and cached. Information consumer may optionally query information of a specific resource directly to GRIS or talk to GIIS to obtain collective information. MDS2[103] defines a resource information model for computational resources only, known as MDS schema. Information providers may also publish GLUE-compatible information model by configuring a LDAP implementation of GLUE schema.

The MDS2 is no longer maintained and replaced by a Web service compatible solution, known as MDS4, in GT4. MDS4 is built upon standard query, subscription and notification protocols as defined in WS-RF and WSN specifications. Based on these standard protocols, a range of GT4 components, such as WS-GRAM and RFT, are implemented as information providers for collection of information from specific resource/services. An adapter interface is also provided for those information providers that are not WSRF compatible. MDS4 also provides two high-level services, the aggregator services and trigger services, for collection and publishing aggregated information from information providers. Both services are implemented based upon a generic aggregation framework. Finally a Web-based interface, WebMDS, provides a visualisation interface for user to view information data. MDS4 uses GLUE schema natively and provides an XML mapping of the GLUE schema.

Security

GT provides a Grid Security Infrastructure (GSI) [104] based on X.509 PKI, which assumes every user and host involved in a Grid has an X.509 end entity certificate signed by trusted CAs. Each Grid transactions is mutually authenticated and encrypted. In order to support Grid-specific requirements on single sign-on and delegation, GSI also supports proxy certificates that are derived from X.509 end entity certificates. User participants may issue a self-signed proxy certificate delegating their rights to another entity within a limited period of time. An online credential management service, MyProxy server[105], is used for generating, querying, and renewing such proxy certificates. Resource access is protected by a simple resource-level authorisation mechanism defined in GSI by mapping the subject of a user certificate to local execution environment, Unix user account for example. In the case, the Grid user has the same access rights of the local account.

Based on GSI, GT4 provides messaging-level security mechanisms by implementing WS-Security[66] and WS-SecureConversation[64] protocols to protect SOAP messages. A high-level authorisation service, the Community Authorisation Service (CAS)[106], is also implemented in GT4 allowing separation of resource providers' security policies and VO security policies. In another word, resource providers may delegate a subset of security policies to the VO. In this sense, CAS provides fine-grain mechanisms for a VO to manage these delegated policies and ensures user requestors are authorised across multiple security domains in a consistent manner.

2.4.2 gLite

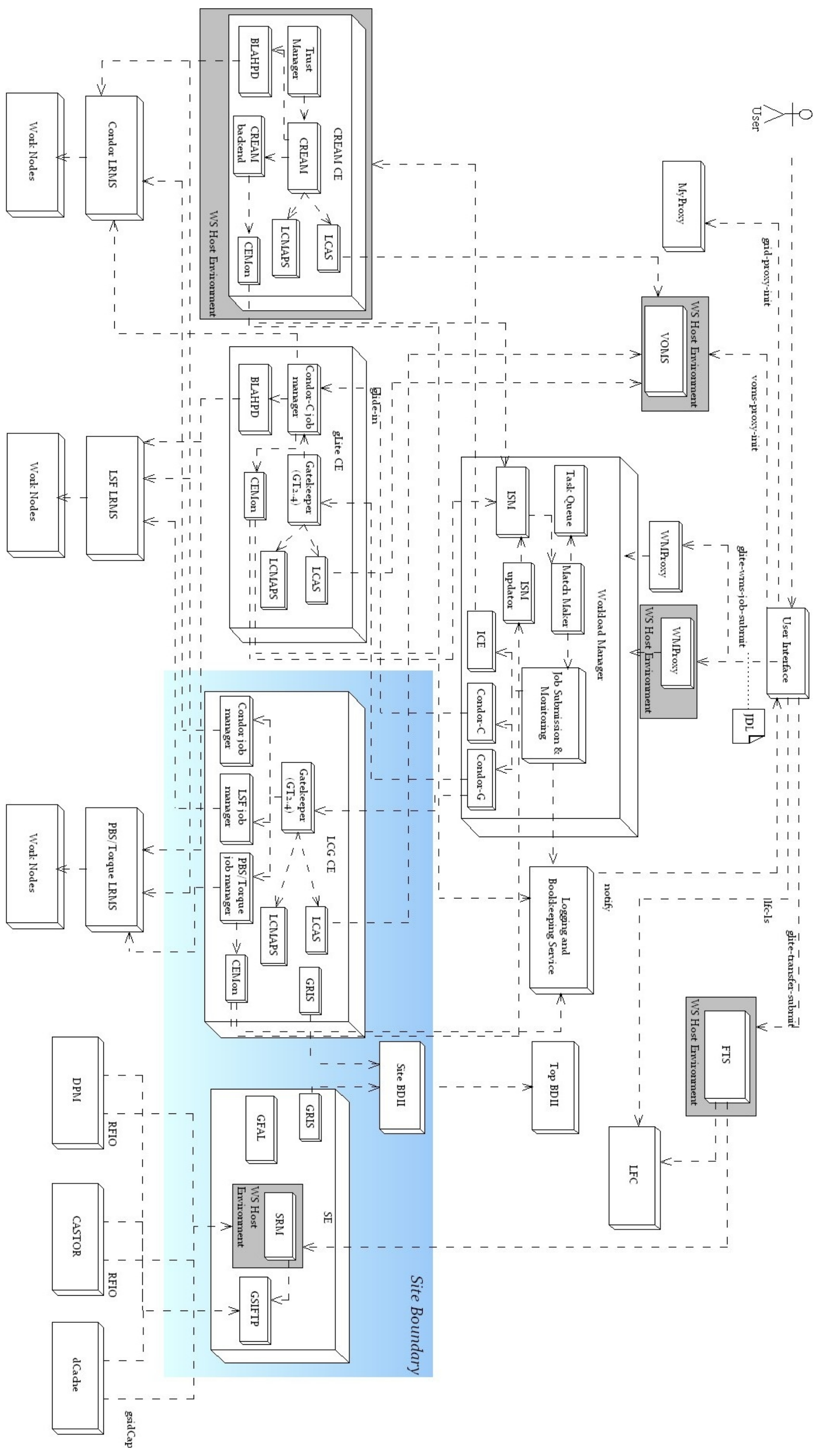


Figure 2.7: gLite3 Component Architecture and Internal Interactions

The gLite, a lightweight Grid middleware solution produced by EGEE project, provides a framework to build Grid applications for diverse research communities. The gLite middleware combines component distributions from a number of other projects, including Virtual Data Toolkit (VDT) [107], European Data Grid (EDG) [108] and World-wide LHC Computing Grid (WLCG)[109] projects. Since EGEE project and WLCG project share a large part of infrastructure consisting of computing and storage resources shared over 200 distributed sites around the world, the gLite middleware [110] is primary deployed on participating sites in EGEE/WLCG project.

As figure 2.7, current release gLite middleware, gLite3, follows SOA design patterns and is evolving to be OGSA compatible where possible. Meanwhile it also reuses some GT2 components for backward-compatibility to LCG deployments.

Access Services

The User Interface (UI) is the entry point to a gLite-enabled Grid. A user accesses gLite resources or services by logging on a UI machine, where user certificates are installed. The UI provides Command-Line Interfaces (CLIs) allowing users to interrogate high-level gLite services. From a UI, a user may submit a job execution request, monitoring job status, get job output, transfer files, etc.

Computing Element

The Computing Element (CE) is a generic terminology defined in EGEE/WLCG referring to a set of computing resource at a site. A CE provides a generic interface, known as the Grid Gate (GG), which is responsible for scheduling jobs to a collection of Worker Nodes (WNs) via a LRMS. The gLite version 3 supports a wide range of LRMS including Portable Batch System (PBS)[111], Condor[112], Load Sharing Facility (LFS)[113], etc. As demonstrated in figure 2.5, there are three implementation of GG in gLite: the gLite CE, LCG CE, and Computing Resource Execution and Management (CREAM)[114] CE.

LCG CE was developed by EDG project [108] and used in LCG. The LCG CE runs a GT2's GRAM gatekeeper and reuse GRAM job manager interface as GG. A site may choose to configure one or more job managers according to LRMS deployed. There is one gatekeeper per CE. The gatekeeper will publish available job managers to the gLite information system. On receiving a job request, the gatekeeper forks a job manager instance after authenticating user identity and mapping it onto a local user account. The job manager instance then dispatches the job to WNs via corresponding LRMS.

The LCG CE processes job requests on per process per user basis, resulting in scalability issues. In EGEE/WLCG project that involves thousands of users, it is very likely that multiple users send job requests to a LCG CE simultaneously. In order to cope with this issue, the gLite introduces a three-tier CE architecture, so-called gLite CE, based on gatekeeper, Condor-C[115] job manager and Batch Local ASCII Helper (BLAH) protocol[116]. The Condor-C is a Condor-to-Condor job scheduler that allows jobs in one Condor queue to be moved to another Condor queue. For those LRMS other than Condor, Condor-C job manager makes uses of the BLAH command for job submission and management. The BLAH protocol defines a set of plain ASCII commands to manage jobs on the batch systems. A lightweight BLAH protocol daemon (BLAHPD) is responsible for converting BLAHP commands into LRMS commands, trigger those commands and report results back in BLAHP format.

Finally the recently developed CREAM CE provides an alternative solution of job submission and management at CE level. The CREAM CE implements OGSA BES specification and uses BLAHPD for job scheduling and management to LRMS including Condor. The CREAM backend is a permanent memory space for storing data related to all cached and executing jobs.

gLite also defines a CE Monitoring (CEMon) service that is deployed at individual CEs and responsible for providing characteristic and status information of the CE. The major consumer of CEMon service is the Workload Manager System (WMS) that performs job submissions by matchmaking job requirements and dynamic CE status information obtained via CEMon services. The CEMon service provides an extension point through which custom CEMon sensor can be plugged in to generate other information. The CREAM sensor, for example, is plugged into CEMon service to generate job status information.

Storage Element

The Storage Element (SE) as defined in gLite provides data access and manageability facilities to storage resources localised at a site. Most of sites participating in EGEE/WLCG project normally provide at least one SE. There are three widely used storage management systems in EGEE/WLCG, the CERN Advanced STORage manager (CASTOR) [117], dCache [118], and the Disk Pool Manager (DPM) [119]. As figure 2.5, these storage management systems have different protocols defined for file access. The Remote File Input/Output (RFIO) provides POSIX-like interface for access files through CASTOR and DPM management system, while dCache uses a GSI-enabled data access protocol, the gsidcap. A high-level abstract, the OGSA SRM service, is implemented by these storage management systems in order to ensure file access through heterogeneous storage management systems in a consistence manner. In addition to system-specific file access protocols, gLite requires all SEs must support a GSI-enabled FTP protocol, the GSI-FTP .

Workload Management System

The Workload Management System (WMS) [120] component provides Grid-wide resource management facilities hiding complex gLite environment from users. The main purpose of WMS is to satisfy user requests by taking appropriate actions on job submission and management on behalf of users. It accepts job execution requests from UI, selects CE candidates, places job execution, and notifies execution results. A user request specifies job and resource requirements in JDL (Job Description Language) [121], which is the Condor ClassAd language therefore legitimate to be used directly to Condor APIs for job management. The JDL allows the description of three request types including simple job request, Direct Acyclic Graph (DAG) job request, and a collection of independent jobs that can be executed in parallel. WMS exhibits two entry points for users, the Network Server (NS) and Workload Manager Proxy (WMProxy). The NS is a generic network daemon that keeps listening to user request from a well-know port. WMProxy provides a Web service interface to access WMS functionalities. Both services check user authorisations and forward JDL to the Workload Manager component.

Workload Manager (WM) is the core component of WMS. On receiving a JDL, the WM spawns a matchmaking process, which evaluates JDL items against Information Super Market (ISM). The ISM consists of a repository of CE information. In order to ensure information is up-to-date, a lightweight process, ISM updater, contacts CEMon service and refreshes ISM repository periodically, approximately every two minutes.

Alternative, ISM may subscribe to CEMon services to receive notifications encompassing needed CE information.

Once a CE candidate identified, the JDL is forwarded to a Job Submission and Monitoring component, which is responsible for creating a wrapper script that creates the appropriate execution environment in the CE worker node. The Interface to CREAM Environment (ICE) is used by WM when interacting with CREAM based CEs. In the case of no matched CEs found immediately, the WM component caches the job request into an internal Task Queue (TQ) for a while. Under this circumstance, the jobs held by WM can be either asking for RB to perform matchmaking periodically (eager scheduling policy) or waiting for an appropriate CE to pull job request from TQ when available (lazy scheduling policy). During a job lifetime, changes of job status are maintained and updated within the Logging and Bookkeeping (LB) service.

Data Management

Two high-level services are provided within gLite3 for file transfer and replica management. The File Transfer Service (FTS) provides low level data movement service that can schedule asynchronous and reliable file replication from source to destination SEs. It also allows participant sites can control the network usage. The FTS interacts between source and destination SEs through standard SRM interfaces and GridFTP protocol. Users and applications locate files or replicas through the LCG File Catalogue service (LFC), which maintains mappings between user-defined Logical File Names (LFN), a Global Unique IDentity (GUID) and physical Storage URL(s) of replicas. The LFC service publishes its service URL in gLite3 Information Services so that it can be discovered by data management tools and other services.

Information Services

There are two Information Services (IS) in gLite3, the pre-WS MDS of GT2 and Relational Grid Monitoring Architecture (R-GMA)[122], a relational database implementation of OGSA-GMA[85] specification. For MDS service, a Generic Information Provider (GIP) runs at resource layer and generates relevant information about computational and storage resources. This information is stored and cached in a GRIS server for each resource. Each GRIS is registered with a site-level Berkeley

Database Information Index (BDII) and populates the database with resource information. The site-level DBIIs are then registered to a top-level BDII used as the top of the hierarchy of a VO. R-GMA is an alternative information service mainly used for accounting purpose and is discussed in more detail in Chapter 3.

Security

The gLite3 middleware uses GSI and MyProxy server for use authentication, single sign-on, and delegation. The authorisation framework in gLite3 composed of a centralised community-based authorisation service, the Virtual Organisation Management Service (VOMS)[123], and site access control suite comprising Local Centre Authorisation Service (LCAS) and Local Credential Mapping Service (LCMAPS). The VOMS organises user information and privileges in a hierarchical structure. Each user in a VO is assigned to a subgroup, a role, and granted capabilities. This information is represented via an extension to user proxy certificate. At the time a VOMS is contacted, a VOMS proxy certificate that encapsulates user's group membership and associated roles into standard proxy certificate is signed by VOMS public key and returned. The VOMS proxy certificate is push into CEs together with job requests. At CE level, the LCAS service is called by gatekeeper to make an authorisation decision based upon user subject name and VO attributes embedded within a proxy certificate. Once authorized, the LCMAPS service takes care of translating grid credentials into Unix credentials local to the site.

2.4.3 UNICORE

The Uniform Interface to COmputing REsources (UNICORE) project [124] was established in 1997 to provide an easy-to-use platform that enables secure access to supercomputer sites in German. After twelve-year development, the UNICORE project has evolved as a SOA Grid middleware for secure access mainly to computational resources. As figure 2.8, the recent released UNICORE version 6, called UNICORE6, is characterised as a vertically integrated Grid system that comprises components of three tiers, the client tier, service tier, and target system tier.

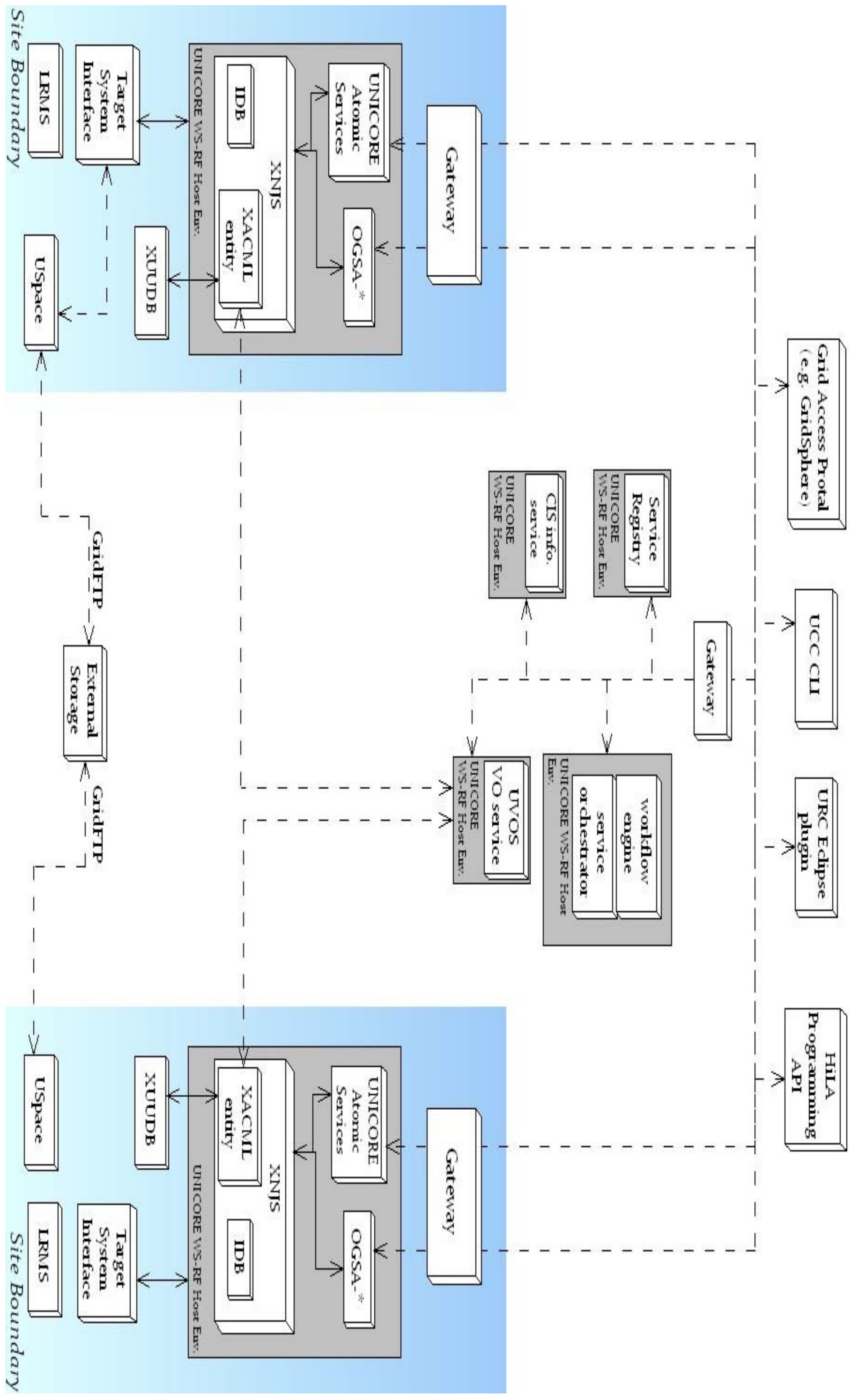


Figure 2.8: UNICORE6 Component Architecture. From [124]

Client Tier

The UNICORE client tier provides a variety of client interfaces to exploit the entire set of services offered by the service tier. The UNICORE Command-line Client (UCC) provides a versatile Command Line Interface (CLI) that allows users to access all service-tier features in a shell or scripting environment. The UNICORE client tier also consists of two programming APIs. The UNICORE rich client is an eclipse-based Grid Programming Environment (GPE) developed by Intel. The co-called “rich” client provides graphical user interface and interoperable GridBeans [125] for Grid application development. Alternatively, application developer can use the single interface of High Level API for Grid applications (HiLA) to implement complex application with just a few lines of codes. Finally UNICORE services can also be accessed from third-party portals, GridSphere[126] for example.

Service Tier

The UNICORE service tier comprises all services and SOA components based on WS-RF and WS-I standards. A site level, UNICORE services consist of two main functional components, the Gateway and enhanced Network Job Supervisor (NJS). The Gateway component acts as a site firewall and performs the authentication of all incoming requests to underlying site resources. The NJS component is the job management and execution engine of UNICORE6. Its functions include storage resource management, file stage in or out and job management. The functionality of the NJS is accessible via two Web service interfaces: The UNICORE Atomic Services (UAS) and OGSA-BES. The UNICORE job definition is compliant with the JSDL standard. A variety of protocols, such as HTTPs, OGSA ByteIO and GridFTP, are also available for staging files between sites or between client and sites. On receiving a job request, the NJS component delegate the JDSL file to the IDB (Incarnation Data Base) component that performs the job incarnation and maps the abstract job description in JSDL to the concrete job description for a specific resource. Information about available applications and resource characteristics has to be defined in this database. For authorisation, the NJS uses the X.509-baed UNICORE User DataBase (XUUDB) to map the subject name of user X.509 certificate into the actual user account and group. XUUDB based authorisation can accommodate all access control requirements within a single site. For resource access cross sites, file transfer from different sites for example, UNICORE6 supports proxy certificates and provides an

XACML entity that can be triggered to delegate access decision to a VO management system, the UNICORE VO service (UVOS) [127].

Like many Grid middleware, UNICORE6 also provides several collective services. Firstly, a single service registry is available to build-up and to operate a distributed UNICORE infrastructure. This service registry is contacted by the clients in order to connect to the Grid. The UNICORE Common Information Service (CIS) is the information service, which gathers and stores both static and dynamic information from all connected XNJS into GLUE 2.0 [88] format. UNICORE also supports workflow management using a two-layered architecture consisting of a workflow engine and the service orchestrator. The workflow engine allows different workflow description dialects to be plugged in according to site requirements. The main responsibility of the service orchestrator is to execute the individual tasks in a workflow, handle job execution and monitor the Grid.

System Tier

The system tier provides an abstract non-WS interface, the Target System Interface (TSI), between UNICORE and underlying LRMS of Grid resources. Communication between XNJS and TSI is through text-based protocols, which are interpreted into system-specific commands. In addition, the TSI component is extended for supporting the DRMAA standard enabling a standardized interface between the TSI and the batch system in UNICORE6. The UNICORE Space (USpace) is the space for job directories. A separate directory created on a per job basis, where the XNJS and TSI stores all input, output and error data. GridFTP can be used for site-to-site file transfer, in particular for data transfer from/to external storages.

2.4.4 Others

The middleware solutions discussed above are widely deployed as production Grids for a variety of research communities. There are some other Grid projects that have custom middleware developed to accommodate local deployment environment. Here list two other common Grid middleware that are being deployed at some participating sites of EGEE/WLCG projects.

Advanced Resource Connector

Advanced Resource Connector (ARC) [128] is the middleware developed by Nordic Grid, a project that aims at providing a Grid infrastructure for Nordic countries. The major design goal of ARC middleware is to provide innovative solutions that are essential for a production quality middleware. The ARC middleware consists of three main components: Grid services, Indexing services, and user interface. ARC Grid services are a collection of services running on resources. Grid jobs are submitted to ARC resources through GridFTP protocol. Each site has a GridFTP server that keeps listening to incoming job requests. A Grid manager is responsible for computational resource management and takes care of job execution and input data cache. Information services are implemented as a “cron” script that periodically updates local resource information to the Indexing service backend. Indexing services (IS) uses GT2 GIIS and maintains a list of local information services and other IS endpoints. The ARC user interface is a set of tools for job submission, monitoring and management. An intelligent resource broker is built in the user interface, which is able to select the best matched resource for user jobs.

Virtual Data Toolkit

The Open Science Grid Project (OSG) aims at bringing together computing and storage resources interconnected over research networks from campuses and research communities in the US into a common, shared infrastructure via a common set of software stack. The OSG software stack relies on Virtual Data Toolkit (VDT)[107] middleware, which ensembles GT2, Condor, EDG and other open source software. The goal of VDT is to make it as easy as possible for users to deploy, maintain and use Grid software rather than defining Grid middleware.

2.5 Tools

Grid middleware provides fundamental services that allow resource access, management and manipulation through well-defined interfaces. However these low-level service interfaces are too complex making the Grid elusive for many users. For example, a scientist must learn details of gLite’s execution services to submit and monitoring a job request. Besides, development of a Grid-enabled application becomes even more complicated and requires developers to become familiar with detailed interfaces. There is a clear need for tools that allow application developers to use, to write Grid-enabled applications, and allows users to easily deploy and run applications on the Grid. These tools should build upon the Grid infrastructure and lie at the application layer (as shown in figure 2.1) therefore known as application tools.

For application developers, Grid application tools should provide high-level abstractions and support a broad class of applications development. This can be achieved by evolving traditional multi-purpose programming models, such as Remote Procedure Call (RPC), message passing, and parallel programming models, to take advantage of the Grid platform. The OGF Grid RPC working group is working on defining a standard Grid-RPC [129] API for both middleware developers and end-users, while ensuring interoperability among domain-specific middleware. The GridRPC model also provides a mechanism for task parallelism by partitioning a complex job into multiple processes to be executed in parallel on multiple Grid resources. Finally, message-passing programming model is the most general model for parallel computing. Grid-enabled implementations of the messaging-passing model have been pursued by many research groups. The MPICH-G2 [130] is such a Grid-enabled implementation of MPI standard based on GT infrastructure.

The second class of Grid application tools is to provide Grid application execution environments allowing user to easily interrogate different services of underlying Grid middleware. There are two common classes of such environments, Grid workflow system and portal, available in most existing Grid middleware. The WMS of gLite3, for example, uses DAGMan [131] as a workflow manager that allows representation of a collection of job dependencies as a directed acyclic graph. A more generic workflow engine in UNICORE, as discussed in section 2.4.2, supports flexible workflow management and enables different workflow dialects to be plugged in. Another effective means of Grid application execution is the Web portal, which is linked with middleware services and provides graphic interfaces.

2.6 Summary

This chapter discussed the concept, architectural principles, standards, middleware solutions and software tools, which makes it possible to develop interoperable and versatile Grid systems. In addition, the content of this chapter also implies future development in Grid applications and technologies. First of all, most of the existing Grid middleware solutions, such as VDT, gLite, and ARC, are based on tools and experiences established over past years, which are not OGSA compatible. The wide deployment of these middleware solutions in production Grid projects makes them hard to be OGSA compatible. One feasible solution would be to implement OGSA components while keeping backward compatibility to existing counterparts. The CREAM project sets a good example by introducing OGSA BES service into gLite. Besides, the OGSA architecture is evolving over time. It is very likely that more features would be added in OGSA architecture. Therefore extensive standardisation efforts should be continuously contributed. Finally, as discussed in section 2.5, Grid application tools play an important role in making Grid technologies user- and developer-friendly.

Chapter 3

Grid Accounting

Grid accounting plays an important role in system administration, resource usage policing and enforcing Grid economic models. The main purpose of Grid accounting is to meter and supply usage information of resources shared in a Grid environment. Collective usage information helps enrich system administrator’s understanding and enhance overall resource utilisation in a Grid system. For most e-Science Grids, computing resources are predominately provided from academic institutes for one or more non-profitable research projects. Individual project and participants are granted a fixed quota, such as computational cycles and storage spaces. Accounting in such e-Science Grid environment enables usage policing that prevents Grid resources from over exploitation by checking the actual resource usage against allocated resource quota of individual project. Resources or services provided by a business Grid are to be utilised in the “pay-per-use” pattern. Accounting in this case is mainly used to provide usage proofs for charging users based on actual resource usage. Besides, Grid accounting can also be used for strengthening security, guaranteeing Quality of Service (QoS), etc.

Having identified the importance of Grid accounting, there are increasing Grid projects that have accounting systems developed and deployed. These accounting systems, however, were designed in various ways to accommodate Grid-specific usage scenarios. In order to provide a consistent and interoperable solution to Grid accounting in the context of OGSA profile, this chapter discusses the concept of Grid accounting, reviews existing accounting solutions in operational Grid projects, and proposes an generic accounting framework. I conducted all the research carried out within this chapter and published in [132].

3.1 Concept

The concept of Grid accounting was firstly proposed as “a process that provides a consistent and Grid-wide view of VO members’ resource utilization” [133] at the time of designing accounting system for Sweden Grid (SweGrid), a national Grid project that provides computational resources to scientific projects in Sweden. This definition, however, merely highlights SweGrid-specific requirements on accounting. It is worthwhile to review the concept of accounting in order to give a more generic definition.

The terminology, accounting, originates from business and financial field as “the system of recording and summarizing business and financial transactions and analyzing, verifying and reporting the results” [134]. Accounting is by no means a new concept in computing either. In a UNIX system, the usage of individual system resources is accurately recorded and maintained. The process accounting, for example, logs every single command run by every single user through the PACCT script. The UNIX operating system can also be configured to enable disk accounting by periodically scanning each file system and finding out its disk usage.

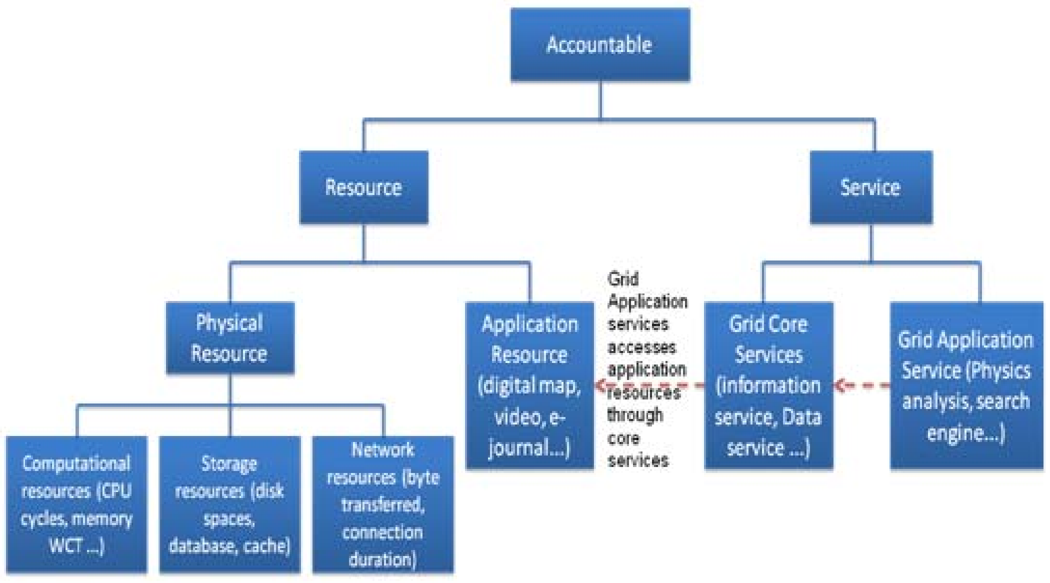

Accountable
Resource
Service
Physical Resource
Application Resource (digital map, video, e-journal...)
Grid Application services accesses application resources through core services
Grid Core Services (information service, Data service ...)
Grid Application Service (Physics analysis, search engine...)
Computational resources (CPU cycles, memory WCT ...)
Storage resources (disk spaces, database, cache)
Network resources (byte transferred, connection duration)

Figure 3.1: Classification of accountable resources in the Grid

Accounting in Grid is similar to UNIX accounting except the heterogeneity of underlying resources and large scale. As illustrated in figure 3.1, accountable resources in a Grid system can be classified into two main categories: the resource and services. Resource accounting is a process that meters and logs usage of physical resources or application-specific resources such as e-journals, digital maps, etc. According to the types of physical resources, resource accounting can be further divided into CPU accounting, storage accounting and network accounting. Service accounting is a process that meters and logs usage of logical services. In an OGSA-compatible system, Grid resources are accessible through OGSA core services. A domain-specific application may define custom services and consume Grid resources through OGSA service interfaces. A map searching service may, for example, enrol multiple computing resources to perform the rendering tasks in parallel through the OGSA-EMS service. Service accounting, in this sense, involves a collection of individual resource usage during the transaction of a particular application service. Based on the classification, the concept of Grid accounting in this thesis is defined as:

A process that logs and provides usage information of resources and services shared in the Grid environment to accommodate requirements of stakeholders and end users within a grid community.

During the course of review, it is learned that the concept of Grid accounting is still confusing to many, particularly its difference from Grid monitoring service, since both services share many common characteristics. First of all, both Grid accounting and monitoring services act at collective layer that provides VO view of Grid resource usage. Moreover both services require gathering and reporting resource usage statistics to enrich system administrator’s understanding of Grid resource usage status. Finally Grid accounting and monitoring services can both used for intrusion detection, auditing, system performance tuning, etc.

However Grid accounting and monitoring services differs in many aspects. Generally speaking, Grid accounting and monitoring services are two different services with different purposes. For monitoring service, its major goal is to provide a view of status of Grid resources, such as current system load, the number of running jobs, job status, etc. Accounting service on the other hand is mainly used for provisioning historic statistics of Grid resource usage as a basis mainly for charging and billing purposes. The main consumers of a monitoring system in the context of OGSA include EMS and Information Service. Monitoring data therefore are resource-centric and require minimal delay to ensure up-to-date resource status information for EMS, for example, to make quick decision where a job should be placed. Compared to monitoring data, accounting data encapsulate more information than resource usage, such as user information, VO information and other event information related to a transaction. In another word, an accounting record is composed of various pieces of information after events, therefore reasonable delay is acceptable. However accounting data need to be as accurate as possible, while small numerical errors and inaccuracy of monitoring data can be tolerant. For example, the CPU utilisation at 70% or 75% may not quite different for EMS to make a decision on job scheduling. Finally considering its timing essence, monitoring data has limited lifetime and does not need to be persistent in database, while historic accounting data are important to be stored safely for economic reporting and auditing purposes. These fundamental differences between Grid monitoring and accounting services are summarised in Table 3-1.

Table 3-1: Comparisons between Grid monitoring and accounting

	Monitoring	**Accounting**
Purpose	To monitor system status, debugging, system profiling, etc.	To keep track of Grid resource usage.
Consumer	System administrator, EMS, Information Service, etc.	VO members, Economic services, etc.
Data delay	LOW	HIGH
Date accuracy	LOW	HIGH
Data persistence	NO	YES

3.2 Usage Scenarios

In order to identify common requirements and issues of an Grid accounting service, the author spent three month to review current practices on developing Grid accounting systems by interviewing stakeholders from various groups, including national Grid service, campus Grid services, regional Grid services, Grid software providers, solution deployers, standard bodies, and end users. The interview was conducted through visits, teleconferences, email, and via a questionnaire. Feedback has been received from over forty people, and summarised in Appendix A. Based on the interview results, there are four common usage scenarios were identified and are discussed in this section along with stakeholder's interests or requirements.

Individual use scenario summarised in this section is structured with a template composed of following three main elements:

- *Description*: a domain-specific description that briefly describes the high-level overview of the scenario.
- *Actors & Goal*: enumerating entities, including human users, organisations and software agents, which play a role in the scenario and their goals.
- *Stakeholders and Interests*: enumerating stakeholders and their interests in the scenario.

3.2.1 Statistical Usage Reporting

Description

GridPP [135] is a collaboration of particle physicists and computer scientists from the UK and CERN, with distributed compute resources spanning 17 UK institutions. GridPP is also the UK's contribution to WLCG project, overseeing the Tier 1 facility at Rutherford Appleton Laboratory (RAL) and the Tier 2 organisations including ScotGrid, NorthGrid, London and SouthGrid. WLCG is a production-level Grid and GridPP has a contractual obligation to provide resource usage data as part of the WLCG project. At present over 200 sites worldwide provide resource usage data to the Grid Operations Centre (GOC) at RAL making aggregation and generates usage statistics.

Actors & Goals

The WLCG Grid operation manager is the main actor for this scenario in the context of system administration. A Grid operational manager is responsible for ensuring fairness and effectiveness of Grid-wide resource utilisation by reviewing usage statistics of resources shared in the Grid.

Stakeholders & Interests

Stakeholders in this scenario include VO managers (e.g. GridPP), resource providers, and end users. From the perspective of resource providers, site-specific resource usage is required to understand how hosted resources are being used, whether they are underutilised or over-exploited for example. At the resource consumer's side, VO managers are interested in reviewing resource usage statistics at VO level, and make sure there are enough resources allocated to accomplish project tasks. A VO manager is also required to review resource usage on a per user basis to prevent allocated resources from malicious usages. Finally, VO members or users are interested in reviewing a summary usage report periodically.

3.2.2 Usage Policing

Description

The National Grid Service (NGS) in UK aims to provide computational and data based resources and facilities to UK researchers, independent of resource or researcher location. This is currently achieved using resources (both compute and data) at four core sites (RAL, Oxford, Leeds and Manchester), and a growing number of partner and affiliate sites, together with the provision of software and services, to enable a consistent method of access to any resource from any location. As fixed resource quotas are granted to a number of non-profitable e-Science projects, it is essential there is a reliable mechanism to account for all aspect of use and enforce usage policing by comparing actual resource usage against allocated quota.

Actor and Goals

The main actors of the usage policing scenario are the NGS's Execution Management Service (EMS) and user account management service. As policing-enabled Grid environment, each user has a registered account associated with granted quota and used quota. The NGS EMS is required to verify the availability of enough quota by comparing remaining quota against historic usage statistics on a per-request and per-user basis. Once a user runs out of the granted quota, the user account management service is triggered to block the user account and send a notification email to the user.

Stakeholders and Interests

Major stakeholders of the NGS scenario in the context of Grid accounting are Grid Operation Support Centre (GOSC), VO manager and end users. There are limited resource quotas allocated to large project as VOs or individual users for education purposes. There resource consumers are interested in knowing how much resource quotas are allocated, being used, and remaining. The GOSC is also required to be aware of resource utilization status and user activities for management purposes.

3.2.3 Grid Economy

Description

Development of accounting systems contributes to the adoption of Grid technologies by industry and the emergence of Business Grids, resources of which are intended to be utilised in a “pay-and-run” manner. In order to enable economic compensation, it is necessary to have other facilities for pricing, charging and billing based on resource usage data generated by accounting systems. The process of accounting together with other economic activities is collectively known as economic accounting.

Actors and Goals

There are three main actors in the Grid economic scenario: the resource management service, pricing and charging service. Compared to traditional resource management services, resource management service within an economy-enabled Grid environment involves an extra process, known as economic authorisation, before allocating resources to service requests. The process can be implemented within an accounting system that estimates resource usage of current service requests and generates resource usage data. Resource management service then checks whether the requestor has enough credits for current request. On completing service execution, the accounting system meters the actual resource usage and generates final resource usage data, which is fed into pricing and charging services for financial transactions.

Stakeholders and Interests

From the commercial perspectives, there are two main stakeholders in the scenario of Grid economy, the resource providers and end users. End users are paying for their computational work to be done or storage capacity to be used. End users therefore are interested in detailed resource usage and charging information of individual paid transactions. Resource providers sell computational resources and storage spaces, and are interested in total resource usage history for making decisions on investing additional resources to increase financial incomes. Resource providers are also interested in profits over a period of time, a financial year for example.

3.2.4 Quality of Service

Description

Current Grid infrastructure operates on a best-effort basis without guaranteed delivered Quality of Service (QoS). Unlike traditional Grid resource management services, which pay more attention to addressing abstraction of management interfaces to low-level and heterogeneous Grid resources, a higher level solution is needed to ensure delivering QoS-enabled services to users, especially for those who have invested a large amount of money. The Service-Level Agreement (SLA) [136] has been considered as the protocol that describes QoS and other business-value commitments by service/resource providers in exchange for financial commitments by consumers against agreed terms, including finishing deadlines, charge and penalties. In order to enable an SLA-oriented management system, resource usage needs to be tracked. This is typically done by an accounting system.

Actor and Goals

The key actor in this scenario is the SLA management system, which aims at performing functions related to the process of agreeing, monitoring and enforcing an SLA between resource providers and consumers. A SLA management system may record the resource usage of a service invocation and optionally constrains and/or charges for the usage. An SLA can contain any number of constraints defined by the service provider, including the placement of usage limits, for instance, maximum amount of CPU time of a particular service invocation. In this case, a SLA management system is required to monitor resource usage status in real time and acts according to service provider policies when usage exceeds a constraint. The real time usage information can be obtained from an accounting system which provides runtime usage accounting facilities. In addition, the cumulative usage, aggregated from all related resource usages, should be reported to the SLA management system by an accounting system on completion of a service invocation.

Stakeholders and Interests

There are two main stakeholders, service consumers and service providers. Detailed service usage information helps service providers to adjust pricing and resource allocation strategies to increase financial incomes. End users pay for services and are interested in knowing how invested money was spent.

3.2.5 Putting Together

These example use case scenarios underlined by Grid accounting services contributed to the vision of Grid economics, provides guaranteed QoS on the pay-per-use basis. The Grid economic model can be built but placing additional layer, upon existing OGSA architecture. This additional layer consists of two main services, the economic services and SLA management services. Economic services provides functionalities related to economic activities, including banking, charging, and billing services, while the SLA management services ensures Grid computing services to be delivered in a QoS-guaranteed manner. In an economic-aware Grid environment, a job submission requires interactions among economic services, SLA management services, accounting services and EMS. An example workflow of a job submission (as Figure 3.2) to economic-aware Grid environment may involves, but not limited to, following steps:

a). A user interacts the SLA management services and instantiates an SLA instance specifying certain QoS metrics and service-level guarantees. Users may also add custom guaranteed terms, such as response time and availability as well.

b). During the SLA instantiation process, SLA management services may need to see whether an agreement can be reached with given user-specified QoS terms and business objectives. This estimation can be implemented by SLA negoation services using simulation tools and applying objective functions, or by the Execution Planning Service (EPS) of EMS (see 2.3.2 for more details).

c). Once an agreement instantiated, it is returned to a user and used as a job submission request to EMS. An SLA instance may specify job runtime specification (i.e. computational, storage, and networking specification), total costs estimated, and other QoS guaranteed terms.

d). EMS then plans, schedules and management the job lifecycle. Before staging a job for execution, the EMC need to perform economic authorisation to make sure the user has enough credits to run the job, and reserve the estimated costs from the user's account.

e). On the completion of the job, a job usage record is generated and fulfilled with the actual resource usage information.

f). The accounted resource usage is then fed into economic services for charging and billing purpose.

g). A user then can view the billing information through economic services.

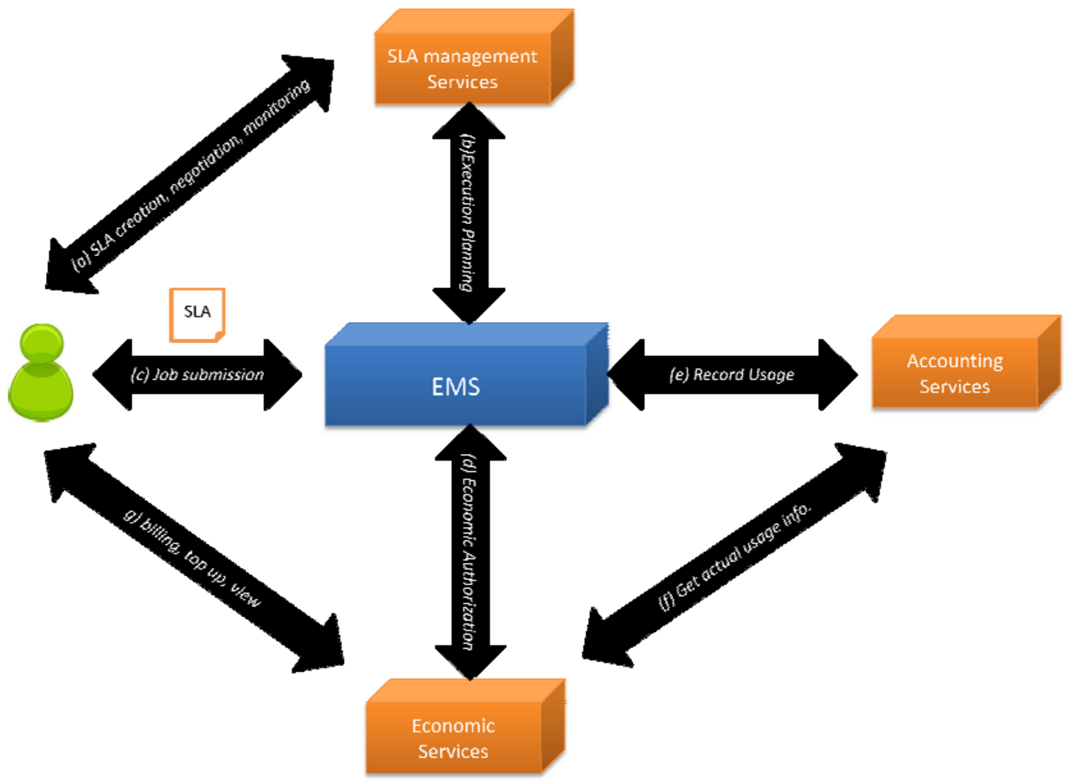


Figure 3.2: Job submission workflow of economic-aware Grid environment

3.3 Accounting Model

As illustrated in Figure 3.3, the Grid accounting process commences from metering and logging usage information of a particular resource or service. These pieces of usage information are then fed into the collection process and composed into well-formatted usage records.

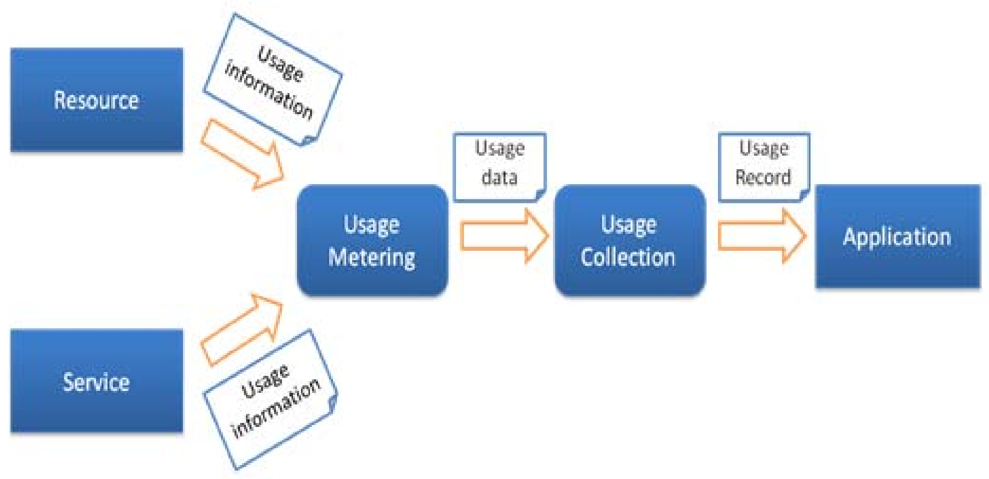


Figure 3.3: Abstract Accounting Processing Model

3.3.1 Usage Metering

As discussed in section 3.1, Grid accounting can be roughly divided into two categories, resource accounting and service accounting. Since resource accounting is resource-oriented, it is possible to define standard measurable metrics of a specific type of resources, such as CPU cycle time of computational resources and disk spaces of storage resources. The standardisation of usage metrics is helpful to ensure data interoperability between different accounting systems. Service accounting differentiates from resource accounting in that it is domain-specific. Metric definitions of a specific application domain are most likely to be different from definitions of another. Besides, service providers of an application domain may specify various usage metrics according to local accounting purposes. In this sense, service provider may define different metrics of services of same application domain making it hard to standardise service accounting metrics.

The metering process can be triggered in two patterns: the passive pattern and active pattern. As with usage scenarios of “usage policing” (section 3.2) and “QoS-enabled resource management” (section 3.4), usage information is required to be metered in real time during resource utilisation or service invocation. Under this circumstance, the metering process of an accounting system is triggered by high-level services, therefore known as the **passive metering**. For other cases when real-time usage information is not critical, metering process can be scheduled to parse resource/service usage actively during a period of time. This pattern of usage metering, known as **active pattern,** periodically scans resource/service usage information by parsing system log files.

3.3.2 Usage Collection

Once usage has been metered, pieces of usage information are to be gathered by the collection process and formatted as usage records. A usage record is a well-formatted representation consisting of a list of usage metrics targeting a particular Unit of Work (UoW), ranging from finest-grained batch jobs to coarse-grained service invocations. The collection process at coarse-grained level involves an extra aggregation process, which summarises usage records of atomic batch jobs related to the service invocation.

As metering process, the collection process has two accordingly process patterns as well. Aligned with active metering process, collection process can be scheduled in as a “cron” job, which periodically consumes the output of metering process and generates usage records. Active collection process normally involves a separate data persistence layer that saves usage records. Alternatively, the collection process can be invoked passively by high-level applications to generate usage record in real time. The passive collection process caches usage records in memory only.

3.3.3 Classification

Based on two dimensional factors, the triggering pattern of the metering process and granularity of UoW, accounting models can be classified into four categories (Figure 3.4) as follows:

- **Fine-grained active accounting**
  In the fine-grained active accounting model, the metering process is scheduled to periodically parse and generate usage records at atomic UoW level.
- **Fine-grained passive accounting model**
  The metering process of the fine-grained passive accounting model is triggered by a third party to generate usage records at UoW level. For example, a user may be interested in knowing the current resource usage status of a long-running job to ensure there is enough quota left until job completion.
- **Aggregate active accounting model**
  The aggregate active accounting model automatically meters usage information of all UoWs, both completed and running UoWs, and generates summarised usage records only.
- **Aggregate passive accounting model**
  The aggregate passive accounting model generates summarised usage records only when a high-level request triggers the metering process.

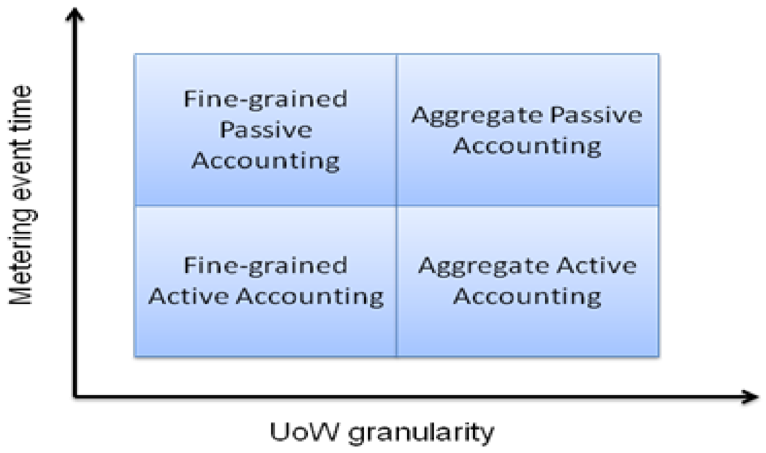

Metering event time
Fine-grained Passive Accounting
Aggregate Passive Accounting
Fine-grained Active Accounting
Aggregate Active Accounting
UoW granularity

Figure 3.4: Accounting model classification

3.4 Standards

There are two accounting-related standards proposed by OGF Usage Record and Resource Usage Service working groups to ensure data and service interoperability between accounting systems.

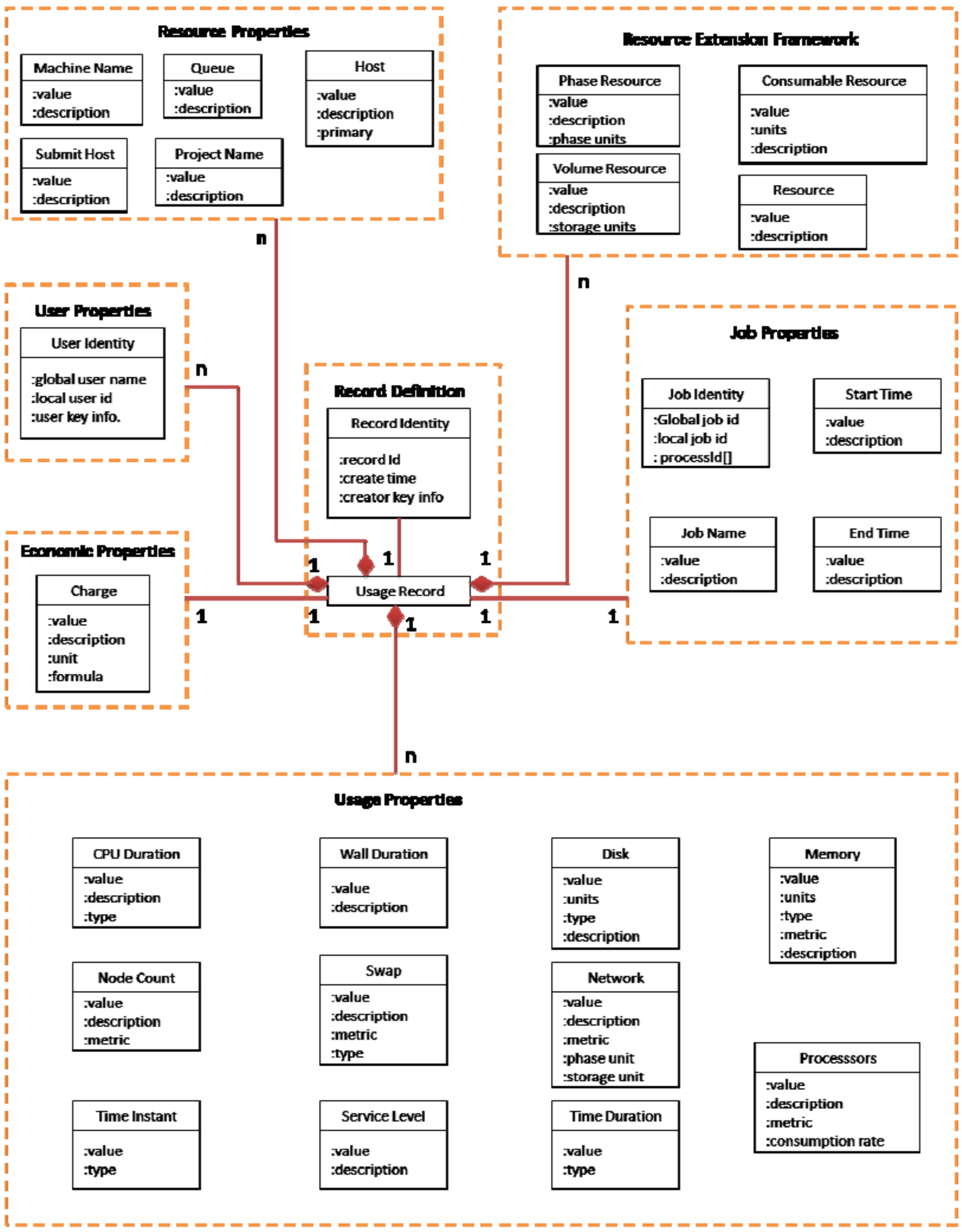


Figure 3.5: OGF Usage Record Information Model

3.4.1 Usage Record Format

As discussed in section 3.3, usage metric definitions vary from accounting systems to accounting systems depending on local deployment requirements and local accounting polices specific resource or service providers. In order to enable data interoperability among independently developed accounting systems, extensive work has been done by OGF usage record working group on defining standard usage metrics and representation format. In 2003, a usage record (UR) format recommendation specification [137] was released and defines a set of well-defined usage metrics and XML format for representation of computational usage of a single batch job. From the information model demonstrated in figure 3.5, the usage metrics defined within UR consists of batch job properties, job owner or user properties, resource properties, computing related usage properties, economic properties, and an extension framework for definitions of custom metrics or properties. These usage metrics are collectively to be represented as a single usage record, with a global unique record identity and other common properties, such as creation timestamp and creator of the usage record.

3.4.2 Resource Usage Service

Another accounting-related draft specification, the Resource Usage Service (RUS) proposed by OGF RUS working group, enables service-level interoperability between different accounting systems through a set of core Web service interfaces. These service interface definitions enable sharing and manipulation of standard OGF UR instances in a standard manner. Rather than providing a monolithic solution to Grid accounting, the RUS is intended to be implemented to support either active or passive accounting models. Since current RUS specification depends on the OGF UR standard, it only allows accounting at atomic level, the batch-job level.

Apart from core functionalities as defined in current RUS specification (version 1.7) [138], the RUS working group has a clear roadmap (figure 3.6) for advanced features including server-side aggregation and hierarchical deployment. It is expected that these advanced features would enable four accounting models and resource/service accounting in a standard manner.

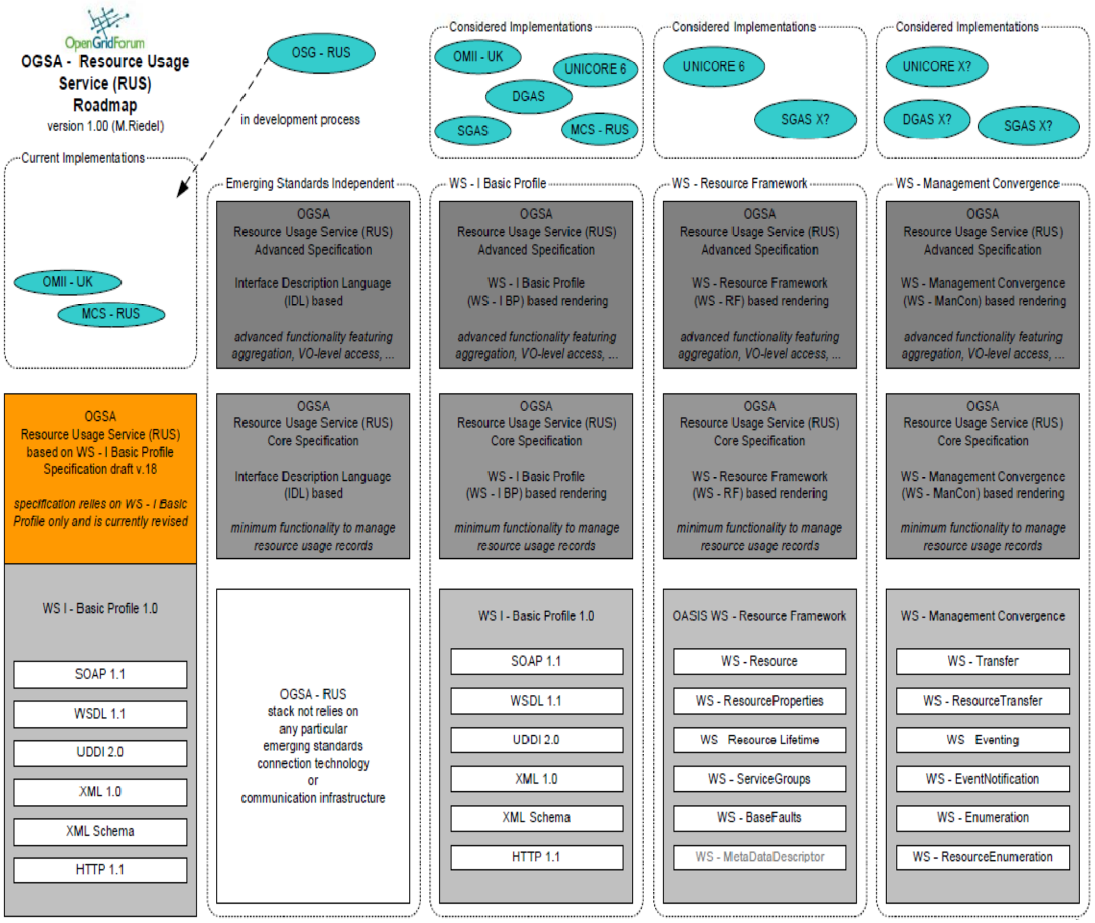

OpenGridForum
OGSA - Resource Usage Service (RUS) Roadmap
version 1.00 (M.Riedel)
OSG - RUS
in development process
Current Implementations
OMII - UK
MCS - RUS
Considered Implementations
OMII - UK
UNICORE 6
DGAS
SGAS
MCS - RUS
Considered Implementations
UNICORE 6
SGAS X?
Considered Implementations
UNICORE X?
DGAS X?
SGAS X?
Emerging Standards Independent
WS - I Basic Profile
WS - Resource Framework
WS - Management Convergence
OGSA Resource Usage Service (RUS) Advanced Specification
Interface Description Language (IDL) based
advanced functionality featuring aggregation, VO-level access, ...
WS - I Basic Profile (WS - I BP) based rendering
WS - Resource Framework (WS - RF) based rendering
WS - Management Convergence (WS - ManCon) based rendering
OGSA Resource Usage Service (RUS) based on WS - I Basic Profile Specification draft v.18
specification relies on WS - I Basic Profile only and is currently revised
OGSA Resource Usage Service (RUS) Core Specification
minimum functionality to manage resource usage records
WS I - Basic Profile 1.0
SOAP 1.1
WSDL 1.1
UDDI 2.0
XML 1.0
XML Schema
HTTP 1.1
OGSA - RUS stack not relies on any particular emerging standards connection technology or communication infrastructure
OASIS WS - Resource Framework
WS - Resource
WS - ResourceProperties
WS Resource Lifetime
WS - ServiceGroups
WS - BaseFaults
WS - MetaDataDescriptor
WS - Management Convergence
WS - Transfer
WS - ResourceTransfer
WS Eventing
WS - EventNotification
WS - Enumeration
WS - ResourceEnumeration

Figure 3.6: OGF-RUS Standardization Roadmap, from [138]

3.5 Accounting Systems

There are many operational grids having accounting systems developed and deployed, some of which are standard compatible while others provide custom solutions. The interoperability, however, has received increasing importance in accounting among grid environments, and contributed to more and more standard non-compatible solutions transiting to be standard compatible. A list of accounting systems (Table 3-2) developed by production Grid projects is reviewed in this section.

Table 3-2: A List of Accounting Systems of Production Grid Projects

Name	**Project**	**Description**	Affiliation
APEL	EGEE/WLCG	An accounting tool used in the LCG project, and is a part of the gLite middleware	STFC RAL
DGAS	EGEE	DGAS (Distributed Grid Accounting System) previously known as the DataGrid accounting system was developed within the EU DataGrid project and is currently being re-engineered by EGEE and OMII-Europe.	Istituto Nazionale di Fisica Nucleare (INFN)
SGAS	SweGrid	SGAS (SweGrid Accounting System), developed for SweGrid, is a Java implementation based on OGSA architecture that is now integrated as a Grid service in Globus Toolkit 4. SGAS has been used in NorduGrid as a standard accounting service.	The Royal Institute of Technology (KTH)
UNICORE Accounting service	UNICORE	The UNICORE accounting system is an OMII-Europe component that provides a WS-RF compatible RUS implementation for real-time usage monitoring.	Forschungszentrum Juelich-FZJ
Gratia	Open Science Grid	Gratia is the Grid accounting system being developed for Open Science Grid, a scientific Grid project funded by National Science Foundation	SLAC National Accelerator Laboratory and Fermilab
User Accounting System	UK National Grid Service	An accounting service developed by UK National Grid Service project for reporting resource usage at user level.	Manchester University

3.5.1 User Accounting System

The User Accounting System (UAS) [139] deployed within most National Grid Service (NGS) sites in UK was originally designed for the Market for Computation Service (MCS) project [140]. The UAS aims at metering and collection of usage information from computational centres around UK.

As illustrated in figure 3.7, the system is composed of two major components for usage metering, the Batch2UR and JBMDB, both of which reside at resource provider site. The JBMDB module is deployed as a “cron” job and scheduled to generate global job-user identity mapping information daily by parsing GRAM log files. Batch2UR component is deployed at Local Resource Management System (LRMS) node and meters usage information on completion of a batch job and compose OGF URF instances that are then fed into the centralised RUS service instance running at Manchester site, through RUS client interfaces. The RUS service also renders and store received URF instances into Oracle Relational Database Management System (RDBMS) with custom relational data model. Metric mappings of this relational data model are given in the Table A-1 of Appendix A. The data schema which are summarized and synchronised on per user basis to Oracle database maintain by Grid Operation Service Centre (GOSC) at Rutherford through Oracle synchronisation protocol. User summary usage information is used to enforce usage policing against allocated quota. A Web portal is also provided and allows user to query how much quota remains so that users can plan resource usage before job submissions.

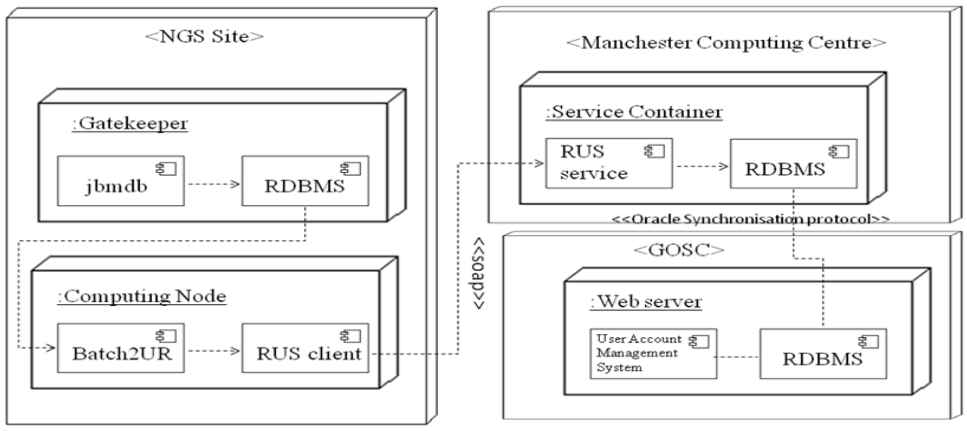


Figure 3.7: NGS User Accounting System Deployment Diagram

3.5.2 Accounting Processor for Event Logs

Accounting Processor for Event Logs (APEL) [141] is the accounting system developed by the WLCG project, and aims at streaming metered resource usage information from participant site to Grid Operation Centre (GOC) at Rutherford Appleton Laboratory (RAL), where an aggregation process is enforced for reporting resource usage statistics on per VO, per site, and per month basis.

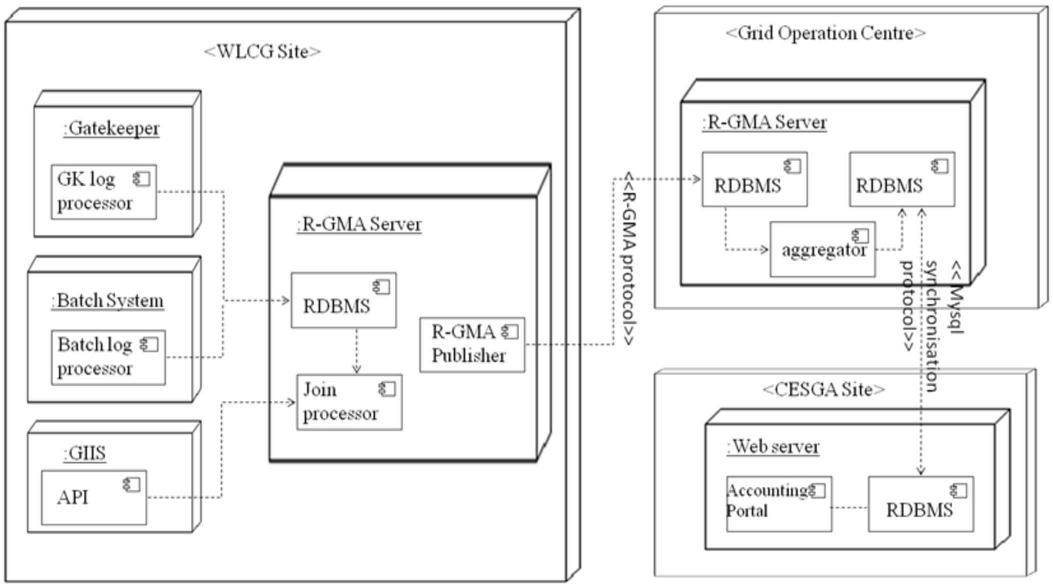


Figure 3.8: WLCG Accounting Processor for Event Logs (APEL) System Deployment Diagram

As illustrated in Figure 3.8, APEL system comprises a variety of log processors, which are scheduled as “cron” jobs and aims at meter resource usage by parsing log files produced by different runtime components, batch systems and Globus gatekeeper in particular. A site-level Relational Grid Monitoring Architecture (R-GMA) [142] server is also deployed at site level to cache metered usage data and compose usage records on per batch job basis in WLCG accounting schema [143] by a lightweight process, the join processor. Metric mappings between WLCG accounting schema and standard OGF-UR schema are outlined as illustrated in Table A-2 at Appendix A. The join processor is also required to contact a site-level information service, the Grid Information Index Service (GIIS), to look up performance for the computational resources where jobs were executed.

This performance information is to be used for normalisation and is of particular importance when dealing with VO applications that run over heterogeneous resources. Job usage records of a particular site are then published through R-GMA protocol and archived in a centralised relational database maintained at the Grid Operation Centre in Rutherford Appleton Laboratory (RAL), where job usage records are aggregated to a separate summary usage database. Aggregate usage information is synchronised to database at Centro de Supercomputación de Galicia (CESGA) site in Spain and accessible by end users through a graphic front-end Web portal.

3.5.3 Distributed Grid Accounting System

Distributed Grid Accounting System (DGAS) [144] is another grid accounting tool developed by EGEE project and widely deployed at participants sites involved in both EGEE and WLCG projects. DGAS is targeted at providing job-level resource usage metering in a client/server infrastructure.

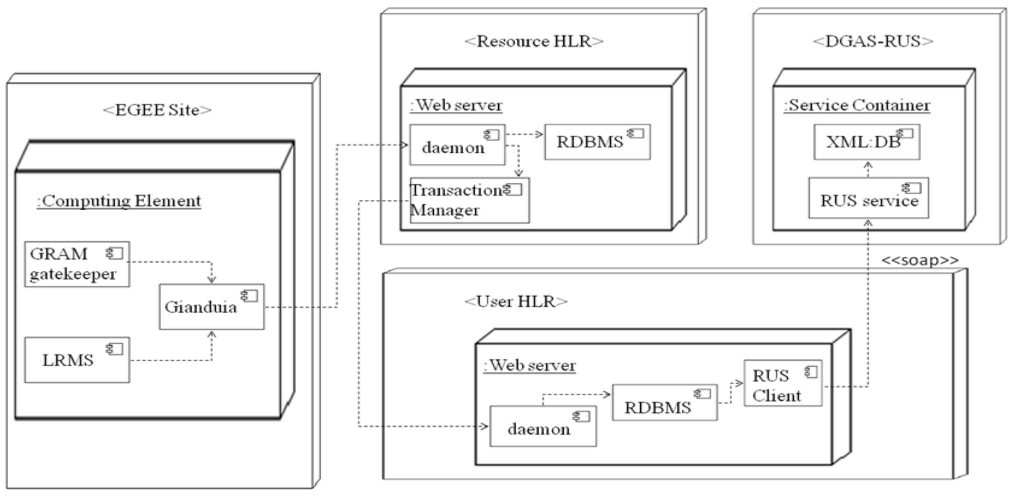


Figure 3.9: Distributed Grid Accounting System Deployment Diagram

The accounting process of DGAS is enforced by two main components, as demonstrated in Figure 3.9, the lightweight usage meter, Gianduia, and the distributed Home Location Registry (HLR), which acts as a repository for usage information related to registered users or resources. Each site has a Gianduia meter deployed and publishes metered usage information to a registered resource HLR, from which usage information can be retrieved for both individual jobs and in aggregate/summary form on per CE basis. Metric mappings between relational accounting schema of resource HLR and standard OGF-UR schema are given in table A-3 of Appendix A. A transaction manager keeps listening to incoming job usage records and is triggered to forward resource-specific job usage records to User HLR, where additional user information is to be added into job usage records. It is also understood that a preliminary RUS prototype, known as DGAS-RUS [145], is being developed for the DGAS system. The RUS interface will enable insertion and persistence usage records from user HLR through RUS client interface into a centralised XML database.

3.5.4 SweGrid Accounting System

The SweGrid Accounting System (SGAS) [146] is an accounting system developed the national Grid test-bed in Sweden, and has been integrated as accounting service of Globus Toolkit.

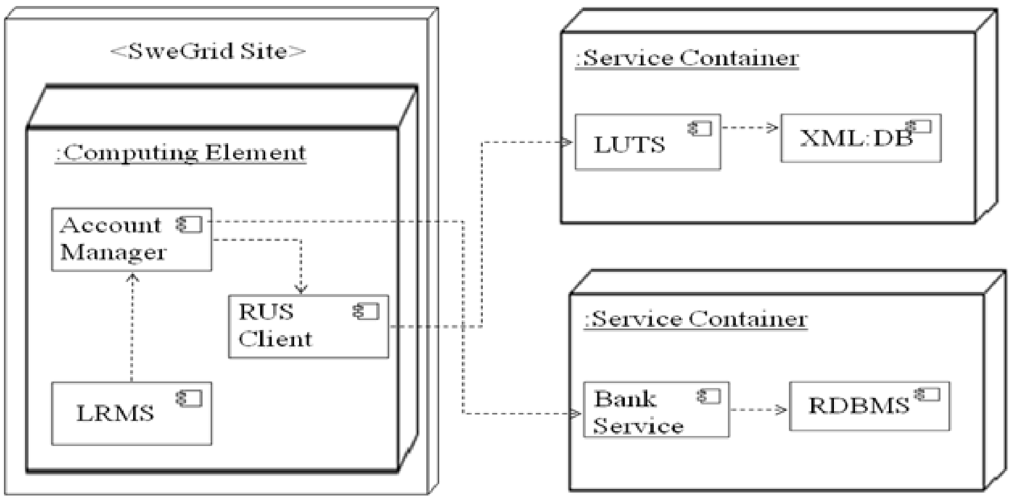


Figure 3.10: SweGrid Accounting System Deployment Diagram

As shown in Figure 3.10, the usage metering is realized through the Job Account Resource Management (JARM) component, which is responsible for providing the accounting system with information from local batch systems. Each user requires a valid account with credits in a banking service. When submitting a job, the JARM computes a maximum cost and reserves that amount of credit on the user's account through the banking service. On completion, the JARM reports the actual resource consumption in the form of a usage record and the associated charge is made to the user's account. The usage record is then populated into the Logging and Usage Tracing Service (LUTS), a RUS instance, for centralized storage. Any query on job usage information is directly sent to LUTS via an authorization service that protects usage data from invalid access. The SGAS exhibits a full standard-compatible solution for Grid accounting. The only extension, as Appendix A.2.3, to URF proposed within SGAS also highlights the importance of VO information.

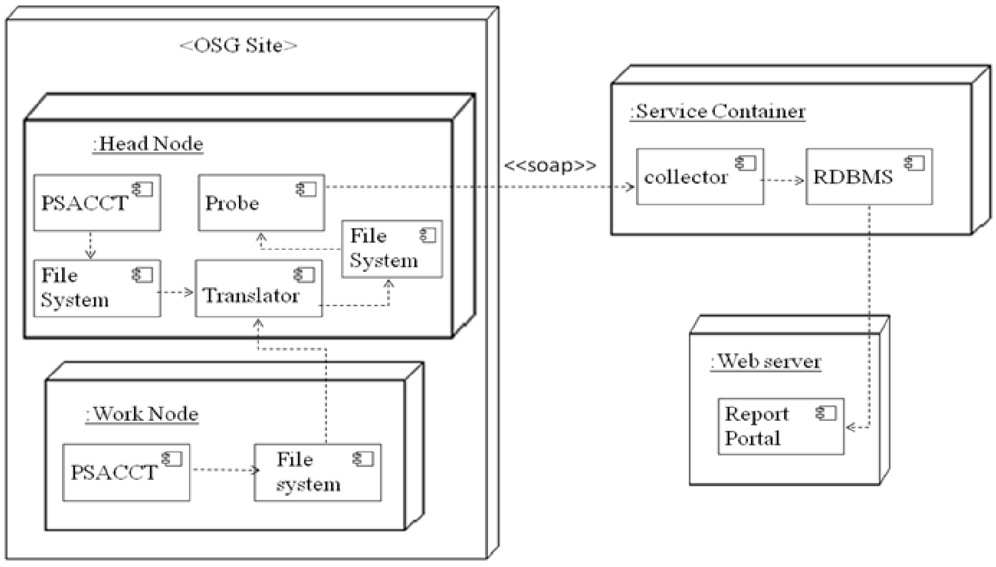


Figure 3.11: Gratia Accounting System Deployment Diagram

3.5.5 Gratia

The Gratia [147] is the accounting system being developed within OSG project. The current implementation of Gratia accounting system is composed of three functional components as illustrated in figure 3.11: the probe, collector and publisher. Usage information of a cluster is kept being logged by a utility script, the PSACCT, for monitoring process activities. At head node, a translator process is running periodically and merges log information of both head and work nodes into complete usage records, which are fed into the probe component and published into remote collector machine through well-defined Web service interfaces. These usage records are stored centrally in a relational database of the collector machine. Metric mappings between Gratia accounting schema and standard OGF-UR schema are listed in table A-4 of Appendix A.

3.5.6 UINCORE Accounting System

The accounting system in the UNICORE project provides a RUS implementation based on WSRF profile [50]. The RUS is integrated within UNICORE infrastructure aiming at exposing usage records generated at the batch system level in real time.

As shown in figure 3.12, the UNICORE accounting system is composed of two components: the URF generator and RUS endpoint. A graphic front-end client, LLView [148], is provided for users to get real-time site-level resource usage on demand. Once triggered, the LLView client interacts with RUS service endpoint and query through the "*RUS::extractUsageRecord*" interface. Rather than maintaining persistent resource usage information, the RUS service endpoint interrogates the usage record generator and returns resource usage information of queued and active batch jobs. Since current RUS implementation however does not enable data persistence, it is not possible to provide historic usage statistics.

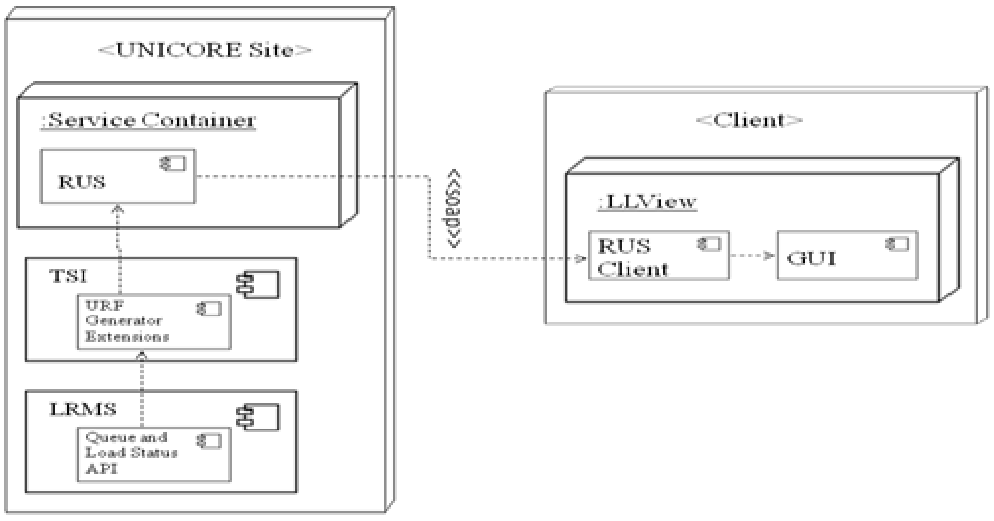
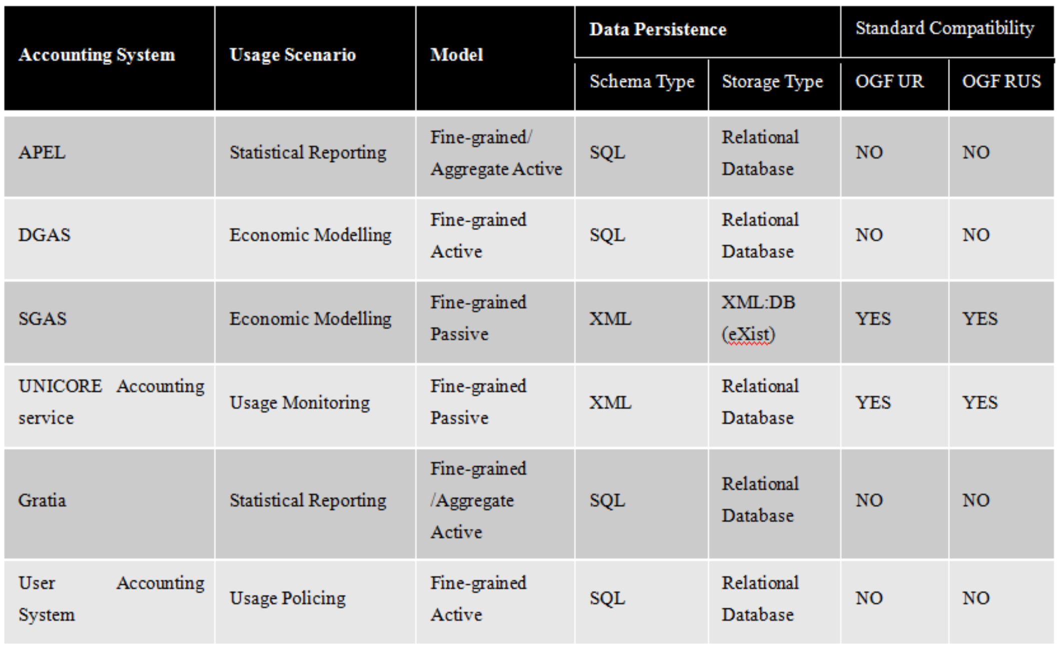


Figure 3.12: UNICORE Accounting System Deployment Diagram

3.5.7 Comparison

As detailed comparisons summarised in Table 3-3, existing accounting systems employed in the production Grid projects are implemented in heterogenous ways with project-specific purposes. Most usages of these accounting systems fall into the four usage scenarios discussed in section 3.2, except the UNICORE accounting system which is used for site-level usage monitoring purposes. Although some accounting systems are used for the scenario, for example both DGAS and SGAS designed for realisation of Grid economic model, theire accounting process are different from each other. DGAS uses active metering pattern that parses job usage information mainly for charging and billing purposes, while the metering process of SGAS is triggered by EMS to perform economic authorisation before staging a job request to local resources. Besides these accounting system uses different data presentation format and data persistence strategies.

Table 3-3: Comparison of Grid Accounting Tools Employed In Production Grids



Accounting System	Usage Scenario	Model	Data Persistence		Standard Compatibility	
			Schema Type	Storage Type	OGF UR	OGF RUS
APEL	Statistical Reporting	Fine-grained/ Aggregate Active	SQL	Relational Database	NO	NO
DGAS	Economic Modelling	Fine-grained Active	SQL	Relational Database	NO	NO
SGAS	Economic Modelling	Fine-grained Passive	XML	XML:DB (eXist)	YES	YES
UNICORE Accounting service	Usage Monitoring	Fine-grained Passive	XML	Relational Database	YES	YES
Gratia	Statistical Reporting	Fine-grained /Aggregate Active	SQL	Relational Database	NO	NO
User Accounting System	Usage Policing	Fine-grained Active	SQL	Relational Database	NO	NO

Given their heterogeneous essence, it is hard for these accounting systems to interoperate with each other to fulfill the requirements on sharing accounting data across Grid infrastructures, unless two accounting systems exhibt common service interfaces and exchange accounting data in a common format. Standardisation therefore is of increasing importance in this sense. However standardisation is a time-consuming process because it is difficult to define a single standard to accommodate various and evolving accounting requirements. Standardisation is further complicated by conerns from additional re-engineering tasks while not breaking existing accounting processes. At the end of the review, there are only two accounting systems, SGAS and UNICORE accounting system, which provide standard compatible solutions to both OGF UR and OGF RUS.

3.5.8 Others

Having recognised the importance of accounting service in Grid systems, there are many commercial Grid products that have custom accounting solutions implemented. The Accounting and Reporting Console (ARCo) [149] of Sun Grid Engine (SGE) [150], for example, enables users to gather live reporting data from the SGE as well as storing accounting data for historic analysis in the reporting database. Besides aforementioned accounting systems developed for large-scale distributed Grid systems, there are also many cluster and High Performance Computing (HPC) systems that have accounting systems embedded. Such examples as Gold Allocation Manager [151] is an open source accounting system designed to dynamically interact LRMS to provide job quotations at job submission time, hold on accounts during job execution, and charge on completion of jobs according to actual resource usage. SAFE is another example accounting tool developed by EPCC for accounting purposes of national HPC services HPCx [152] and HECToR [153] as well as local EPCC machines. A Java-based web interface to SAFE provides graphical usage monitoring and allows Principal Investigators to administer their projects' users and resources.

3.6 A Generic Accounting Framework

Based on reviews of existing accounting tools, both standards compatible and incompatible, there are several common issues identified. First of all, these accounting systems are implemented in a Grid or project-specific manner, making it hard to be reused across project domains. Interoperability is another challenge in the sense of lacking a standard way of mapping custom usage metrics to those standardised within the OGF-UR schema. Custom metric definitions using OGF UR extension framework further complicates the interoperability issue. As Table A-5 given in Appendix A, most accounting systems have similar metric extensions defined in different way. Although there are some standard-compatible accounting solutions available, such as SGAS, UNICORE accounting system and preliminary implementation of DGAS-RUS, a native XML database is widely used to save OGF usage records instances natively, making it hard to implement a standard compatible accounting solution especially for those that use relational database for data persistence. There are also several other non-functional issues that should be considered when developing an accounting system for large-scale distributed systems as the Grid, including responsiveness, flexibility, fault tolerance, and security.

In order to avoid duplicate efforts and provide an integrated and widely adopted approach to accounting in real production Grids, a generic accounting framework is proposed to JISC as one of the outputs of our review efforts described in this chapter. As Figure 3.13, the proposed framework abstracts basic functionalities of an accounting tool based on a Client/Server (C/S) infrastructure.

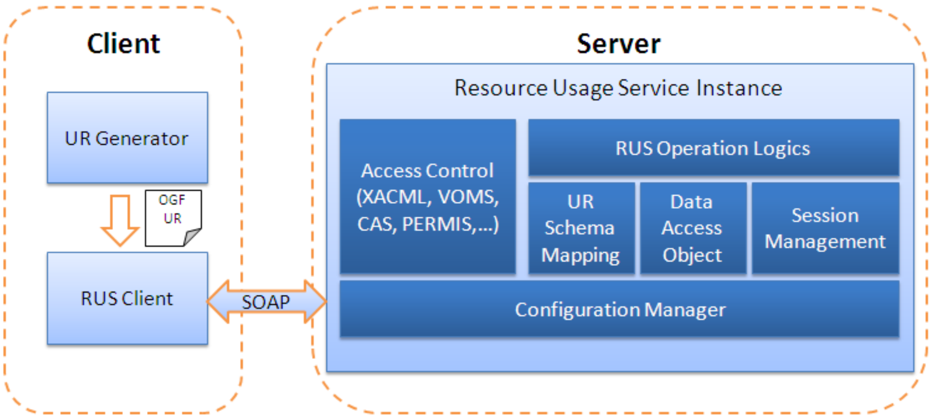


Figure 3.13: Generic Accounting Framework (Component Architecture)

At the client-side, a general-purpose UR generator component is defined and used to meter accounting metrics and compose accounting data in standard UR format. The UR generator component exhibits an abstract layer and allows different implementations upon usage meters of underlying systems. Accounting data instances are then streamed into a RUS service endpoint through RUS client interfaces.

The RUS service endpoint at server side consists of a set of abstract functional components to be added as required within an RUS implementation. The access control module acts as a gateway to RUS logics and protects accounting data from unauthorised accesses. A RUS service must provide an implementation of this module and apply local security policies to guarantee data privacy. A RUS implementation may choose to implement one or more RUS logics or operations. By implementing two optional modules, the UR mapping module and Data Access Object (DAO) module, a RUS implementation can be developed without changing existing accounting data model and persistent storage types. In order to ensure system QoS, a session is required to maintain the accumulation of transaction information on per user per transaction basis. A RUS implementation may define their own data structures inside a session for various purposes. When a user query a huge amount of accounting data, for example, the session can be used to maintain an enumeration context and allows user to iterate query results through multiple interactions. A session normally has a limited lifetime. The session management module is thereafter defined and responsible for lifetime management of sessions. Finally the configuration manager component is used to provide configuration facilities for a RUS system.

3.7 Summary

This chapter investigated the philosophy of accounting in the Grid environment, and reviewed the state-of-art standardisation and development efforts on accounting systems of operational Grid projects and commercial Grid products. However, these accounting systems were developed in a variety of ways depending on Grid-specific understanding of accounting and customised high-level usages. Having identified the importance of interoperability for sharing usage data across accounting systems in particular, there are an increasing number of accounting systems being developed or migration to be compatible to OGF UR and OGF RUS standards. Early adoptions of these standards, however, are implemented upon specific accounting systems, making it hard to be reused for others. In order to enable a consistent solution and avoid duplicate efforts, this chapter proposed a generic accounting framework with identified key features, which ensures interoperability while allowing maximum customisation to accommodate local deployment requirements. This proposed accounting framework forms the basis of the rest chapters of this thesis.

Chapter 4

Design of Resource Usage Service for World-wide LHC Grid

According to the Memorandum of Understanding (MoU) [154], participating sites of the World-wide LHC Grid (WLCG) project are required to provide resource usage or accounting information to the Grid Operational Centre (GOC) for the purpose of overall project operation and management. As a composite Grid environment, the accounting process of WLCG is currently empowered by four accounting systems, APEL and DGAS, SGAS and Graita developed by WLCG/EGEE collaborative project, Nordic Data Grid Facilities (NDGF), and Open Science Grid (OSG) project respectively. These project-specific accounting systems were designed and implemented based on project-specific accounting requirements and purposes, therefore lacking interoperability and portability. In order to automate accounting process in WLCG, three transportation methods are being introduced for streaming accounting data metered by Grid-specific accounting system into GOC at Rutherford Appleton Laboratory (RAL) in the UK, where accounting data are aggregated and accumulated throughout the year. These transportation methods, however, are introduced on per accounting system basis, i.e. targeting a particular accounting system, making it hard to customise. This chapter describes a standard-compatible solution, the WLCG-RUS as an alternative method for sharing accounting data, while ensuring interoperability, portability and customisability. Relevant publications related to this chapter have been published in[155][156][157][158].

4.1 Introduction

Accounting activities within WLCG requires collection of accounting data from all participating sites in EGEE and WLCG projects as well as from sites of other collaborating Grid projects into a central accounting database in GOC at RAL. These accounting data are to be processed offline to generate statistical summaries that are reportable through EGEE/WLCG accounting portal[159]. There are two main accounting processes introduced within EGEE/WLCG accounting framework[160]: the job accounting and aggregate accounting. The job accounting process generates accounting records describing the resources consumed by a single executing job. Job accounting records are composed at sites and streamed into a central database at GOC, where offline aggregate processes take effect to summarise resource usage consumed by a collection of jobs. These two types of accounting processes fall into categories of the "fine-grained active accounting" and "aggregate active accounting" models as classified in section 3.3.3.

Job accounting process in WLCG is mainly enforced by accounting systems of EGEE/WLCG and other collaborative Grid projects. These project-specific accounting systems are being deployed at sites to meter and generate accounting records in heterogeneous formats. In order to share job usage records within the GOC centre, there are three transportation methods (Figure 4.1) introduced, each of which was designed to provide accounting system-specific solution. For most EGEE/WLCG sites, APEL[141] has been deployed as one of main accounting systems, which generates accounting records in WLCG accounting schema. The job accounting records metered at sites by APEL therefore can be automatically synchronized to the centralised job accounting database maintain at GOC centre through R-GMA[122] protocol. Another accounting system widely deployed at EGEE/WLCG sites is the DGAS [144], which generates job accounting records in a format different from WLCG accounting schema. Before streaming accounting data to GOC, DGAS accounting records are required to be transformed into WLCG accounting data format. A lightweight component, DGAS2APEL, transforms DGAS accounting records into the WLCG accounting data format and streams them into GOC through R-GMA protocol. The third and most straightforward transportation method is called "direct SQL insertion". Rather than automating data sharing process, this method requires extra administrative efforts to manually populate accounting records by executing Structured Query Language (SQL) insertion statement. The "direct SQL insertion" method has been widely adopted by sites from collaborative Grid projects, OSG and NDGF in particular, to share aggregate accounting data only applying to local security policies. An offline aggregation process is scheduled at GOC and summarises resource usage data daily.

There are three accounting data formats, collectively known as WLCG schemas[143], defined in WLCG for data persistence on relational databases, the WLCG job record schema, anonymous aggregate record schema, which represents summarised resource usage information on per site, per VO, per month, and per year basis, and user aggregate accounting record schema, which represents summarised resource usage information on per site, per VO, per user, per group per role, per month and per year basis.

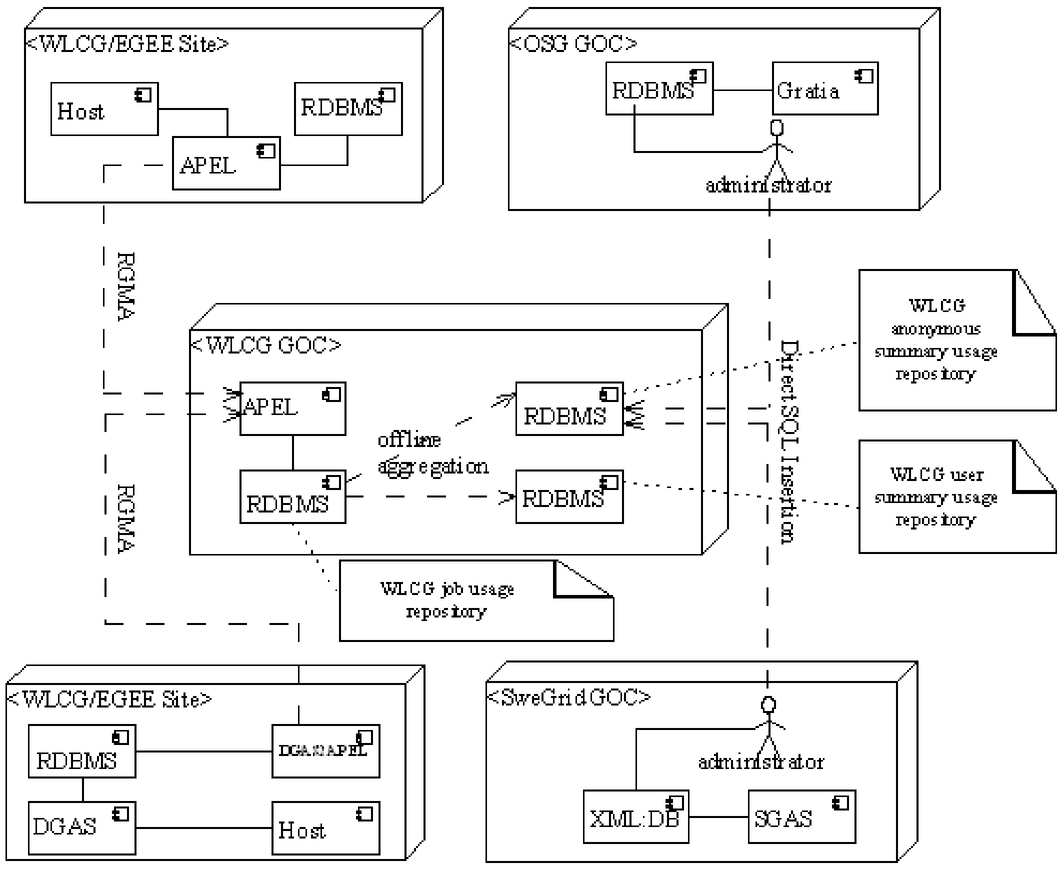


Figure 4.1: Current EGEE/WLCG accounting deployment scenarios with three transportation methods introduced in WLCG accounting

This chapter describe the design and implementation details of WLCG-RUS, as an alternative, but standard-compatible method to share accounting data.

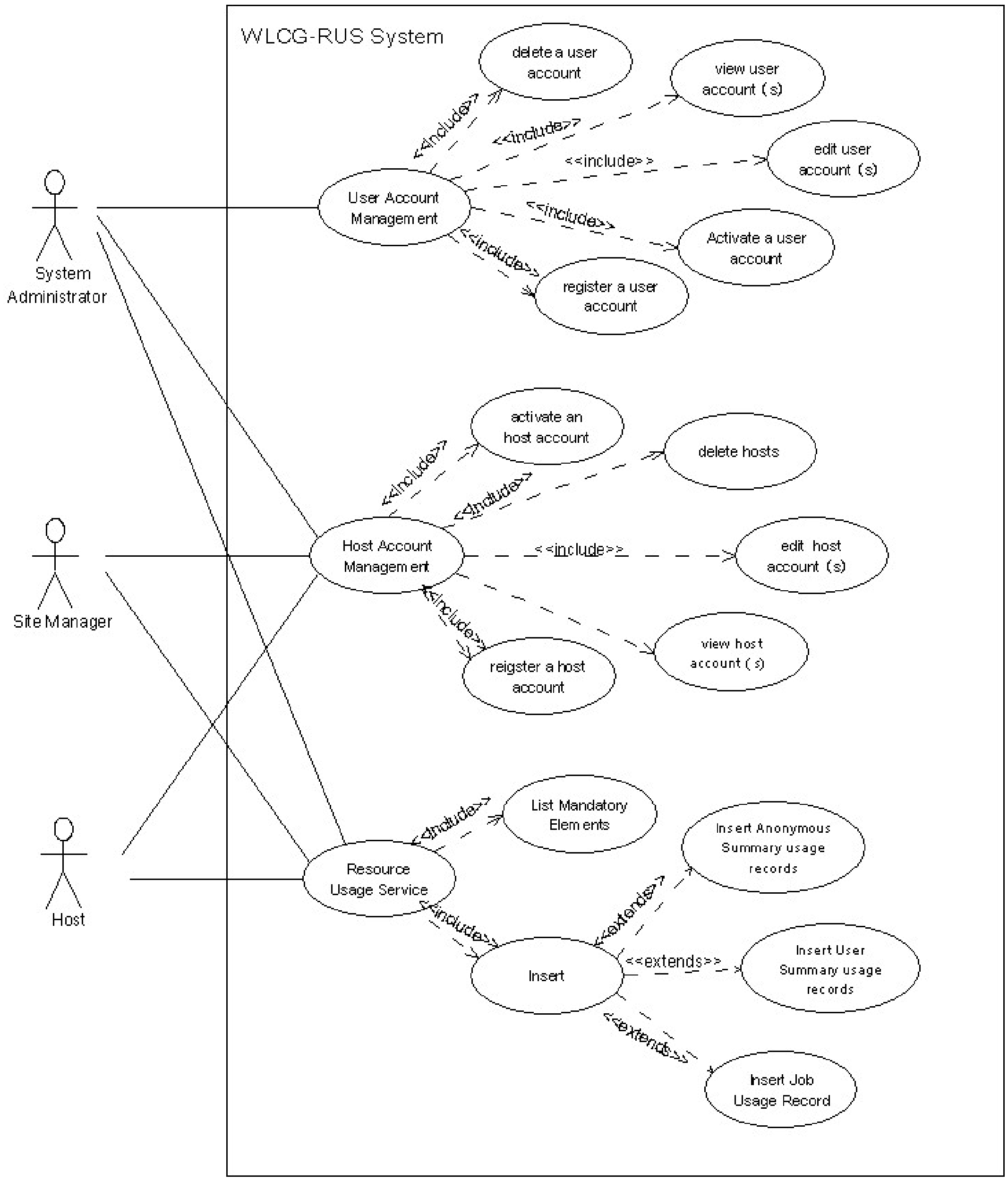


Figure 4.2: The main use cases that the WLCG-RUS is expected to implement in conjunction with the actors generalised from existing WLCG accounting scenarios.

4.2 Requirement Analysis

This section discusses the requirements that shaped the design of the WLCG-RUS system.

4.2.1 Use Cases

In order to identify system design requirements, a use case analysis was carried out based on three generalised roles in the context of existing WLCG accounting: the site manager, site hosts, and system administrator. Detailed use cases are illustrated in Figure 4.2 and listed in Appendix B.1.

Site manager

A site manager is the manager of a participating site, normally an institution or research centre, in the provision of the WLCG with a Tier1 and/or Tier2 computing centre. An actor taking the role of site manager should hold a valid X.509 certificate. A site manger in the context of WLCG-RUS system has the privilege to register one or more hosts to the WLCG RUS system so that these hosts can upload accounting data. A site manger is also able to manage host account through WLCG-RUS interface.

Host

The host is the head node of EGEE/WLCG computing element and holds a valid host certificate signed by a recognised Certificate Authority (CA). A host is able to publish host-specific or site-specific accounting data to WLCG-RUS system only if it has a valid account registered by owned site manager and activated by system administrator.

System Administrator

The system administrator is senior to other roles and takes the responsibility of system management. A system administrator has views and controls over all hosts registered to WLCG-RUS system. Besides, a system administrator takes care of user management and role assignment. Finally a system administrator is required to have administrative rights on WLCG-RUS system configurations.

4.2.2 Requirements

Capability or Functional Requirements

Based upon review of identified use cases, key functional requirements of the WLCG-RUS system are summarised as follows.

1. Data Publishing

The key functionality of WLCG-RUS system is to provide a data publishing mechanism through which participating sites can upload accounting data. The design of data publishing is required to enable both fine-grained at batch job level and aggregate accounting models, to facilitate various data sharing in the context of WLCG accounting. In the case of aggregate accounting model, the aggregation process should be triggered at the same time of data publishing. The design of data publishing facility in WLCG-RUS system is also required to support various aggregation strategies in a customisable manner making it easy to adapt existing WLCG anonymous aggregate strategy, user aggregate strategy and new aggregate strategies.

2. Host Management

From perspective of site managers, the WLCG-RUS system is required to provide host management facilities for host registration, view registered hosts, and edit host profiles.

3. User Account Management

User management is an important functionality for system administrator. The WLCG-RUS system is designed to provide user account management facilities for system administrator to view user registration requests, and grant and revoke privileges to site managers.

Interface Requirements

There are three different interfaces intended to be provided in the WLCG-RUS system, the internal, external and user interfaces. The following provides a list of requirements on interface design.

4. Internal Interface

The WLCG-RUS system must exhibit well-defined internal interfaces for customisation and extensibility, so that new features can be implemented independently and plugged, without affecting fundamental architectural design.

5. External Interface

The external interface is the client-side interface used for hosts uploading accounting data through standard RUS interfaces. In this case, the design of external interface should be command line oriented, in particular scripting language based, so that the uploading process can be automated in a scheduled manner (e.g. cron job).

6. User Interface

The design of the WLCG-RUS system needs to provide user-friendly interfaces for site managers and system administrators to perform management tasks.

Data Requirements

Data representation in the WLCG-RUS system is a two-folded issue. On the one hand WLCG-RUS is intended to be deployed upon existing WLCG accounting data without fundamental schema changes. In this sense, the relational WLCG schema must be reused. On the other hand, when hosts upload accounting data to WLCG-RUS system, standard and XML-based usage records are streamed as SOAP message payloads as defined in RUS specification. A consistent set of mapping rules should be applied to transform standard usage record instances into appropriate WLCG accounting data formats.

7. Internal Data

Accounting data uploaded from hosts must be represented in a compatible format to the WLCG schemas and persistent in relational database.

8. External Data

Since WLCG-RUS system is intended to provide OGF RUS compatible solution, usage records must be presented in OGF UR format when streaming into WLCG-RUS system.

Security and Privacy Requirements

Security is of highest importance in the design of the WLCG RUS system in order to ensure authenticated and authorised data sharing as well as preventing accounting data from being compromised during network transportation.

9. Authentication

The WLCG-RUS must provide X.509 certificate based authentication. Compared to traditional user/password authentication, the certificate-based authentication provides a higher level of security to prevent the system from unrecognised accesses.

10. Authorisation

The design of WLCG-RUS system should provide Role-Based Access Control (RBAC) and ensure fine-grained access control on operation and per usage record basis. The role-based access control should also apply to host management and user management facilities.

11. Data Integrity

Data integrity ensures that accounting data or usage records are not compromised during network transmission from remote hosts to WLCG-RUS server.

Other Requirements

Besides the above requirements, following requirements should also be put into consideration during the design of WLCG-RUS system.

12. Interoperability

The system must provide a standard compatible solution, in particular compatible to OGF RUS[138] and OGF UR[137] standards, as a consistent mechanism for data sharing other than introducing specific transportation mechanism to individual accounting system as current WLCG accounting process. A standard compatible solution also ensures interoperability to other standard compatible accounting systems in an implementation transparent manner.

13. Performance

It is witnessed that an increasing number of sites, over 200 until now, are participating in the WLCG project and share resource usage data to GOC. It is critically important for the WLCG-RUS system to ensure efficient performance and serve simultaneous requests within reasonable time.

14. Fault Tolerance

The WLCG-RUS system must be tolerant to runtime and service failures without breaking data consistency.

4.3 Design

This section discusses the design of WLCG-RUS system architecture and details of composite subsystem designs.

4.3.1 System Architecture

The WLCG-RUS architecture, as the deployment diagram illustrated in Figure 4.3, consists of two subsystems: the RUS service and WLCG-RUS Admin.

The RUS service implements two RUS interfaces: the “*RUS::insertUsageRecord*” and

“*RUS::listMandatoryUsageRecordElements*” interfaces, through which site-specific hosts query the mandatory element configurations and populate usage records. In order to automate data uploading process, it also provides a command-line client that can be scheduled to execute periodically. The communication protocols between RUS client and service is based on SOAP over HTTPS, which ensures data integrity and mutual authentication. On receiving a request, the RUS service endpoint delegates the request to a sequence of runtime components for fine-grained access control on per usage record basis, validation of received usage records against mandatory element configuration, rendering usage record instance to WLCG accounting data format, and saving accounting records into local relational database. In the case of active aggregate accounting, an additional aggregation strategy is triggered during the command execution. In order to enhance customisation and extensibility, the design of RUS services is based on a set of loose-coupled internal components, each of which exhibits well-defined internal interfaces.

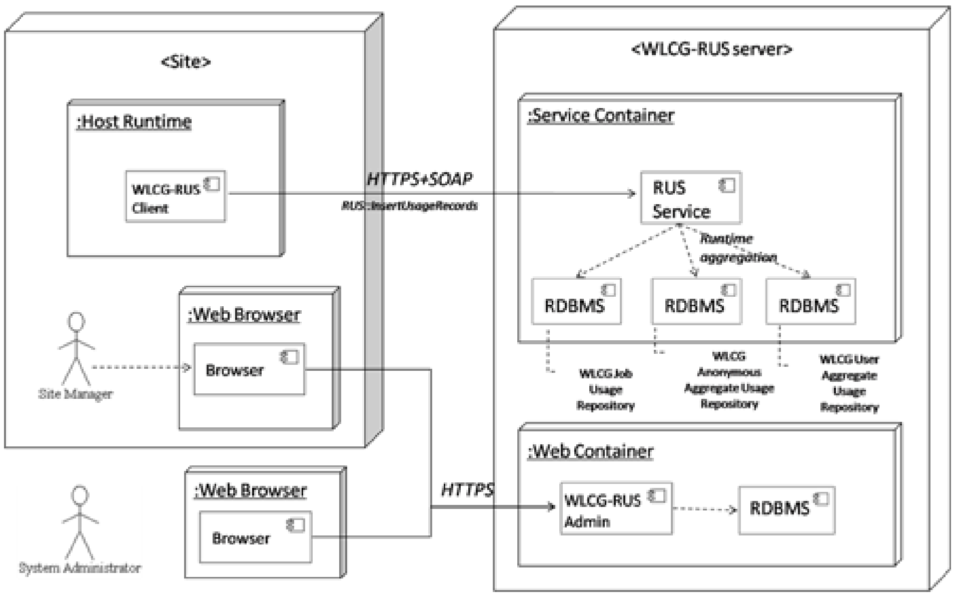


Figure 4.3: The Major Components of WLCG-RUS System and interactions

The WLCG-RUS Admin is a Web application that provides management facilities for both the system administrator and site managers. Specifically, the design of WLCG-RUS Admin is intended to provide three management facilities: user management, host management, and system administration. A user becomes a site manager candidate only when the user registers an account to the WLCG-RUS Admin system. The registration request is queued and to be activated by the system administrator. When activated, the site manager receives a confirmation email and can create one or more host accounts, which are required to be approved by system administrator before sharing usage records through RUS service. The WLCG-RUS Admin also keeps system configuration of RUS service, such as mandatory elements as well as custom implementation of internal components. System administrator can specify and change these system configurations at runtime without restarting the system. Because WLCG accounting data formats are reused within WLCG-RUS system, existing EGEE accounting portal can still be used as a Web-based graphic interface for resource usage reporting.

4.3.2 Detailed System Design

This section describes the design details of WLCG-RUS subsystems and individual composite components.

External Aggregate Data Representation

The design of WLCG-RUS system is intended to enable both active fine-grained and aggregate accounting models. With fine-grained accounting model, the OGF UR is used as the standard external accounting data representation. However, there was no standard aggregate data format available at the design time of the system. In 2006, we collaborated with researchers from Fermilab and CCLRC, and proposed a standard Aggregate Usage Record (AUR) schema, which had been submitted to OGF UR working group as a draft specification [158] for public review. This recommended aggregate usage record schema is adopted as an external data presentation for aggregate usage records.

An AUR instance represents resource usage of more than one Unit of Work (UoW) summarized according to a specific grouping criterion, also known as aggregation strategy. As shown in figure 4.4, the content model of AUR reuses most of usage metrics of URF and defines a category of aggregate properties. The common aggregate properties

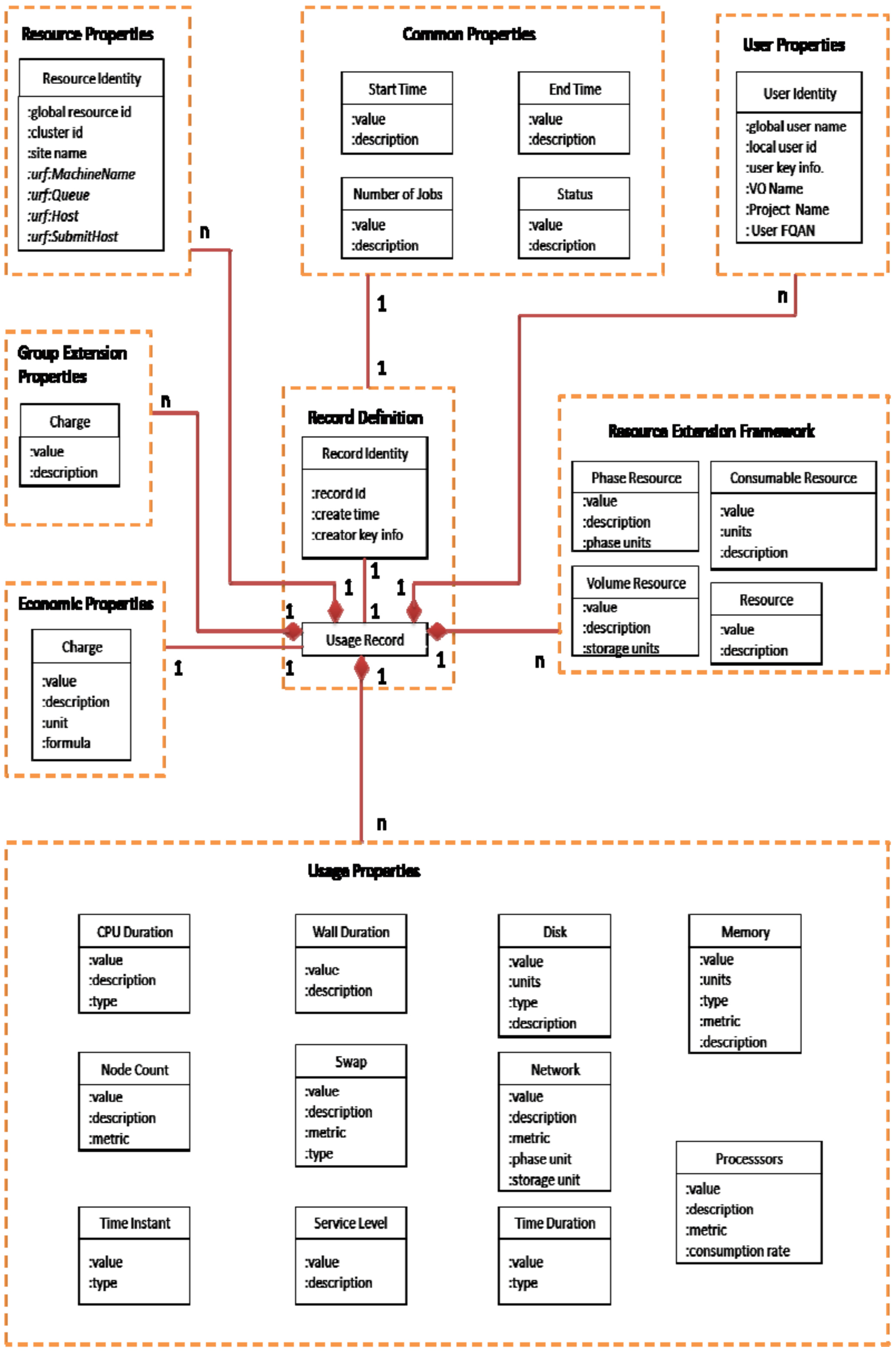


Figure 4.4: Proposed content model of aggregate usage record schema

are used to represent common metrics of a usage record instance, including total number of jobs aggregated, aggregation interval from the start time of earliest job to the end time of the lasted job, and overall status of jobs aggregated. These common properties are allowed to appear exactly once. User properties define ownership of aggregated jobs within a record instance. Besides user properties defined within OGF UR, AUR introduces additional user-related properties, Virtual Organization (VO) and Full Qualified Attribute Name (FQAN). Resource-related properties are encapsulated within a resource identity element, and are divided into local and global resource properties. Local resource properties include site-specific machine name, queue name and execution host name, which are defined within OGF UR , while global resource properties defines Grid-wide properties such as global resource identity, cluster identity and participating site name. User and resource aggregate properties can appear more than once within a record instance, depicting certain aggregation strategy. The WLCG anonymous summary record, for example, defines aggregation strategy that summarizes resource usage of jobs on per VO, per site, per month and per year basis. For an aggregation strategy requires custom properties not defined with aggregate properties of AUR, the grouping extension property can be used.

Design of Resource Usage Service

In order to enhance customisation and extensibility, the design of RUS service is based on component architecture, consisting of a set of loose-coupled and reusable components. Each component realises certain functionality and exhibits well-defined interfaces. These components have been designed to be loosely coupled, so that they can be easily customised, upgraded and replaced to adapt to local deployment requirements.

As the class diagram illustrated in Figure 4.5, there are four major functional components defined within RUS service. The “Command” component is the main functional component that encapsulates all required information associated with execution of RUS logic operations. A single common interface, “*execute()*”, decouples completely between RUS service endpoint and various “Command” component implementations. With a single interface, a RUS service can delegate incoming requests to different “Command” implementations in a consistent manner. A RUS service may chose to implement a single “Command” implementation that serves all RUS requests or to have multiple “Command” implementations that serve individual

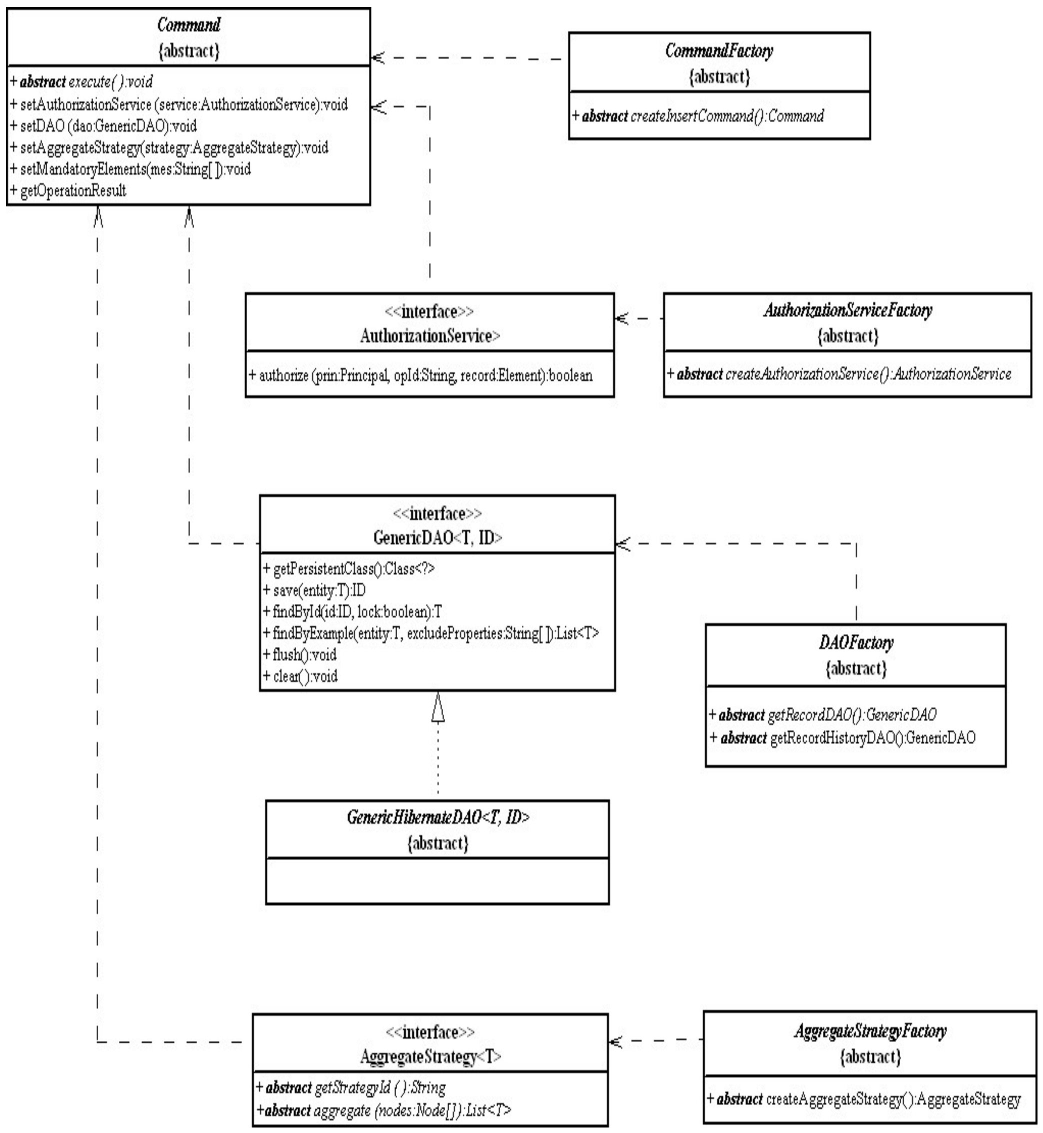

Command
{abstract}
+ abstract execute( ):void
+ setAuthorizationService (service:AuthorizationService):void
+ setDAO (dao:GenericDAO):void
+ setAggregateStrategy(strategy:AggregateStrategy):void
+ setMandatoryElements(mes:String[ ]):void
+ getOperationResult
CommandFactory
{abstract}
+ abstract createInsertCommand():Command
<<interface>>
AuthorizationService>
+ authorize (prin:Principal, opId:String, record:Element):boolean
AuthorizationServiceFactory
{abstract}
+ abstract createAuthorizationService():AuthorizationService
<<interface>>
GenericDAO<T, ID>
+ getPersistentClass():Class<?>
+ save(entity:T):ID
+ findById(id:ID, lock:boolean):T
+ findByExample(entity:T, excludeProperties:String[ ]):List<T>
+ flush():void
+ clear():void
DAOFactory
{abstract}
+ abstract getRecordDAO():GenericDAO
+ abstract getRecordHistoryDAO():GenericDAO
GenericHibernateDAO<T, ID>
{abstract}
<<interface>>
AggregateStrategy<T>
+ abstract getStrategyId ( ):String
+abstract aggregate (nodes:Node[ ]):List<T>
AggregateStrategyFactory
{abstract}
+ abstract createAggregateStrategy():AggregateStrategy

Figure 4.5: Class diagram of RUS service runtime components

RUS service interfaces. The execution of various "Command" component shares common requirements, including checking user permissions, data persistence, and runtime aggregation. These common requirements can be realised through other three components defined within RUS service. The authorisation service component provides an interface for fine-grained access control over per operation and per usage record. Different authorisation mechanisms can be applied by implementing authorisation service. Data Access Object (DAO) component provides a higher-level abstraction upon underlying data persistent storage, and can be potentially implemented for XML:DB[161], relational database, file systems and other storage format. Considering that various relational databases might be used for usage record persistence, a separate DAO component, the Hibernate [162] DAO, is also implemented by extending the generic DAO component and places another abstraction upon heterogeneous relational database engines. For runtime aggregation, different aggregation algorithms can be implemented by extending the aggregate strategy interface. Each component of RUS service has an associated factory class that creates and instantiates appropriate component instances dynamically. These component and factory implementations are snapped together to provide a complete solution of RUS service.

Design of Administrative Web Application

The design of the WLCG-RUS admin Web application is based on Model-View-Controller (MVC) pattern, with models encapsulating domain-specific representation of data, controllers representing domain-specific logics operating upon to data, and views providing Web-based interfaces allowing end-user interactions. As Figure 4.6, the WLCG-RUS admin Web application is intended to provide administrative and host management facilities for the WLCG-RUS system administrator and site managers.

In order to access the WLCG-RUS admin system, a user must have a valid and recognised X.509 user certificate, and has a valid user account. Each user is directed to specific view according to granted role. Site manager only have access to host management facilities, which allows host registration, exploring host status, and deleting a host. Newly registered host cannot share accounting data or usage records through RUS service endpoint until its registration request is approved by the system administrator. A site manager only has management authority of owned hosts. System administrator has an administrative view, which provides user management facilities and host management facilities. A system administrator can create a new role, grant a role to registered users, revoke a user, publish system announcements, and have full control over all hosts registered by site managers. Another important usage of WLCG-RUS admin system is to specify RUS service configurations, including factory of RUS service functional components and mandatory elements for validating incoming usage records.

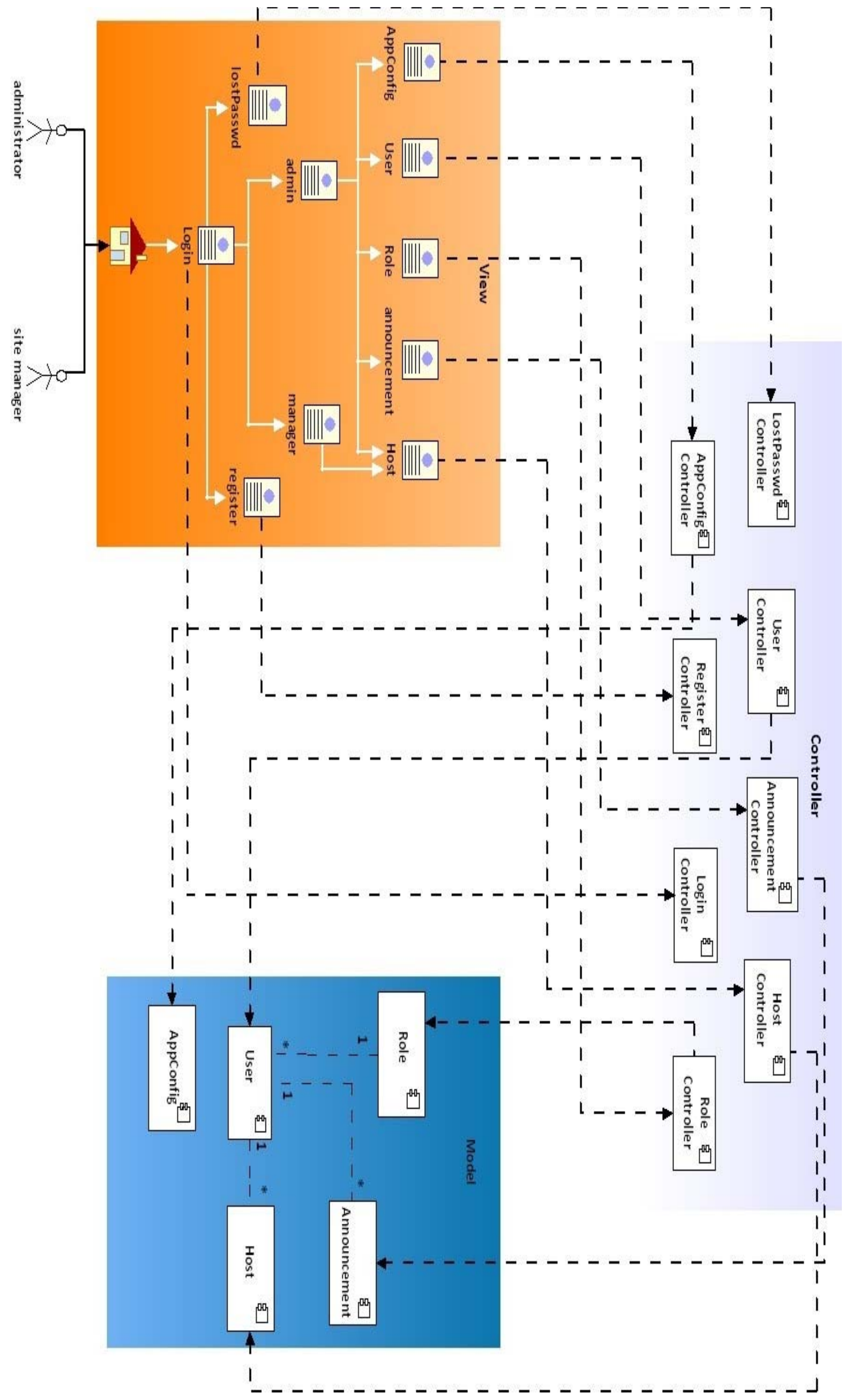


Figure 4.6: WLCG-RUS Admin MVC Model

4.4 Implementation

The implementation of WLCG-RUS system is based on Service-Oriented Architecture (SOA) in the profile of Web Service Interoperability (WS-I) version 1.2[163]. The development of RUS service makes use of Apache Axis version 1.4[164] as SOAP engine, which has proved to be a stable and reliable system, and is widely used for commercial application servers. Java was chosen as the language for the system because it is platform independent and has well-defined design patterns. The development of WLCG-RUS Admin is based on Grails[165], an open source Web application framework, and ideal for developing MVC Web applications. The Grails leverages the Groovy[166] programming language, which is based on Java platform as well, making it easy for communication between RUS service and WLCG-RUS Admin. By using Grails, the WLCG-RUS Admin and RUS service are packaged as a single software release. This section discusses implementation details of WLCG-RUS system and its subsystems.

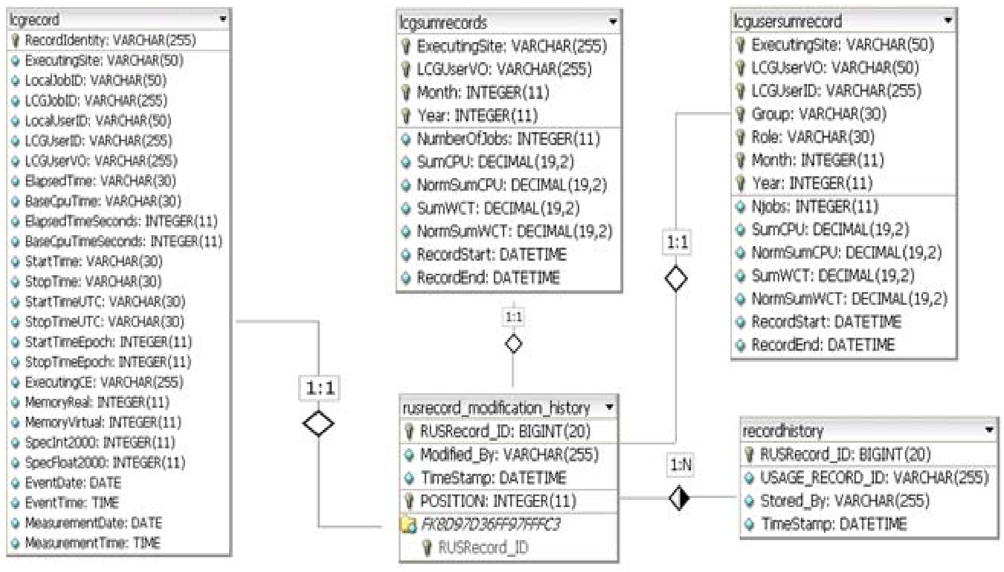


Figure 4.7: Internal data model of RUS service reuses existing WLCG accounting schema with additional record history model

4.4.1 Resource Usage Service

External-Internal Data Mapping

In order to ensure ease of uploading accounting data through standard RUS interface, a data mapping mechanism is required to enable dynamic transformation from external data represented in standard OGF usage record and aggregate usage record formats into corresponding WLCG accounting formats (Figure 4.7). The mapping rules of OGF UR and WLCG schema have been discussed in section 3.5.2 and given in Appendix A-2. Similar mapping rules are also introduced for mapping between proposed standard AUR properties and WLCG summary schema as Appendix A-6.

Apart from mapping rules between standard usage and WLCG usage metrics, another important issue to be solved is the data consistency, when uploading accounting data to RUS service endpoint, in particular for time synchronisation and storage units, which are summarised as follows:

- Considering the fact that WLCG accounting usage records might come from sites of countries in different time zones, the default implementation of RUS service requires every timestamp-related usage properties to be expressed in ISO8601[167] format (e.g. 2008-10-01T20:39:28Z or 2008-10-01T21:39:28+01:00), and transforms timestamp values to Coordinated Universal Time (UTC) values therefore ensuring time consistency.
- For volume resource usage properties, such as memory and disk usage, the default storage unit is KB, unless it is explicitly specified as the RUS service configuration property, "*storage.unit*".
- During fine-grained aggregate accounting process, individual usage records are to be summarised before being stored a into local database. WLCG aggregate strategies involve a normalisation process that normalise the CPU usage information across disparate sites into a common reference scale based on SpecInt2000 benchmark.

The mapping and data consistency rules between standard usage record instances are implemented and ensured by three entity classes. Each entity class has two constructors, the default constructor instantiating an empty entity instance, and the constructor that takes a usage record instance as a parameter and instantiates an entity instance by applying mapping rules and data consistency constraints.

Job Accounting Model

The job accounting model is implemented within WLCG-RUS system by extending the internal components of RUS service. As Figure 4.8, the "*LcgRecord*" class is the object model that represents WLCG relational job accounting data model. The "*LcgRecordDAO*" component extends the internal "*GenericDAO*" interface with typed parameters referring to "*LcgRecord*" object model and its identity data type. The WLCG-RUS job accounting model uses Hibernate Object-Relation Mapping (ORM) engine for mapping and saving Java objects to MySQL relational database.

As shown in Figure 4.9, in the processing of job accounting information in the WLCG-RUS system involves the following steps and a sequence of interactions between internal components of RUS service:

1) A client sends a "*RUS::insertUsageRecords*" SOAP request message to RUS service endpoint.
2) On receiving insertion request, the RUS service endpoint loads command factory, DAO factory, authorisation service factory, and mandatory elements from RUS service configuration. The RUS service endpoint then instantiates an "*LcgRecordInsertCommand*" instance and set DAO instance, authorisation service instance and mandatory elements to the command instance.
3) The RUS service endpoint delegates insertion request to the command through the "*execute()*" interface.
4) For each usage record, the execution of command firstly checks user authority to perform insertion operation upon the usage record.
5) Once authorised, the command then validates the current usage record against mandatory element configuration.
6) If validated, the command creates an *LcgRecord* instance by passing the current usage record to *LcgRecord* constructor.
7) The command then invokes the save method of *LcgRecordHibernateDAO* and passes the instantiated *LcgRecord* instance.
8) The DAO object makes the *LcgRecord* instance persistent into local relational database and returns a record identity.

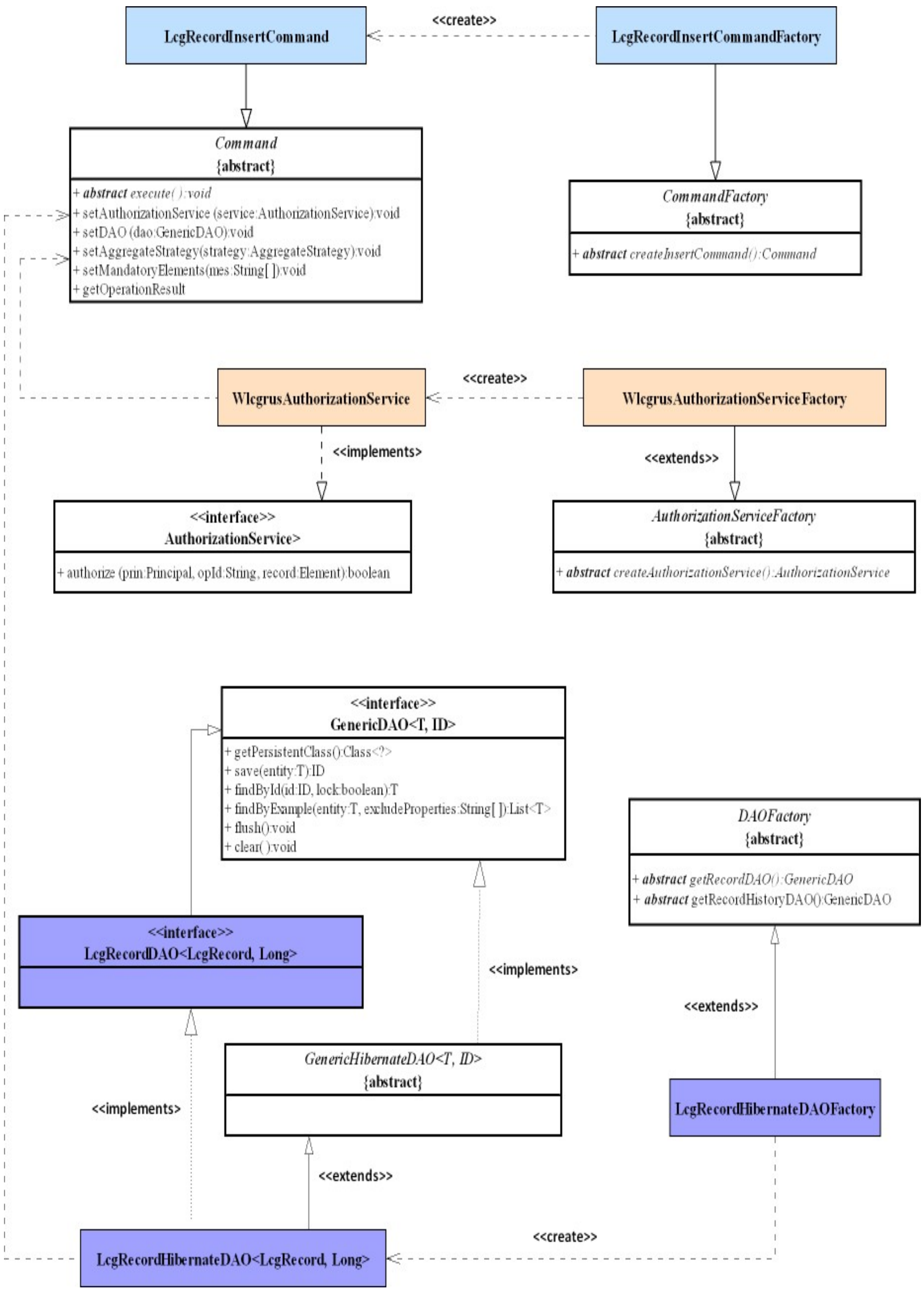

LcgRecordInsertCommand
<<create>>
LcgRecordInsertCommandFactory
Command
{abstract}
+ abstract execute( ):void
+ setAuthorizationService (service:AuthorizationService):void
+ setDAO (dao:GenericDAO):void
+ setAggregateStrategy(strategy:AggregateStrategy):void
+ setMandatoryElements(mes:String[ ]):void
+ getOperationResult
CommandFactory
{abstract}
+ abstract createInsertCommand( ):Command
WlcgrusAuthorizationService
<<create>>
WlcgrusAuthorizationServiceFactory
<<implements>
<<extends>>
<<interface>>
AuthorizationService>
+ authorize (prin:Principal, opId:String, record:Element):boolean
AuthorizationServiceFactory
{abstract}
+ abstract createAuthorizationService( ):AuthorizationService
<<interface>>
GenericDAO<T, ID>
+ getPersistentClass():Class<?>
+ save(entity:T):ID
+ findById(id:ID, lock:boolean):T
+ findByExample(entity:T, excludeProperties:String[ ]):List<T>
+ flush():void
+ clear( ):void
DAOFactory
{abstract}
+ abstract getRecordDAO( ):GenericDAO
+ abstract getRecordHistoryDAO():GenericDAO
<<interface>>
LcgRecordDAO<LcgRecord, Long>
<<implements>
<<extends>>
GenericHibernateDAO<T, ID>
{abstract}
<<implements>
LcgRecordHibernateDAOFactory
<<extends>>
<<create>>
LcgRecordHibernateDAO<LcgRecord, Long>

Figure 4.8: WLCG-RUS job accounting model implementation (UML Class diagram)

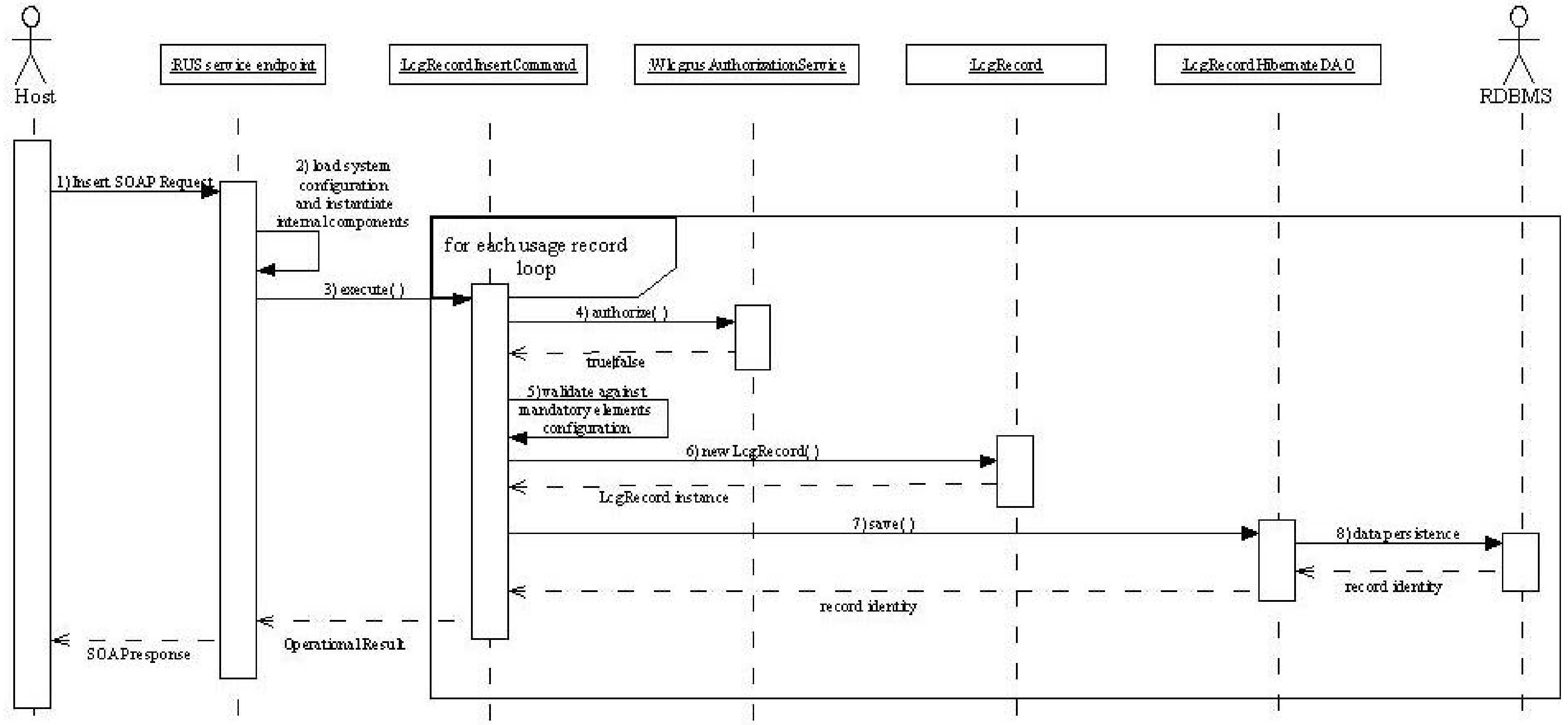


Figure 4.9:WLCG-RUS Job Accounting Model (UML Sequence Diagram)

Aggregate Accounting Model

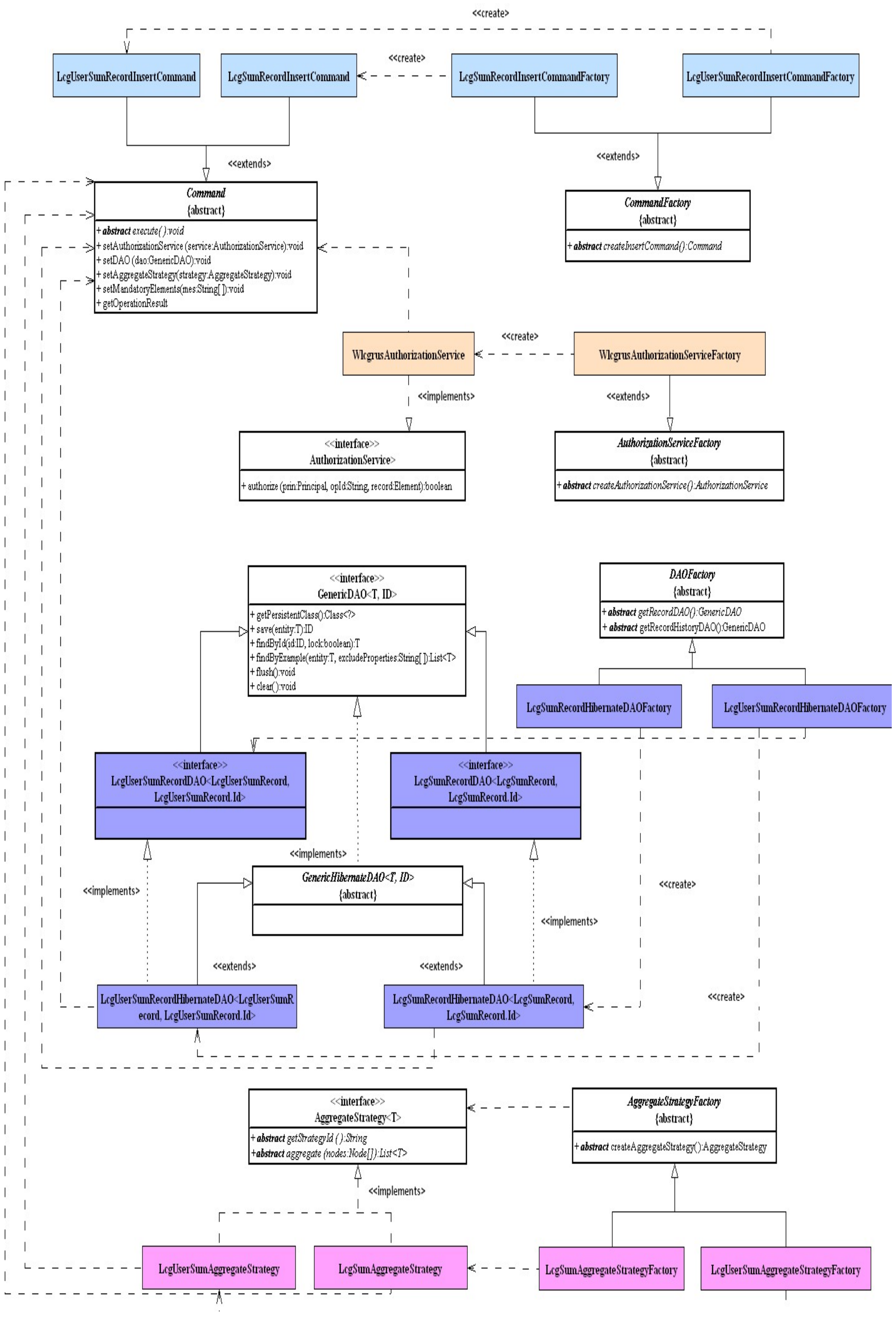
Aggregate accounting model implemented in WLCG-RUS system accepts pre-aggregated usage records in OGF AUR format as well as job usage records in OGF UR format, which are to be aggregated during execution of insertion. In the latter case, an aggregate strategy should be applied to generate appropriate AUR instances. As Figure 4.10, there are two aggregate strategy classes implementing WLCG anonymous and user aggregate strategies respectively. These aggregate strategies are to be triggered by corresponding command implementations, and generate instances of either WLCG anonymous aggregate records or WLCG user aggregate records, which are to be stored into relational databases through DAO implementations.

Aggregate accounting processing models implemented within WLCG-RUS is given in Figure 4.11, and involving following processing steps:

1) A client sends a "*RUS::insertUsageRecords*" SOAP request message to an RUS service endpoint.
2) On receiving insertion request, the RUS service endpoint instantiates command, authorisation service, DAO and aggregate strategy instances through configured factory classes, and loads mandatory element configurations into runtime.
3) The RUS service endpoint delegate insertion request to the command through *execute( )* interface.
4) For each usage record instance, the execution of command firstly checks user authority to perform insertion.
5) Once authorised, the command then validates the current usage record against mandatory element configurations.
6) If received usage records are OGF UR instances, an aggregate strategy is triggered and generates one or more instances of target aggregate object model, instances of WLCG anonymous aggregate records in this example.

6.1) Otherwise, the command creates an instance of target aggregate object model by passing the current OGF AUR instance to "*LcgSumRecord*" constructor.

7) The insert command then invokes the save method of "*LcgSumRecordDAO*" and passes the "*LcgSumRecord*" instance.
8) The DAO object makes the "*LcgSumRecord*" instance persistst into a local relational database and returns a record identity.



<<create>
LcgUserSumRecordInsertCommand
LcgSumRecordInsertCommand
<<create>
LcgSumRecordInsertCommandFactory
LcgUserSumRecordInsertCommandFactory
<<extends>
<<extends>
Command
{abstract}
+ abstract execute( ):void
+ setAuthorizationService (service:AuthorizationService):void
+ setDAO (dao:GenericDAO):void
+ setAggregateStrategy(strategy:AggregateStrategy):void
+ setMandatoryElements(mes:String[ ]):void
+ getOperationResult
CommandFactory
{abstract}
+ abstract createInsertCommand():Command
WlcgrusAuthorizationService
<<create>
WlcgrusAuthorizationServiceFactory
<<implements>
<<extends>
<<interface>>
AuthorizationService>
+ authorize (prin:Principal, opId:String, record:Element):boolean
AuthorizationServiceFactory
{abstract}
+ abstract createAuthorizationService():AuthorizationService
<<interface>>
GenericDAO<T, ID>
+ getPersistentClass():Class<?>
+ save(entity:T):ID
+ findById(id:ID, lock:boolean):T
+ findByExample(entity:T, excludeProperties:String[ ]):List<T>
+ flush():void
+ clear():void
DAOFactory
{abstract}
+ abstract getRecordDAO():GenericDAO
+ abstract getRecordHistoryDAO():GenericDAO
LcgSumRecordHibernateDAOFactory
LcgUserSumRecordHibernateDAOFactory
<<interface>>
LcgUserSumRecordDAO<LcgUserSumRecord, LcgUserSumRecord.Id>
<<interface>>
LcgSumRecordDAO<LcgSumRecord, LcgSumRecord.Id>
<<implements>
GenericHibernateDAO<T, ID>
{abstract}
<<implements>
<<create>
<<implements>
<<extends>
<<extends>
<<create>
LcgUserSumRecordHibernateDAO<LcgUserSumRecord, LcgUserSumRecord.Id>
LcgSumRecordHibernateDAO<LcgSumRecord, LcgSumRecord.Id>
<<interface>>
AggregateStrategy<T>
+ abstract getStrategyId ( ):String
+ abstract aggregate (nodes:Node[]):List<T>
AggregateStrategyFactory
{abstract}
+ abstract createAggregateStrategy():AggregateStrategy
<<implements>
LcgUserSumAggregateStrategy
LcgSumAggregateStrategy
LcgSumAggregateStrategyFactory
LcgUserSumAggregateStrategyFactory

Figure 4.10: Class diagram of RUS default implementation for aggregate accounting

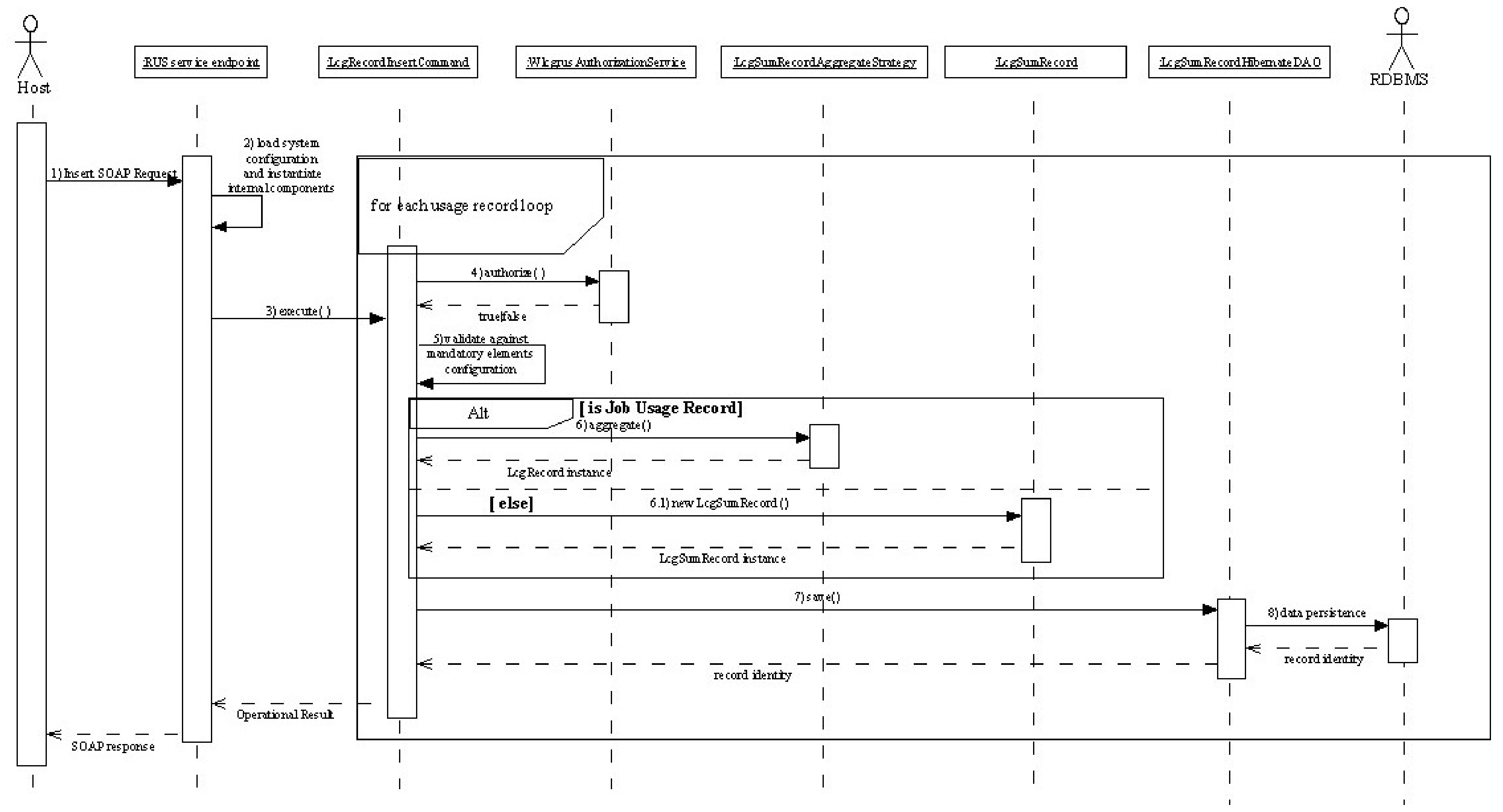


Figure 4. 11: WLCG-RUS Aggregate Accounting Process Model (UML Sequence Diagram)

4.4.2 WLCG-RUS Admin

WLCG-RUS Admin Web application is implemented based on Grails framework and use the Groovy script language. The implementation adopts the passive MVC model with one controller exclusively manipulating one model and refreshing changes of model to views.

Host Management

As illustrated in Figure 4.12, the implementation of host management consists of a host controller, a host model class, and a set of view pages. The host controller class defines a list of methods that serve HTTP GET and POST requests. On receiving HTTP request, the hosting server of WLCG-RUS system invokes an appropriate method defined in the host controller, which then evaluates conditions using host model class if required. The host controller also decides which view should be built with the required data obtained from model class and renders the view to HTML for display.

An instance of host model class encapsulates meta-information of a host as a registry entry, including the host name, host certificate distinguished name as displayed in host certificate, the site name it belongs to site manager, registration date, and status. There are four "do-GET" methods defined within the host controller, "list", "create", "edit", and "show". Each "do-GET" method has an associated view to display for user interaction. The "list" method is used to display all host registry entries. The "list" method is triggered to display host registry entries owned by the site manager, while displaying all host registry entries maintained inside the WLCG-RUS system to the system administrator. The "list" view also connects to the "show" view and the "edit" view, for displaying detailed registration information of individual host and updating registration details except the host's distinguished name. A site manager may create a new host entry by filling the form as displayed by "create" view. On submission of the form, the "do-POST" method "save" is triggered to create a new instance of host model and make it persistence into local relational database. Other "do-POST" methods defined within host controller include "delete", "update", "enable" and "disable". A site manager may update host registration information except its enabling status. Every newly registered host is disabled by default. Only the system administrator has the authority to approve or disapprove a host through the "enable" or "disable" methods defined in the host controller.

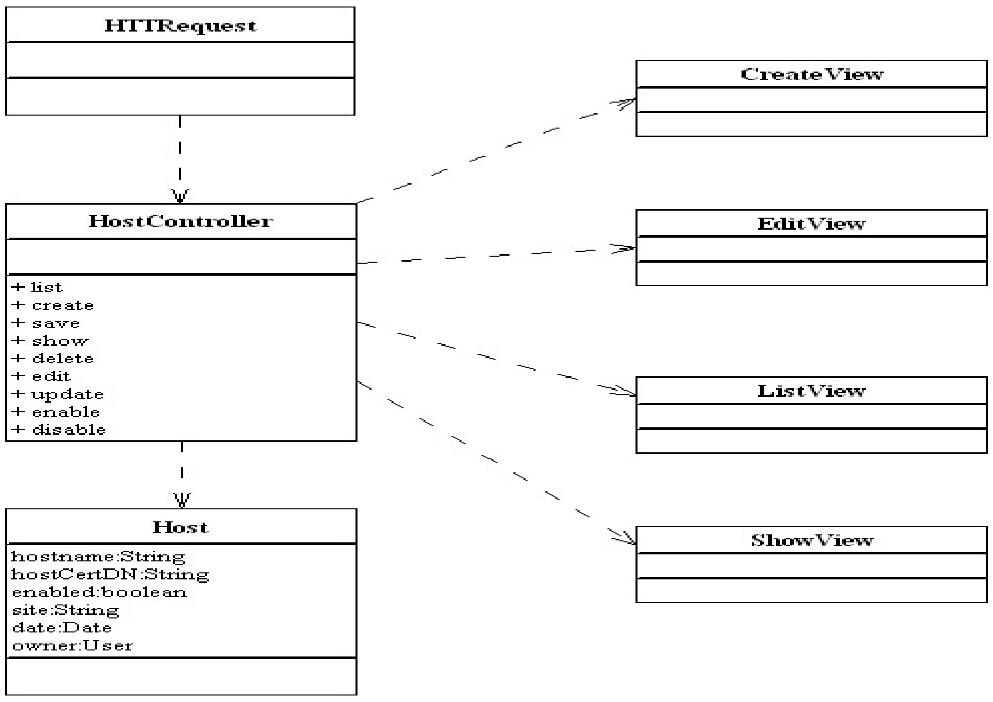


Figure 4. 12: Class Diagram of the Host Management Implementation

User Management

The user management facilities implemented within WLCG-RUS Admin is based on the default security plug-in for the Spring framework [170], called Acegi[171], which manages most of the complexity of role-based authorisation, user login, and request-URL mapping issues. As shown in Figure 4.13, the Acegi plug-in generates two main model classes that can be used for user management tasks of WLCG-RUS system. Each model class has an exclusive controller class dealing with HTTP requests for creation, deletion, modification, and listing of user accounts, and role definitions.

However, the default Acegi security implementation only provides simple user-password authentication. The implementation of WLCG-RUS Admin adds a certificate-based authentication. In this sense, a user can access WLCG-RUS Admin only if the user holds a valid X.509 certificate signed by a recognised CA. When entering into the main page, the user is required to be registered and assigned to a role. There are two predefined roles in WLCG-RUS Admin system, the site manager and system administrator. User

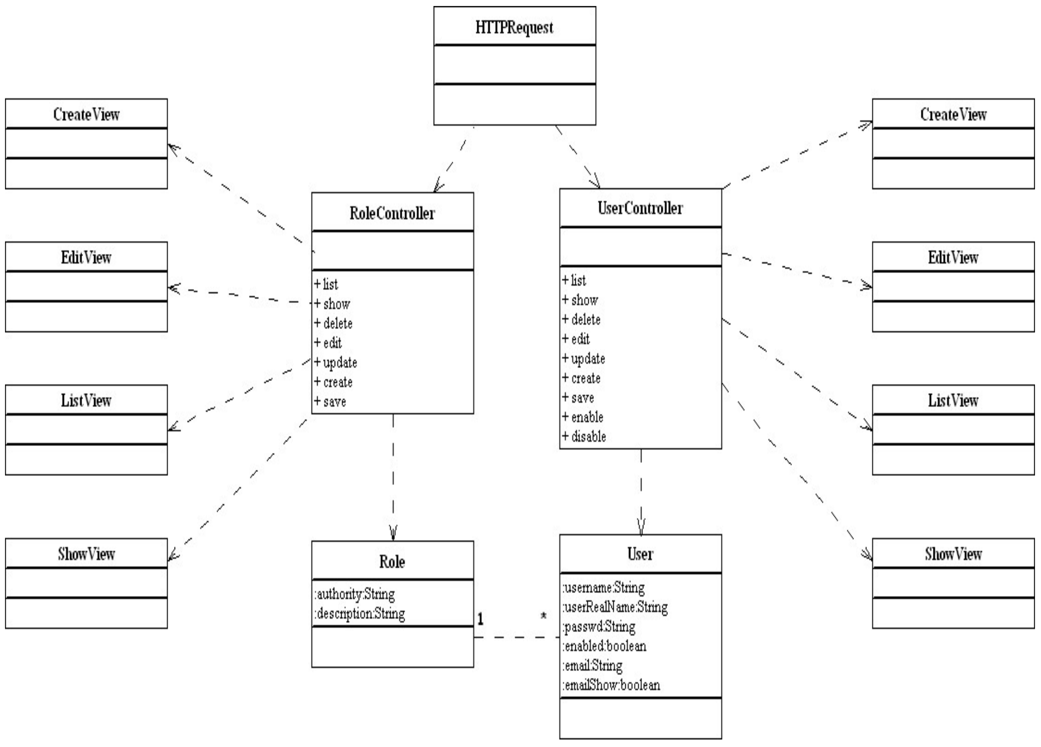


Figure 4.13: Class Diagram of the Host Management Implementation

registration requests are pending for system administrator to approve. The system administrator can approve or disapprove a user through the “enable” and “disable” interfaces defined within user controller. Enabled users should receive an email notification when their accounts are approved by system administrator.

System Management

As discussed in section 4.4.1, the RUS service runtime involves interactions between a sequence of implementations of internal components. These components are required to be configured properly and be instantiated appropriately to perform job and aggregate accounting processes. The system management facilities provided by WLCG-RUS Admin system allows system administrator to specify, or modify, or delete RUS service runtime configurations.

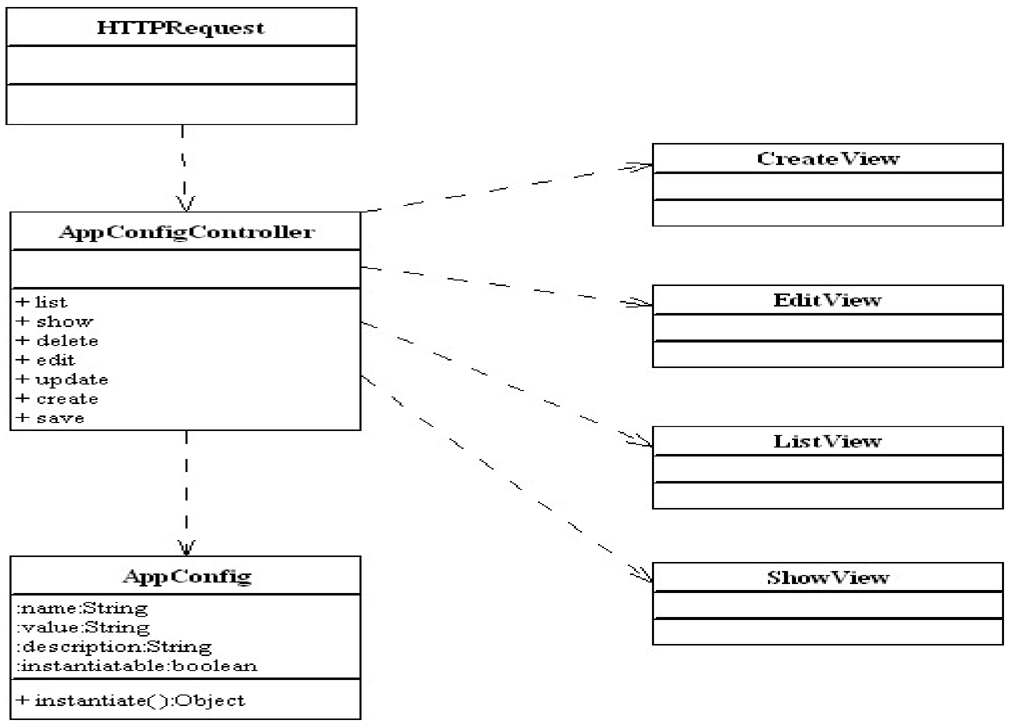


Figure 4.14: Class Diagram of the System Management Implementation

As Figure 4.14, the "*AppConfig*" model class represents a single name-value configuration property. RUS service configurations can be divided into two categories, instantiatable properties, such as factories of runtime components, and uninstantiatable properties, the mandatory usage record elements for example. Instantiatable properties use the "instantiate ()" method defined with the model class. Same as other model classes, the "*AppConfig*" model has an exclusive controller class, which serves incoming HTTP requests and directs users to difference views.

4.4.3 User Interface

WLCG-RUS system provides two user interfaces for both RUS service and WLCG-RUS Admin Web application.

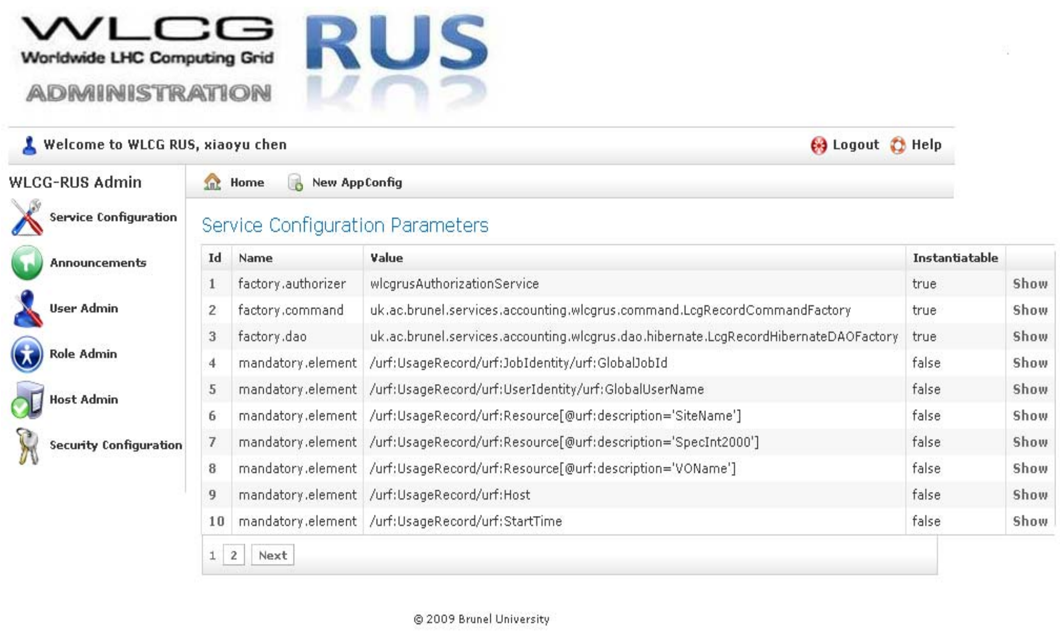
Command Line Interface

The RUS service provides client-side interfaces and implementations for interaction with WLCG-RUS system through standard RUS interfaces, in particular "*RUS::ListMandatoryUsageRecordElements*" and "*RUS::InsertUsageRecords*". The RUS service client is implemented using Java programming language, and is wrapped by a shell script. The client accepts arguments as shown in Appendix B. The client defines two actions, the "list" and "insert", both must specify a "service URI" parameter setting the value of target RUS service endpoint address. For the "insert" action, either a single file location, or a directory, or a list of files is required to be specified for the actual usage record files or directories. The usage of an optional parameter, "delete-after-insertion" tells the client whether to delete the usage record files after successful insertion. Another optional parameter is the "max-elements" that is used to specify the maximum number of usage records per insertion. If this parameter is omitted, the default maximum number is set to 10. If any errors are encountered during execution of insertion, the target file name is changed and appended with an "ERROR" suffix. The RUS service client is to be used by host machines to upload usage records to WLCG-RUS system. The shell client can run as a "cron" job to be scheduled to populate usage records periodically.

Web Interface

WLCG-RUS Admin also provides a Web interface for site manger and system administrator to perform management tasks. This Web-based interface exhibits two views, the manager view to site manager, and admin view for system administrator. Once a user logs in successfully, the WLCG-RUS Admin system redirects the user to a different view according to user's granted role.

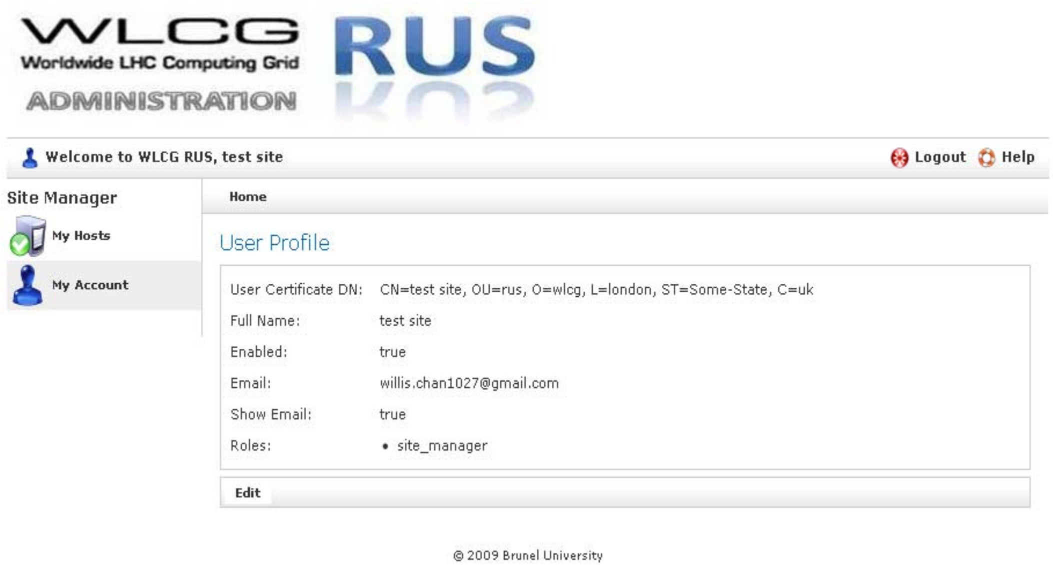
As Figure 4.15, the admin view provides navigation to user management and system management facilities. The screen shot also gives the list of RUS service configurations, including the authorisation factory class name, command factory class name, DAO factory name, as well as other configurations such as mandatory usage record elements. These mandatory element configurations are represented as XPath [170] expressions, which are to be evaluated against received usage records. As Figure 4.16, the manager view provides the link to host management. The manager view also allows a site manager to view and edit personal profile. However, a site manager is not allowed to modify account status, and user certificate distinguished name that is parsed by the system and not modifiable.



Service Configuration Parameters

Id	Name	Value	Instantiatable	
1	factory.authorizer	wlcgrusAuthorizationService	true	Show
2	factory.command	uk.ac.brunel.services.accounting.wlcgrus.command.LcgRecordCommandFactory	true	Show
3	factory.dao	uk.ac.brunel.services.accounting.wlcgrus.dao.hibernate.LcgRecordHibernateDAOFactory	true	Show
4	mandatory.element	/urf:UsageRecord/urf:JobIdentity/urf:GlobalJobId	false	Show
5	mandatory.element	/urf:UsageRecord/urf:UserIdentity/urf:GlobalUserName	false	Show
6	mandatory.element	/urf:UsageRecord/urf:Resource[@urf:description='SiteName']	false	Show
7	mandatory.element	/urf:UsageRecord/urf:Resource[@urf:description='SpecInt2000']	false	Show
8	mandatory.element	/urf:UsageRecord/urf:Resource[@urf:description='VOName']	false	Show
9	mandatory.element	/urf:UsageRecord/urf:Host	false	Show
10	mandatory.element	/urf:UsageRecord/urf:StartTime	false	Show

Figure 4.15: WLCG-RUS Admin View



User Profile

User Certificate DN:	CN=test site, OU=rus, O=wlcg, L=london, ST=Some-State, C=uk
Full Name:	test site
Enabled:	true
Email:	willis.chan1027@gmail.com
Show Email:	true
Roles:	• site_manager

Figure 4.16: WLCG-RUS Site Manager View

4.5 Performance

This section provides details on performance evaluation of the WLCG-RUS system. The test results are intended to provide reference guidance for deployment of WLCG-RUS system with optimal performance.

4.5.1 Testbed

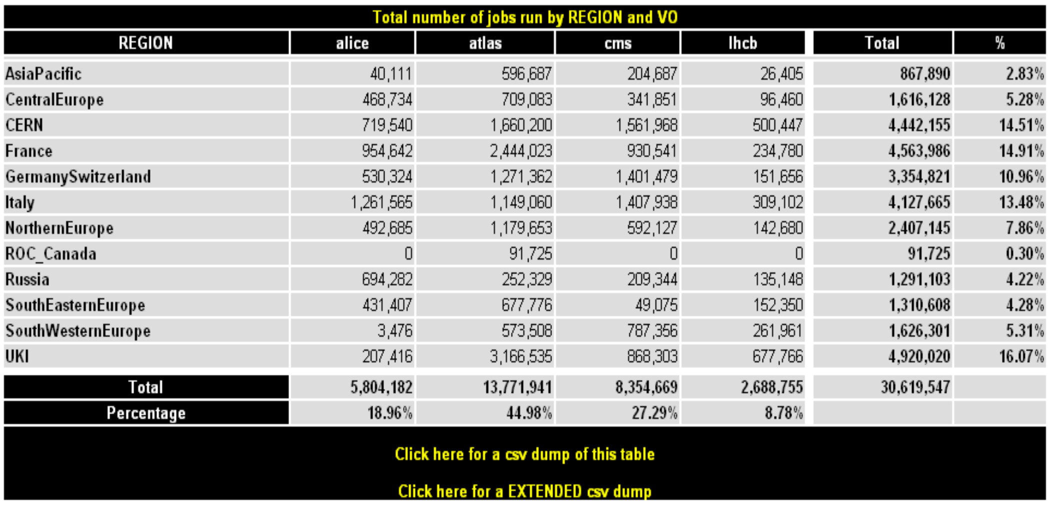
In order to evaluate the performance of WLCG-RUS system, a testbed is set up in the Brunel Information Technology Laboratory (BITLab) at Brunel University to simulate the accounting process in production Grid environments. The testbed consists of two workstations in BITLab and are interconnected by Local Area Network (LAN). One dedicated workstation is used to host WLCG-RUS server, which keeps listening insertion requests from clients. The hardware and runtime environment details of the WLCG-RUS server are listed in Table 4-1. On the other workstation, a number of clients along with a usage record generator are deployed to simulate the accounting process at Grid participating sites. The usage record generator simulates the metering process and generates standard OGF UR and AUR instances into the local file system. One or more WLCG-RUS clients can be scheduled to read usage record instances from the directory and populate them to the WLCG-RUS server simultaneously through the standard RUS::InsertUsageRecords interface. A thread pool is also provided to hold multiple WLCG-RUS client threads and ensure a fixed number of threads that interrogate the WLCG-RUS server at a time.

Table 4-1: Test server machine specification and runtime environment

	Description	**Version**
Central Processing Unit	Genuine Intel (R) Duo Core 1.66 GHz	-
Random Access Memory	1024 MB	-
Operating Ssytem	Ubuntu 32-bit	9.04
Web Container	Apache Tomcat	5.5.23
Service Container	Apache Axis	1.4
DBMS	MySQL Community Server	5.1

In the accounting scenario of the largest world-wide Grid environment, the WLCG environment, there are over 100 participating or Tier-2 sites that reports usage information to 12 regional or Tier-1 sites. These collected usage records at regional sites are then shipped to the GOC site at RAL and generate statistic usage reports. As illustrated in Table 4-2, there were over 30,000,000 jobs submitted to WLCG across four Virtual Organisations (VOs) in 2007, approximately 80,000 jobs executed at Tier-2 sites.

Table 4-2: WLCG job statistics from four VOs and 12 Tier-1 or regional sites. From[159]



Total number of jobs run by REGION and VO

REGION	alice	atlas	cms	lhcb	Total	%
AsiaPacific	40,111	596,687	204,687	26,405	867,890	2.83%
CentralEurope	468,734	709,083	341,851	96,460	1,616,128	5.28%
CERN	719,540	1,660,200	1,561,968	500,447	4,442,155	14.51%
France	954,642	2,444,023	930,541	234,780	4,563,986	14.91%
GermanySwitzerland	530,324	1,271,362	1,401,479	151,656	3,354,821	10.96%
Italy	1,261,565	1,149,060	1,407,938	309,102	4,127,665	13.48%
NorthernEurope	492,685	1,179,653	592,127	142,680	2,407,145	7.86%
ROC_Canada	0	91,725	0	0	91,725	0.30%
Russia	694,282	252,329	209,344	135,148	1,291,103	4.22%
SouthEasternEurope	431,407	677,776	49,075	152,350	1,310,608	4.28%
SouthWesternEurope	3,476	573,508	787,356	261,961	1,626,301	5.31%
UKI	207,416	3,166,535	868,303	677,766	4,920,020	16.07%
Total	5,804,182	13,771,941	8,354,669	2,688,755	30,619,547	
Percentage	18.96%	44.98%	27.29%	8.78%		

Click here for a csv dump of this table

Click here for a EXTENDED csv dump

Therefore the testbed is designed to evaluate the WLCG-RUS system performance by simulating the hierarchical deployment of WLCG-RUS server at both WLCG regional sites and the GOC site.

- The deployment of WLCG-RUS system at each regional site to collect job usage from region-wide Tier-2 sites.
- The deployment of WLCG-RUS system at WLCG GOC site to collect job usage from 12 regional sites.

Accordingly the evaluation objectives include:

- Unit performance evaluation: to evaluate the performance of individual WLCG-RUS runtime components (section 4.3.2), the result of which is to be used by deployers to have a detailed picture on how WLCG-RUS system perform, and by developer to improve system performance through custom implementation of particular runtime components.

- Insertion performance evaluation: to evaluate how the WLCG-RUS system performance varies with different deployment options, in particular the number of usage records per insertion transaction, known as bulk size, and the number of client threads. The result of the insertion performance test is expected to be used by deployers to make decisions on how to deploy WLCG-RUS system to obtain optimal performance.

4.5.2 Unit Performance

Figure 4.17 plots the performance of runtime component units of different accounting models, both job accounting and aggregation accounting models. Multithreading is intentionally avoided in these tests so that overall time of a series of runtime steps of various enabled accounting models within WLCG-RUS system can be fairly observed and compared.

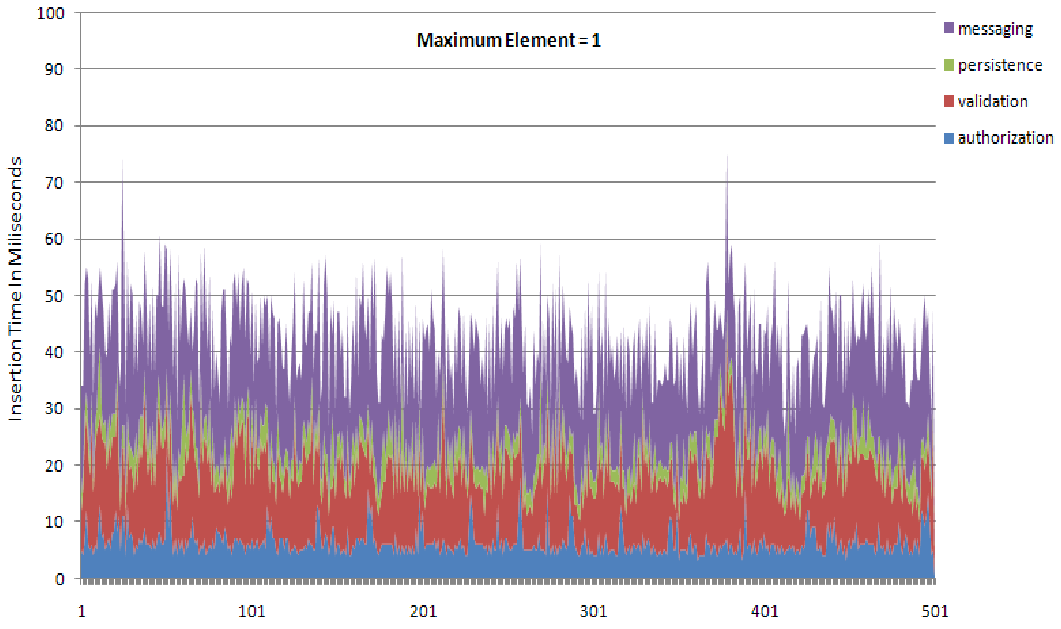
There are four common processing steps for both fine-grained job and aggregate accounting models as follows:

- On receiving a usage record, the Axis SOAP engine de-serialises received SOAP request message, and forward a request object to command component.
- On completion of insertion, the command returns a response object back to Axis SOAP engine, which then serialises the response object and sends response message to WLCG-RUS client. The de-serialisation and serialisation processing enabled by Axis SOAP engine are collectively defined as messaging process.
    - The execution of insertion command can further be divided into additional three sub-processes: delegating request to authorization service to check user's authority to perform insertion on per usage record basis; validating usage record against mandatory element configuration; rendering usage record node into an appropriate persistent object and making data persistence.
- An extra process, the aggregation process, is triggered when job usage record is detected during aggregate accounting process.

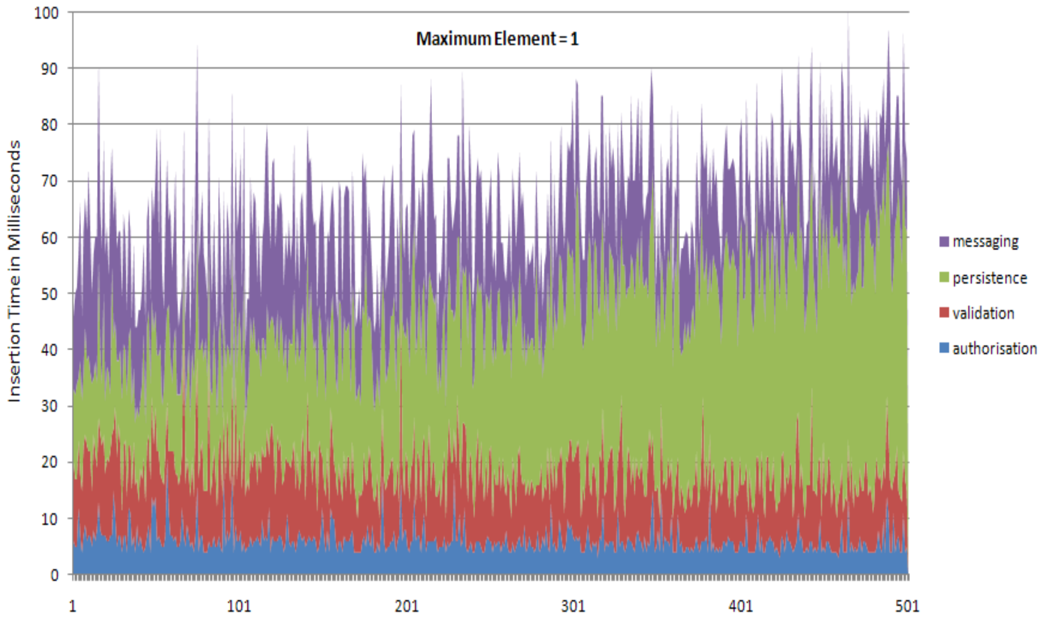
As summarised in Table 4.3, the average performance of authorisation, messaging and validation processes are similar with slight difference less than 0.008 second. Comparing to job accounting model, aggregate accounting models exhibits worse performance mainly because of additional complexity introduced on the data persistence process. On receiving an insertion request of an aggregate usage record, the WLCG-RUS system runtime requires check whether there is an existing aggregate usage record using same aggregate strategy. In the case of WLCG anonymous aggregate strategy for example, the WLCG-RUS runtime is required to the existence of an aggregate usage record with certain month/year, certain VO and certain executing site. If an existing record found, the WLCG-RUS runtime is then add usage information to the existing record, and change the aggregation starting and ending time accordingly. Therefore the data persistence process introduces average 0.02 second overhead. In the aggregate accounting model with runtime aggregation, additional 0.003-second overhead is introduced by the enforcement of the WLCG anonymous aggregation strategy. However this figure can be quite different depending on the complexity of an aggregation strategy implementation.

Table 4-3: Comparison of unit performance of job accounting model, aggregate accounting (without runtime aggregation) and aggregate accounting (with runtime aggregation)

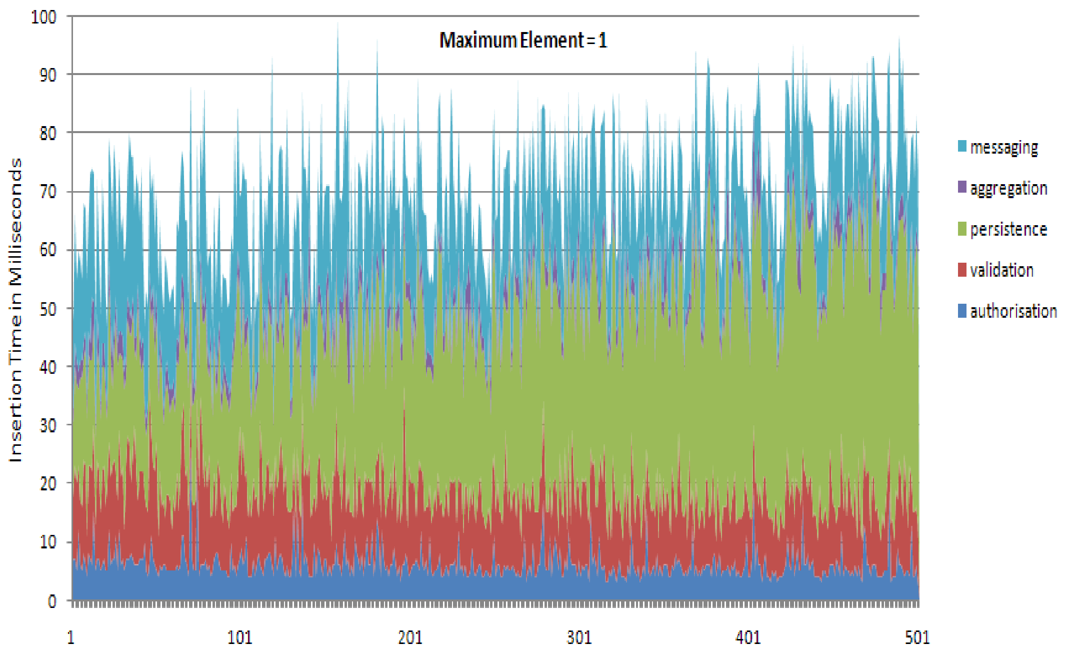
Accounting Model	**Average Costs (Time in Milliseconds)**					
	Authorisation	**Validation**	**Aggregation**	**Persistence**	**Messaging**	**TOTAL**
job accounting	6.334	13.330	-	3.858	18.924	42.446
Aggregate Accounting (no runtime aggregation)	6.266	12.542	-	28.880	18.146	65.834
Aggregate accounting (with runtime aggregation)	6.194	13.298	3.48	30.868	18.492	72.332


Maximum Element = 1
Insertion Time In Miliseconds
100
90
80
70
60
50
40
30
20
10
0
1
101
201
301
401
501
messaging
persistence
validation
authorization

(a)


Maximum Element = 1
Insertion Time in Milliseconds
100
90
80
70
60
50
40
30
20
10
0
1
101
201
301
401
501
messaging
persistence
validation
authorisation

(b)



(c)

Figure 4.17: (a) Unit performance of job accounting model (b) Unit performance of aggregate accounting model (no runtime aggregation) (c) Unit performance of aggregate accounting model with runtime aggregation.

4.5.3 Insertion Performance

As discussed in section 4.4.1, the WLCG-RUS system runtime can be configured to accept one or more usage records per insertion transaction. The number of usage records per transaction is also called bulk size. The first part of the insertion performance test is to evaluate the WLCG-RUS system performance with different bulk size. In this test, the client machine continuously inserts 35,000 job usage records to the WLCG-RUS server. Successive execution time is logged when finishing insertion of 5,000, 10,000, 15,000, 20,000 25,000, 30,000 and 35,000 usage records. As the performance plot described in Figure 4.18 and Figure 4.19, the insertion time decreases gradually with the increasing bulk size until the bulk size is 10, and then increases exponentially. Based on the test results, the maximum elements should be set between 10 and 15 in order to gain optimal insertion performance.

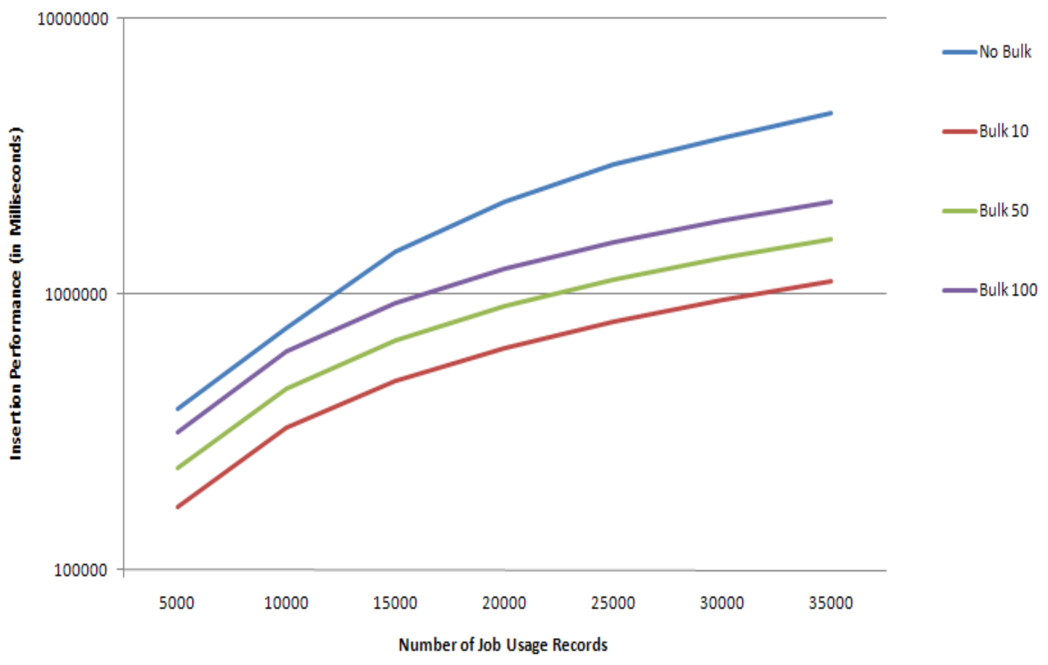
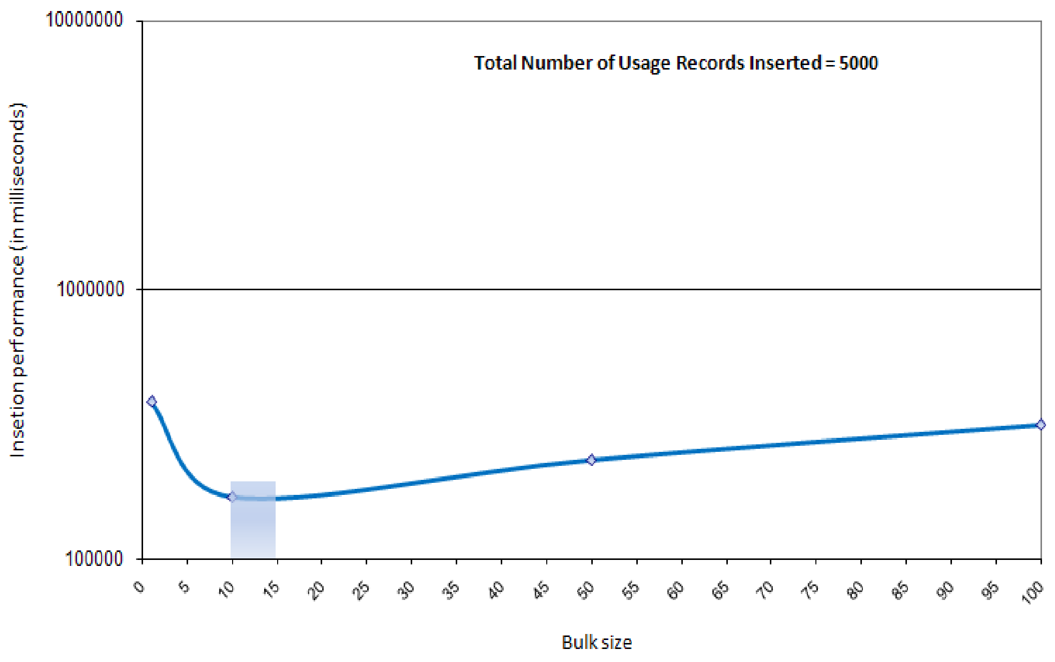
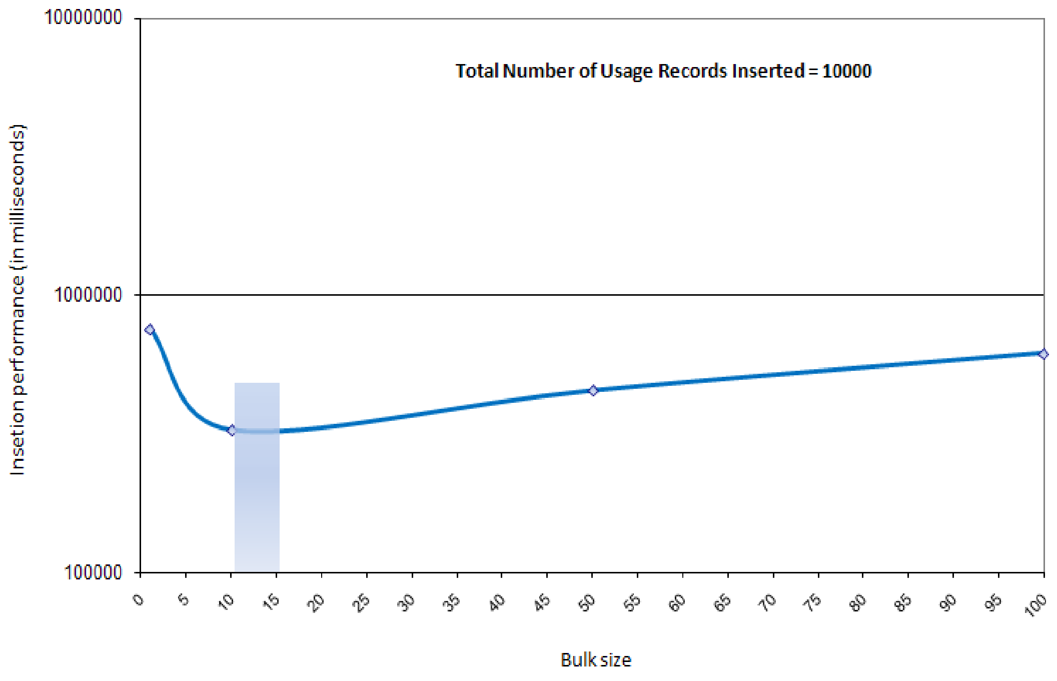


Figure 4.18: Comparisons of insertion time against different granularities of usage records per transaction.

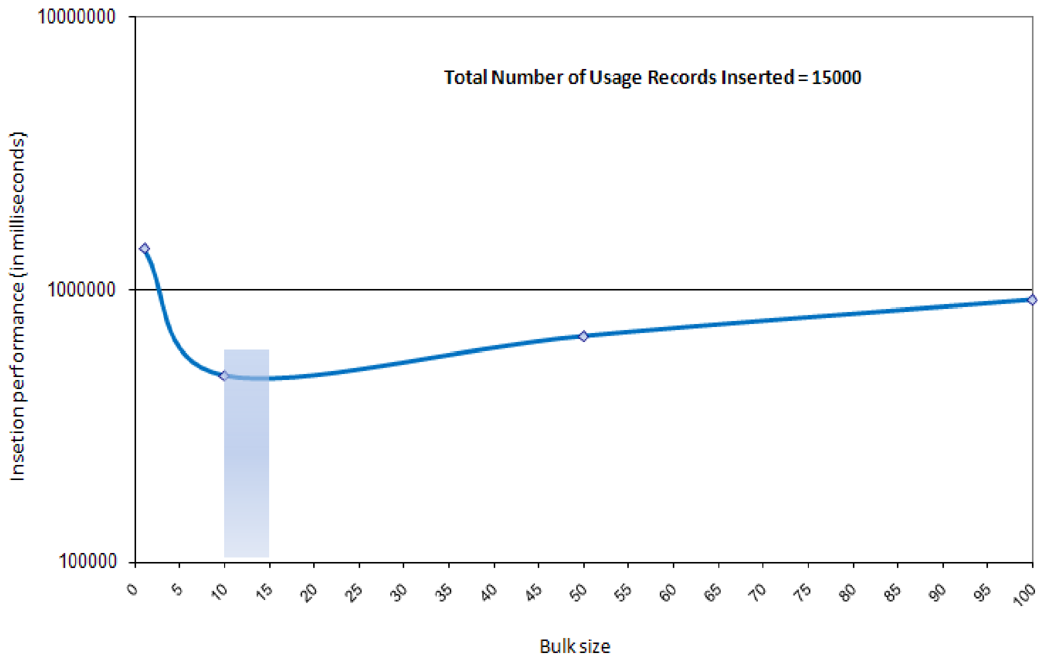


(a)



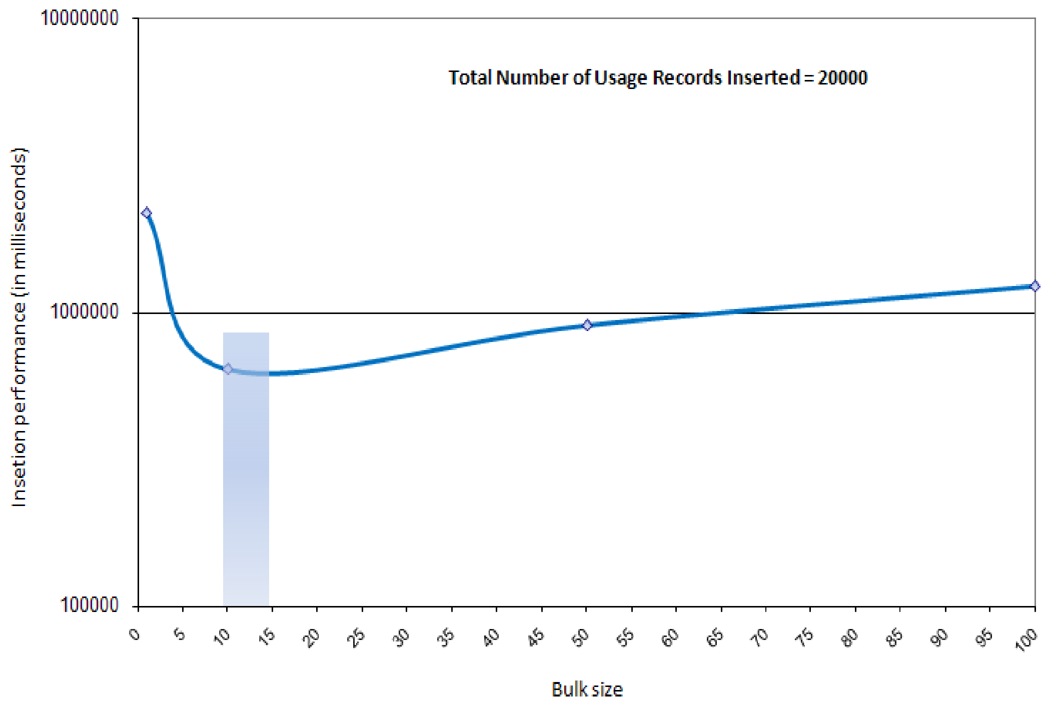
Total Number of Usage Records Inserted = 10000
Insetion performance (in milliseconds)
10000000
1000000
100000
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100
Bulk size

(b)

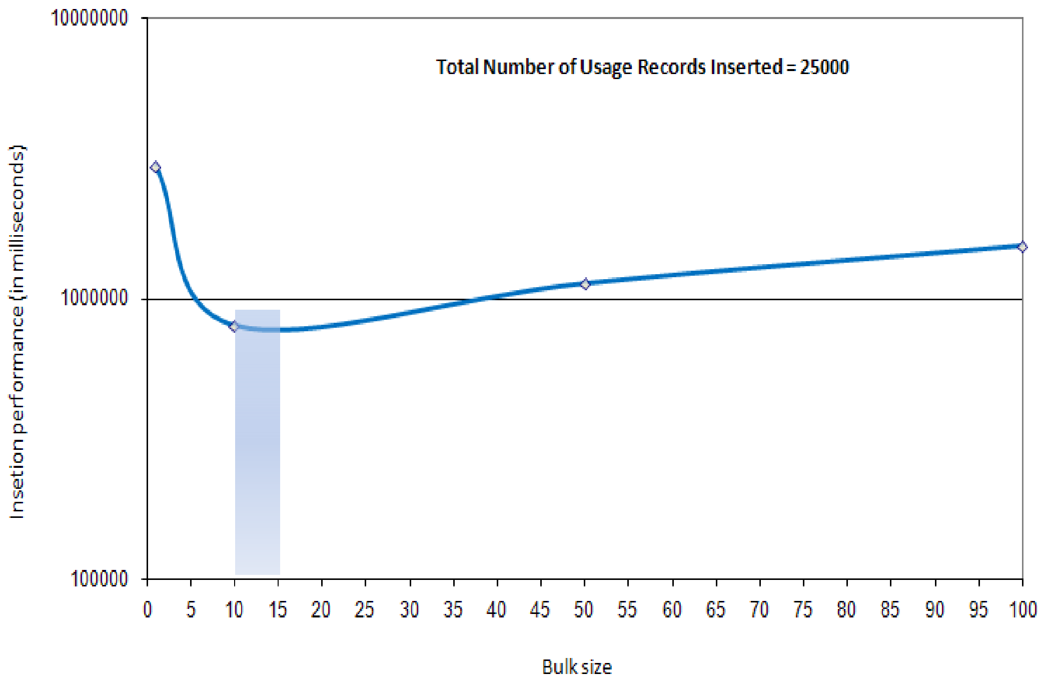


Total Number of Usage Records Inserted = 15000
Insetion performance (in milliseconds)
10000000
1000000
100000
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100
Bulk size

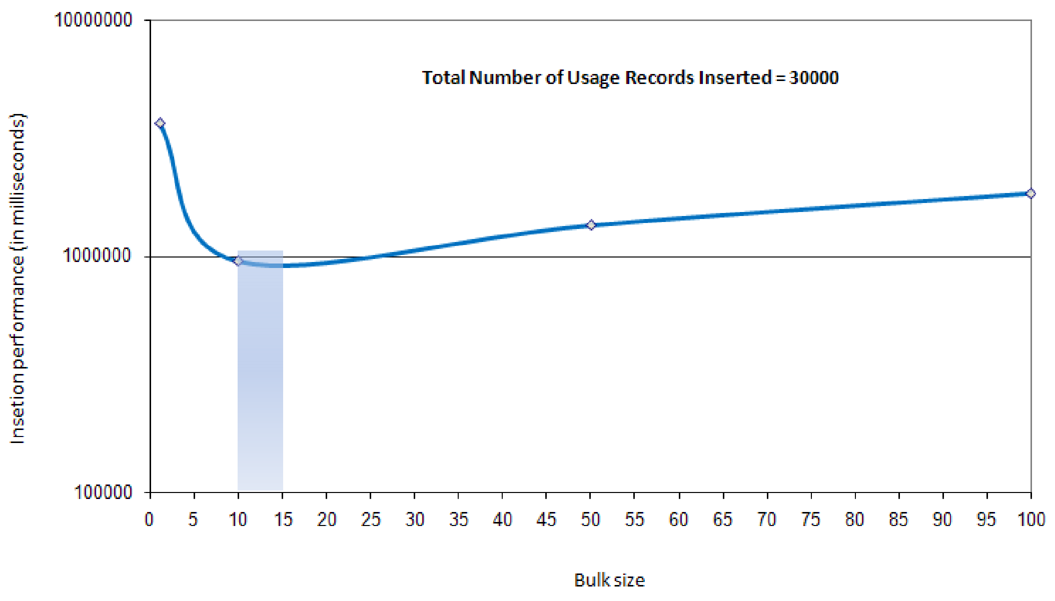
(c)


10000000
1000000
100000
Insetion performance (in milliseconds)
Total Number of Usage Records Inserted = 20000
0
5
10
15
20
25
30
35
40
45
50
55
60
65
70
75
80
85
90
95
100
Bulk size

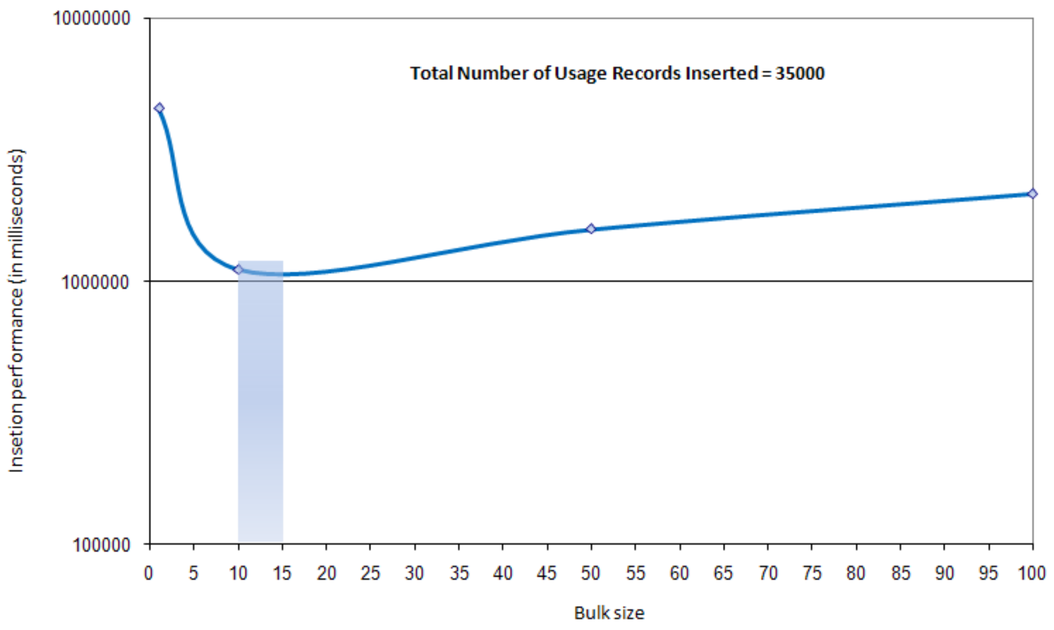
(d)


10000000
1000000
100000
Insetion performance (in milliseconds)
Total Number of Usage Records Inserted = 25000
0
5
10
15
20
25
30
35
40
45
50
55
60
65
70
75
80
85
90
95
100
Bulk size

(e)



(f)



(g)

Figure 4.19: (a) insertion performance of 5,000 usage records against bulk size (b) insertion performance of 10,000 against bulk size (c) insertion performance of 15,000 usage records against bulk size (d) insertion performance of 20,000 against bulk size (e) insertion performance of 25,000 usage records against bulk size (f) insertion performance of 30,000 against bulk size (g) insertion performance of 30,000 usage records against bulk size.

The WLCG-RUS system can be deployed in two ways in the context of the WLCG accounting process. It can be either deployed at the GOC centre as a singleton entry point or hierarchically deployed at each regional site responsible for region-wide accounting purposes while streaming accounting data to the main WLCG-RUS server at GOC. For both cases, the WLCG-RUS system is required to serve multiple client requests at a time. In order to figure out the performance of WLCG-RUS system when dealing with multiple client requests simultaneously, and find out which way is of best performance for the WLCG accounting process, a multi-threading test is conduced to evaluate WLCG-RUS system performance against different number of client threads. As the performance plot illustrated in Figure 4.20, the WLCG-RUS system performance decreases with the increasing number of client threads. In the case of 100 client threads insert usage records at same time, the total time cost for insertion of 35,000 usage records reaches 2.6 hours (0.27 second per transaction), comparing to 1.26 hours (0.13 second per transaction) when using a single client thread. In the case of WLCG accounting, it is better to adopt the hierarchical deployment manner, with multiple WLCG-RUS server deployed at regional sites and one central WLCG-RUS server deployed at GOC site to accept requests from regional sites only. It is worth noting that the performance of WLCG-RUS system may gain better performance when deployed on modem server machine with multi-core or multi-CPUs supports.

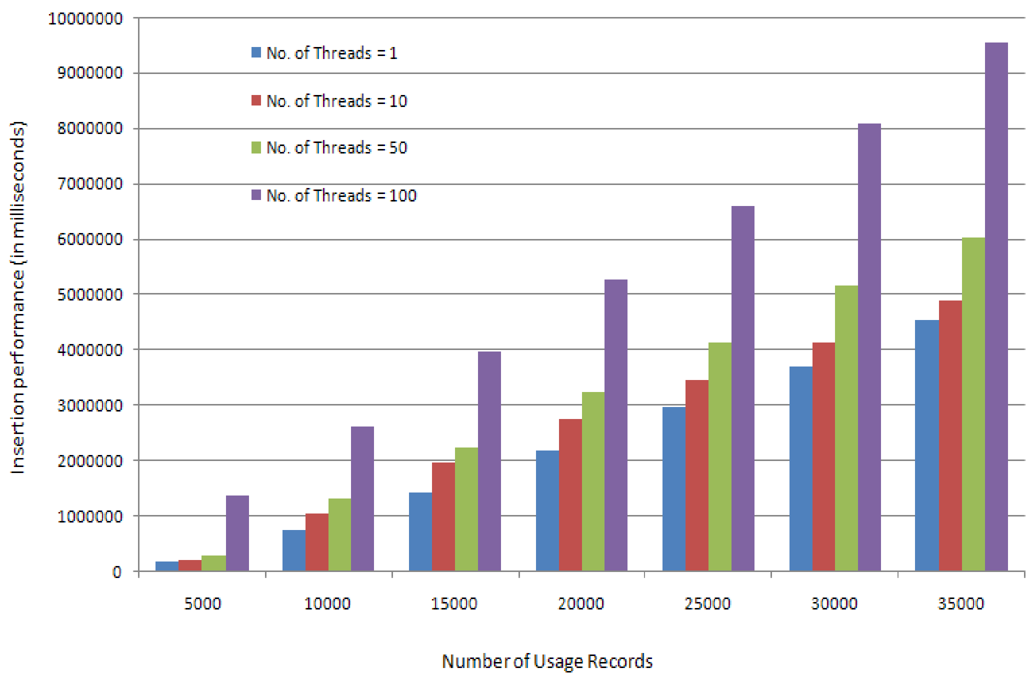


Figure 4.20: insertion performance against the number of simultaneous client threads

4.6 Summary

This chapter described the design and implementation of WLCG-RUS system, which provides an alternative, but standard-compatible, solution for sharing WLCG accounting data from participating sites to GOC centre. The WLCG-RUS system is composed of two subsystems, the RUS service and the WLCG-RUS Admin. The RUS service provides an implementation of OGF RUS service interface definitions. The current RUS service only provides implementations of two RUS service interfaces, the "*RUS::ListMandatoryUsageRecordElements*" and "*RUS::InsertUsageRecords*", which are mainly used for accounting data uploading. The design of RUS service in WLCG-RUS system consists of a set of loose-coupled runtime components, which uses a set of well-defined design patterns, such as factory, strategy, and command design patterns, and exhibits well-defined internal interfaces for custom implementation. Rather than performing off-line aggregation as current WLCG accounting process, the RUS service also allows runtime aggregation and proposed a standard aggregate usage record representation. The WLCG-RUS provides a Web-based administrative interface for site managers and the WLCG-RUS system administrator to performance host management, user management and system management activities. This chapter also provided detailed performance evaluations, which provide development guidance for developers who are intended to use WLCG-RUS framework while providing custom implementations of runtime component units, as well as deployment guidance for decision makers who are considering deploying the WLCG-RUS system as part of an accounting system.

Chapter 5

Design of Grid Resource Usage System Middleware

Standardisation is of high importance on enabling interoperability between independently developed accounting systems. The development of the WLCG-RUS system has presented an exemplary standard-compatible solution for sharing accounting data across heterogeneous accounting systems in the multi-Grid environment of WLCG project. The WLCG-RUS system implemented some functional components as defined in the JISC-proposed accounting framework [132] mainly for uploading accounting data, which is however not functional enough to support various high-level application scenarios, such as usage monitoring, Grid economy, and usage policing. Besides, the design of the WLCG-RUS system uses reverse engineering upon existing WLCG accounting schema making it limited to be reused for accounting purposes on other Grid projects. Lessons were also learned from the RUS specification based on implementation of the WLCG-RUS system. Particularly there are no standard supports to aggregate accounting models in the current RUS specification. The content of this chapter aims at addressing these issues by introducing a refined RUS specification and an implementation of JISC-proposed framework as a middleware solution, the Grid Resource Usage System (GRUS), which makes it easy to migrate custom accounting system to be standard compatible with minimum re-engineering efforts. The design of GRUS middleware reuses WLCG-RUS system components where appropriate. Relevant publications of the work conducted in this chapter include [171] [172].

5.1 Introduction

The JISC proposed generic accounting framework (section 3.6) is a recommendation based on an analysis of stakeholders and their requirements. It was designed to assist development and deployment of a Grid accounting system based on standard specifications. Standardisation is of high importance in the sense of maximising interoperability between independently developed accounting systems, especially for accounting in such Multi-Grid environment as WLCG. The development of the WLCG-RUS system presented how a standard-compatible solution contributed to exchanging accounting data across heterogeneous accounting systems in a consistent manner. Standardisation also makes it easy to migrate high-level applications from one accounting system to another through exhibiting a set of public and common service interfaces. Since most production Grid projects have their own accounting system deployed, the JISC proposed accounting framework (see section 3.6) recommended a loosely coupled component architecture that allows extensions and customisations for adaption to local accounting environment while preventing duplicate efforts on common functional requirements.

Although the WLCG-RUS system implemented some functional components as defined in the JISC-proposed accounting framework mainly for uploading accounting data, which is however not functional enough to support various high-level application scenarios, such as usage monitoring, Grid economy, and usage policing as listed in section 3.2. Besides the aggregate accounting enabled within the WLCG-RUS system only allows specific and predefined aggregate strategy to be applied for streaming accounting data into the RUS service. Higher flexibilities should be allowed to enable custom aggregate strategies to be defined on per transaction basis, especially for query transactions. For example, a VO manager may be interested in getting query results of total CPU usage of a specific VO for last month, while it is also able to get query results of how much memory quota is used as a Grid user. In this case, different aggregate strategies should be automatically generated and applied to individual query transaction. Moreover, the WLCG-RUS system was motivated to reuse WLCG accounting data model, making it limiting to be reused for accounting purposes on other Grid projects, which have custom accounting data formats defined.

Lessons were also learned from the RUS specification based on implementation of the

WLCG-RUS system. Particularly there are no standard supports to aggregate accounting models in the current RUS specification. This chapter proposes a refined RUS service interface definitions, as published in the draft Resource Usage Service Core WS-I rendering specification [171], which deals with observed issues of the current standard, and provides a middleware solution, the Grid Resource Usage System (GRUS)[172], which makes it easy to migrate custom accounting system to be standard compatible and minimises re-engineering efforts on existing accounting systems. The design of the GRUS middleware extends and reuses WLCG-RUS system components where appropriate.

5.2 Requirement Analysis

This section discusses refined and advanced design requirements that are necessary to provide a middleware solution for the JISC proposed accounting framework.

5.2.1 Use Cases

Besides roles and use cases discussed in section 4.2, two additional roles were identified from the perspective of query usage records and are intended to be supported in the GRUS system as extensions to WLCG-RUS system in particular for query of resource usage. Detailed use cases are illustrated in Figure 5.1 and listed in Appendix B.2.

Grid User

A Grid user, the end user of a Grid computing system, can be an ad-hoc user or belongs to a Virtual Organisation (VO). In latter case, the Grid user is also known as the VO member. In order to use the GRUS system, a Grid user must hold a valid X.509 certificate signed by CAs that are recognised by a GRUS system instance. A Grid user in the context of the GRUS system has privileges to query resource usage records of jobs owned by the user through standard RUS interface definitions.

Virtual Organisation Manager

The Virtual Organisation (VO) manager has been recognsied as an important role who is reponisble for managing user membership of a particular VO. In the context of accounting, the VO manager has authority to view resource usage information of jobs executed by members of a managed VO, as well as historic resource usage summaries.

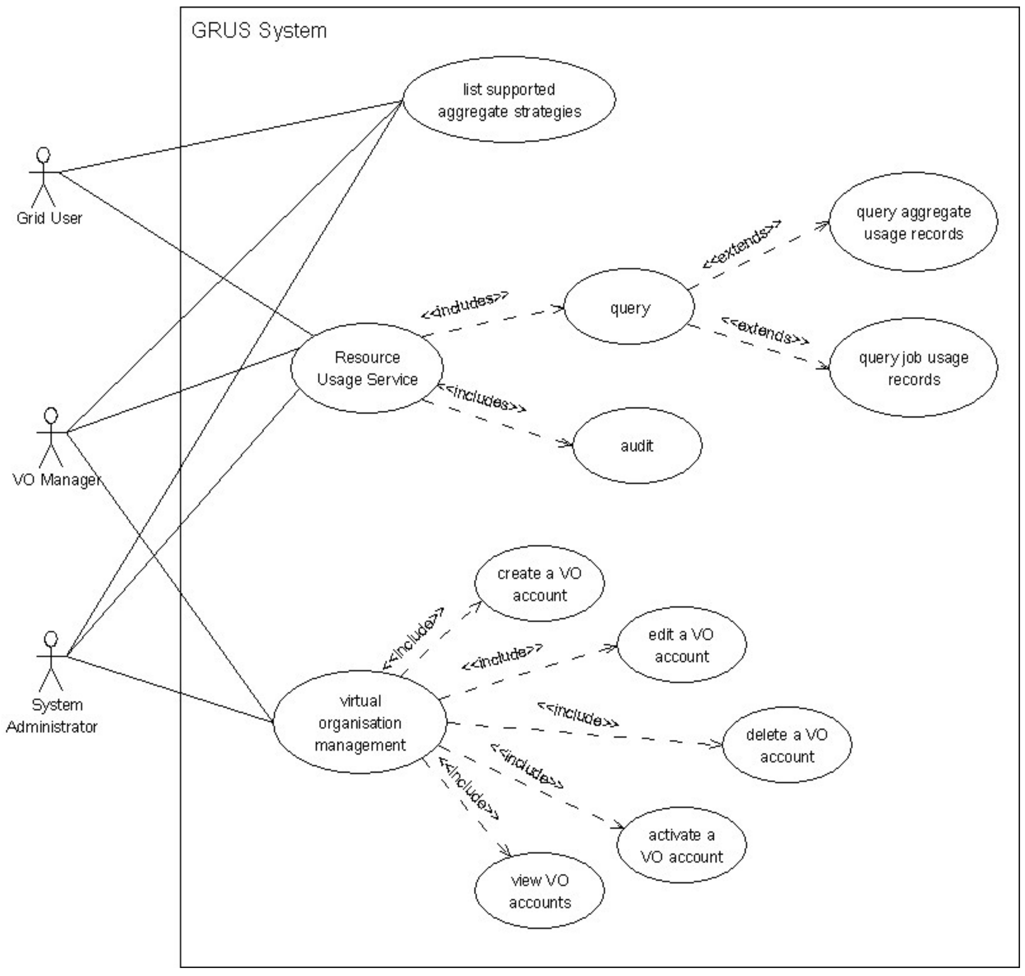


Figure 5.1: Additional use cases that the GRUS system is expected to implement based on existing WLCG-RUS framework

5.2.2 Requirements

From the above advanced use cases, the following design requirements for the GRUS middleware are extracted in addition to those of the WLCG-RUS system as discussed in section 4.2.1.

Capability or Functional Requirements

Apart from functional requirements identified in the design of WLCG-RUS, the following capabilities are to be enabled in GRUS middleware.

1. Query Accounting data

Key to the design of GRUS system is to allow the Virtual Organisation manager and Grid end users to query usage records through standard RUS service interface definition, specifically the "*RUS::extractUsageRecords*" service interface. This RUS extraction logic should ensure certain flexibility in two senses. In the case of query without runtime aggregation, the query operation should allow flexible queries on both complete usage record instances and partial usage information set. The query operation should also provide runtime aggregation facilities. Rather than applying a predefined and specific aggregation strategy, the query operation should allow the requestor to define or specify a preferred aggregation strategy for a particular transaction. Finally, the query operation design of the GRUS system needs to provide a solution to deal with the situation of potentially large volumes of query results triggered by a complex query. Under this circumstance, the query operation should allow returning query results to the clients through multiple transactions by dividing results into chunks.

2. Virtual Organisation Management

A VO manager is able to query VO-specific usage records through the GRUS system. In order to enforce authorisation policies at runtime, the GRUS system must provide a registry mechanism enabling a VO manager to register one or more managed VO accounts. The VO management facility is also expected to provide manageability interfaces for system administrator to view, edit, and remove VO registry entries.

Interface Requirements

3. Internal Interface

In accordance with internal interfaces defined in the WLCG-RUS system for custom implementation of the RUS insertion runtime, the GRUS system is intended to define internal interfaces for custom implementations of RUS extraction runtime.

4. Service Interface

Besides interface requirements identified for the WLCG-RUS system design, another important design goal of the GRUS system is to cope with the deficiencies of RUS service interfaces, in particular for integration of aggregate accounting facilities. Also additional service interfaces are to be defined where necessary.

Data Requirements

5. Internal Data

Rather than reusing the WLCG accounting schema as an internal data representation, the GRUS system should be able to adapt to any accounting data representation as defined by local accounting systems, and allows implementation of custom mapping rules for runtime transformations between internal and external standard formats.

Security and Privacy Requirements

The security design requirements of the GRUS system share the requirements as specified in section 4.2 for the WLCG-RUS system design.

Other Requirements

6. Usability

In addition to other requirements listed in WLCG-RUS system design, the design of GRUS system exhibits an extra requirement on usability. As a middleware solution the GRUS system should be not only end user oriented but also developer oriented. In this sense, the GRUS middleware must provide easy-to-implement facilities for development of custom solutions upon local accounting environment.

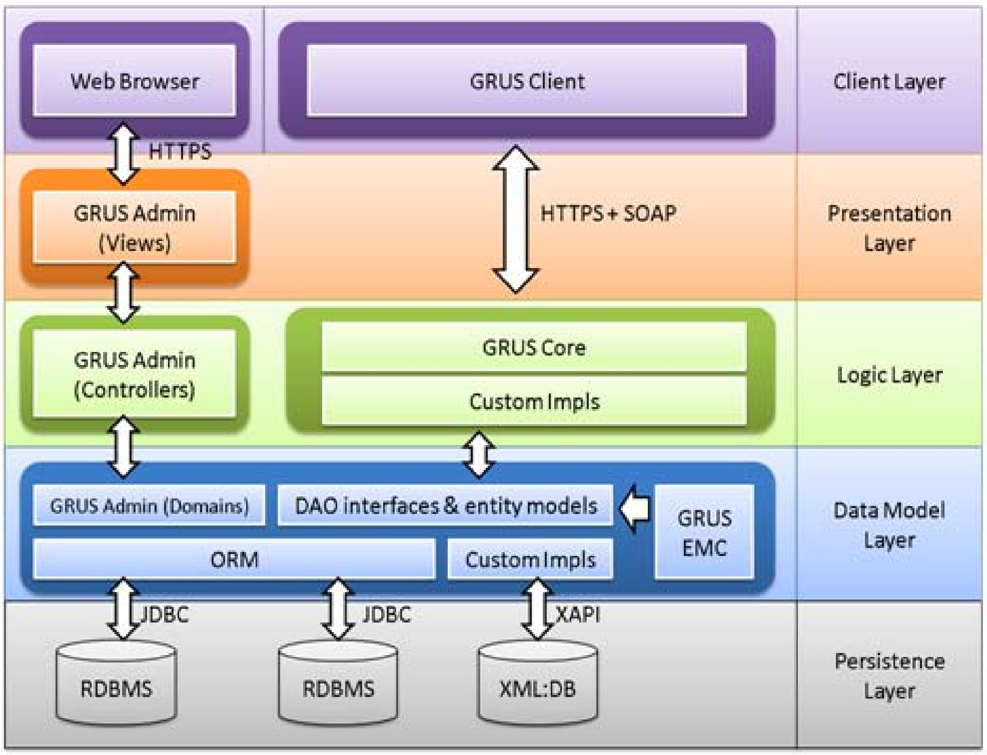


Figure 5.2: Layered component architecture of GRUS middleware

5.3 Design

This section discusses the design details of the GRUS middleware, including the system architecture design and composing subsystems or components.

5.3.1 System Architecture

In accordance to the WLCG-RUS system design, GRUS system is composed of two subsystems, the GRUS Admin Web application and the GRUS service. The GRUS Admin application extends WLCG-RUS Admin and provides VO management facilities for VO managers. The GRUS service provides a development framework for customising the implementation of the RUS service endpoint. The design of the GRUS system is based on a layered component architecture as presented in Figure 5.2, and consisting of runtime components across five bottom-up layers: the persistence layer, the data model layer, the logic layer, the presentation layer, and the client layer.

Persistence Layer

The persistence layer contains the data structures, including the accounting data structures of the GRUS service and management data structures of GRUS Admin subsystem. This persistence layer is designed to use a relational database for data persistence. Custom implementations may also use other types of data storages, e.g. XML databases, for accounting data.

Data Model Layer

The data model layer contains necessary elements that link object data to the relational database structures. Rather than using specific internal usage data representations, the GRUS service is intended to enable automated persistence of data model objects to various internal accounting data structures of relational databases through the Object-Relational Mapping (ORM) mechanism. The design of the data model layer also exhibits a higher level abstraction using Data Access Object (DAO) pattern. Each data model object has an associated DAO, which exhibits common and primitive Create, Read, Update, and Delete (CRUD) data operations. Customised data operations can be defined by extending abstract DAO interfaces. By using DAO design pattern, it is also possible to define custom DAO implementation upon data stores other than relational database, the native XML database for example. In order to ease custom implementations, a utility tool, the Entity Model Compiler (EMC), is also provided to take any XML standard schema and generate data models and DAO source codes.

A set of domain objects are also defined at the data model layer for the GRUS Admin Web application, which extends manageability facilities defined in WLCG-RUS Admin system with additional VO management functionality.

Logic Layer

The logic layer defines a GRUS core framework, which provides runtime support of the RUS logics. The core framework consists of a set of runtime components, the design of which utilises object-oriented design patterns and exhibits well-defined internal interfaces. The pluggable design of the GRUS core framework allows implementations to choose to customise one or more runtime components according to local accounting requirements.

The logic layer also shows a GRUS Admin component that extends WLCG-RUS Admin and defines controllers for VO management facilities. The controllers act upon the underlying model objects and refresh changes of domain objects to the GRUS Admin views.

Presentation Layer

The presentation layer defines views for GRUS Admin Web application and provides a Web-based interface to end users, VO managers, site managers, and system administrator. The GRUS Admin views consist of a sequence of Web pages and presentation style sheets.

Client Layer

Both the GRUS Admin and the GRUS service provide client-side interfaces. The GRUS client provides command-line interfaces mainly for sharing and querying accounting data through standard RUS Web service interfaces. Authorised users may also execute appropriate management tasks through the GRUS Admin Web portal and Web browser.

5.3.2 Detailed System Design

This section describes the design details of the GRUS system components, including redesign of RUS service interfaces, messaging protocols for runtime aggregate query, EMC code generator, GRUS core framework, and the GRUS Admin Web application.

Redesign of RUS Interface Definitions

Based on the developmental experiences on the WLCG-RUS system and feedbacks from other RUS implementations, i.e. SGAS and DGAS-RUS, there are some non-trivial issues identified and summarised as follows:

- The current RUS service interface definitions are too reliable on OGF UR specification making it hard to use the other standard usage record representations, in particular OGF AUR draft specification that has been recognised as an important data representation for aggregate accounting purposes. Besides, the OGF UR specification has a narrowed scope based only on batch job CPU usage metrics. It is understood that a single OGF UR is not enough to accommodate accounting representation of other resource types, such as storage, network, and even application-specific resources. Therefore the RUS service interface definitions should be flexible to accept various usage record formats in compatible to existing, as well as emerging, standard resource usage schemas.

- Although the current RUS specification does not restrict internal storage format for usage record persistence, it does specify individual usage records retrieved from a RUS endpoint should in the Resource Usage Record Format (RURF), which encapsulates a RUS-wide unique global identity, an OGF UR instance, and record modification histories. This data type definition as query result implies potential issues. First of all, the query operation results add more transportation payloads with additional record histories appended to individual usage record even though the client are not interested in. Performance can be further undermined when a complex query returns a large number of usage records. Secondly, the insertion operation as defined in the RUS specification returns a list of RUS record identities raised by a RUS endpoint for successfully inserted usage records. The list of RUS record identities are not meaningful to the client in the sense of indicating unsuccessfully inserted usage records.

- The query interface definition, "*RUS::extractRUSUsageRecords*", only returns the complete RURF instance. In many cases, query clients only interested in partial or fragmental usage information set, CPU usage information for example. Therefore the query interface definition of RUS should allow flexible query on both complete usage records and partial information sets. Furthermore, the current query operation returns all usage records evaluated against the query term within a single transaction, which is inappropriate for complex query with large volumes of query results.

- There have been long discussions about the usefulness of encapsulating modification operations within the RUS specification, since the most important feature of accounting is to provide accurate resource usage information, which provides a proof of how Grid resources had been utilised. In reality, it is unlikely these generated accounting data will be changed or updated. Since most sites or GOC keeps a local repository of collected accounting data, it is more straightforward and secure for a system administrator to update or remove accounting data through local database management system.

Having identified issues of current RUS Service Interface Definitions (SIDs), we collaborated with OGF RUS working group and refined RUS SIDs as the outcomes of group discussion in OGF 20. In the middle of 2007, the first proposed draft of the RUS Core Interface Definition Language (IDL) specification [173] was released with changes or add-on features applied to observed issues. Major changes of SIDs made within the proposed RUS Core specification are listed in Table 5-1 and summarised as follows.

- Rather than defining a separate RUS usage record representation, the RUS service is intended to reuse existing OGF UR and ensure flexibility on other emerging standard usage record schemas.

- Insertion request message defines an extension element, the “*<xsd:any>*” element, that can used to pass any usage records in the format other than OGF UR to a RUS service endpoint.

- The extraction service interface is renamed as “*RUS::extractUsageRecords*” and accepts a filter expression that can be constrained to be a Boolean predicate as well as ad-doc support projections depending on a RUS endpoint implementation. The extraction service interface definition also allows iteration through query result set in a similar way as defined in WS-Enumeration specification [174].

- The interface definition, “*RUS::extractRUSRecordIds*”, is removed from RUS specification.

- Service interfaces related to modifying usage records are simplified with a single service interface definition, the “*RUS::modifyUsageRecords*”, which accepts an updating expression, e.g. XQueryUpdate[175], and returns operational results.

- A single service interface is also defined for deleting usage records matched by evaluation filter expression input.

- A RUS service endpoint may apply different standard or custom dialects for query, updating and deleting usage records. For example, a RUS service endpoint may be implemented using XQuery[176] dialect for query, XQueryUpdate and SQL dialects for updating, and XPath dialect for expressing a Boolean predicate for deletion. A client may get supported operation-dialect pairs of a RUS service endpoint through proposed "*RUS::listSupportedDialects*" interface.

- A new operation is also proposed in RUS Core specification and allows a client to audit record creation or modification history through the "*RUS::extractRecordHistory*" interface.

Table 5- 1: A Comparison of Service Interface Definition between RUS specification (version 1.7) and Proposed RUS Core specification

Function	**Service Interface Definition**	
	RUS Specification (version 1.7)	**RUS Core Specification**
configuration	RUS::ListMandatoryUsageRecordElements	RUS::ListMandatoryUsageRecordElements
		RUS::listSupportedDialects
Insertion	RUS::insertUsageRecords	RUS::insertUsageRecords
Extraction	RUS::extractRUSUsageRecords	RUS::extractUsageRecords
	RUS::extractRUSRecordIds	
Updating	RUS::incrementUsageRecordPart	RUS::modifyUsageRecords
	RUS::modifyUsageRecordPart	
	RUS::replaceUsageRecords	
Deletion	RUS::deleteRecords	RUS::deleteUsageRecords
	RUS::deleteSpecificRecords	
Auditing	-	RUS::extractRecordHistory

Messaging Extensions for Aggregate Accounting

In order to qualify RUS Core specification in particular in the sense of integrating aggregate accounting at RUS runtime and query fragmental usage information sets, the GRUS messaging framework defines a set of SOAP header data types and reuses some of the control headers and WS-Enumeration extensions defined within Web Service Management (WS-Management) specification proposed by Distributed Management Task Force (DMTF). Definitions and usages of these extensions together with RUS service interface definitions are described as follows.

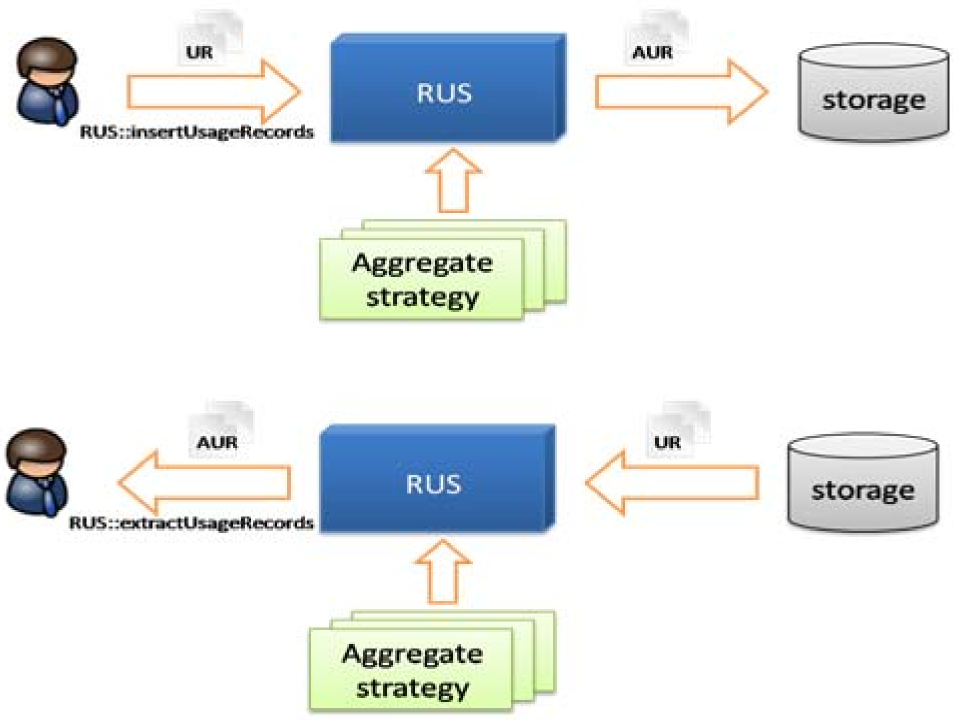


Figure 5.3: Runtime Aggregation Process at RUS Insertion and Extraction Runtime

Aggregation processes typically take place at RUS insertion and extraction runtime. As Figure 5.3, the runtime aggregation process at RUS insertion runtime accepts multiple job usage records in the OGF UR format and aggregates them into one or more OGF AUR instances by applying a specific aggregate strategy, while the runtime aggregate process during RUS extraction runtime summarises filtered job usage records using a specific aggregate strategy and returns standard OGF AUR instances to the client. Compared to runtime aggregation at RUS insertion runtime, further flexibility should allow the client to define custom aggregation rules that apply to current query transaction only.

In order to enable runtime aggregation in a RUS compatible way, the GRUS message framework defined a "*grus:AggregateStrategy*" element, which is used to specify a pre-defined aggregate strategy or ad-hoc aggregation rules. A client initiates a RUS request with runtime aggregation by placing the "*grus:AggregateStrategy*" element inside the SOAP header section as follows:

Example: In the following example template, runtime aggregation is enforced by a RUS service with proper aggregate strategy information attached to request message header.

```
<s:Envelope ...>
<s:Header ...>
...
<wsa:Action mustUnderstand="true">
http://schemas.ogf.org/rus/2007/09/core/extractUsageRecordRequest">
<grus:AggregateStrategy id="strategy-id">
<grus:Interval>"{hour|week|day|month|year}"</grus:Interval>
<grus:Entity ...>xsd:QName</grus:Entity>*
</grus:AggregateStrategy>
</s:Header>
<s:Body ...>
<rus:ExtractUsageRecordsRequest>
...
</rus:ExtractUsageRecordsRequest>
</s:Body>
</s:Envelolpe>
```

The following definitions provide additional, normative constraints on the "*grus:AggregateStrategy*" information model:

- grus:AggregateStrategy

  The optional header element contains a global unique identity of a specific aggregate strategy, and child elements for specifying aggregation rules. On receiving a request with runtime aggregation, a RUS service endpoint must apply a pre-existing aggregate strategy identified by the identity value of this header element or composing an aggregate strategy dynamically. Dynamic aggregate strategy allows a client to define custom aggregate rules for a particular extraction transaction.

- grus:Interval

  This element defines the aggregation intervals. There are five defined intervals including day, week, month, year, and hours.

- grus:Entity

  This element may occur more than once to declare the qualified name of one or more Grid resource entities to be grouped. The element can be further restricted by placing attribute values.

The definition of aggregation strategy header introduces flexibility in specifying a specific aggregation strategy as well as defining custom aggregate stragety at runtime. The aggregation strategy header can be specified along with RUS insertion and extraction logics. The following gives example request messages in the context of WLCG accounting allowing:

- a host to populate job usage records to the WLCG anonymous summary usage repository by specifying the WLCG anonymous aggregation strategy in the RUS::insertUsageRecords request message. Each aggregation strategy implementation has a global unique identity (e.g. urn:grus:strategy:aggregation:wlcg-user). On receiving the request message, the RUS service runtime looks up and instantiates an aggregate strategy instance, which then performs runtime aggregation upon job usage records embedded within the request message.

Example: insertion request message with aggregation strategy header

```
<s:Envelope ...>
<s:Header ...>
...
<wsa:Action mustUnderstand=”true”>
http://schemas.ogf.org/rus/2007/09/core/insertUsageRecordR
equest”>
<grus:AggregateStrategy
id=”urn:grus:strategy:aggregation:wlcg-user” />
</s:Header>
<s:Body ...>
</s:Body>
</s:Envelolpe>
```

- a VO manager to query WLCG job usage repository and generate summry usage information by specifying custom aggregation rules. A general-purpose aggregation strategy (urn:grus:strategy:aggregation:dynamic) is defined to apply user-defined aggregation rules upon query results. On receiving the following extraction request, a RUS endpoint firstly filters usage records of jobs in the VO name of CMS, and creates an instance of the general-purpose aggregation strategy, which then generates aggregate usage records summarised on the per-user, per-VO, and per-month basis.

Example: extraction request message with custom aggregation rules

```
<s:Envelope ...>
<s:Header ...>
...
<wsa:Action mustUnderstand=”true”>
http://schemas.ogf.org/rus/2007/09/core/extractUsageRecordR
equest”>
<grus:AggregateStrategy
id=”urn:grus:strategy:aggregation:dynamic” />
<grus:Interval>Month</grus:Interval>
<grus:Entity>urf:UserIdentity</grus:Entity>
<grus:Entity
urf:description=”VOName”>urf:Resource</grus:Entity>
</s:Header>
<s:Body ...>
<rus:ExtractUsageRecordsRequest>
<rus:Filter   dialect=”http://www.w3.org/TR/1999/REC-xpath-
19991116”>
/urf:UsageRecord[urf:Resource[@urf:description=’VOName’]
</rus:Filter>
</rus:ExtractUsageRecordsRequest>
</s:Body>
</s:Envelolpe>
```

The GRUS messaging also reuses some non-functional control headers and extension elements to WS-Enumeration as defined within WS-Management specification mainly for the purpose of fragmental and optimised query usage records. A RUS implementation may restrict appearance of following control header and extension elements as demonstrated in following example request message.

Example: The following example request message integrates control headers and enumeration extensions as defined within WS-Management specification

```
<s:Envelope ...>
<s:Header ...>
...
<wsman:OperationTimeout>xsd:long</wsman:OperationTimeout>
<wsman:RequestTotalItemsCountEstimate />
</s:Header>
<s:Body>
<rus:ExtractUsageRecordRequest>
<rus:EndTo>wsa:EndpointReferenceType</rus:EndTo>
<rus:Expires>wsen:ExpirationType</rus:Expires>
<rus:Filter dialect="xsd:anyURI">xsd:any</rus:Filter>
<rus:MaxElements>xsd:PositiveInteger</rus:MaxElements>
<rus:EnumerationContext>wsen:EnumerationContextType
</rus:EnumerationContext>
<wsman:Filter dialect="xsd:anyURI">xsd:any</rus:Filter>
{xsd:any}
</rus:ExtractUsageRecordRequest>
</s:Body>
</s:Envelope>
```

The following definitions provide additional, normative constraints on the usage and interpretation of control headers and enumeration extensions embedded within request messages:

- wsman:OperationTimeout
  This optional header element defined within WS-Management specification is reused as a quality-of-service constraint. A RUS implementation may define a default maximum operational timeout to prevent system performance from being undermined by complex requests. The value of timeout can also be specified by a client for time-critical requests. If a RUS service endpoint does not support this element, the endpoint may either ignore this control header or return a "*rus::UnsupportedFault*" message if it must be understood. When a request is processed beyond the specified interval limit, a RUS service endpoint should kill the server process and return a "*rus:ProcessingFault*" with the "time-out" reason.

- wsman:RequestTotalItemsCountEstimate

This optional element is the control header defined by WS-Management specification to indicate a RUS service endpoint should return an estimate of total number of items associated with a specific RUS extraction transaction.

- rus:EnumerationContext
  If a RUS service endpoint supports iterative query results, an enumeration context should be established and encapsulating necessary information for iterative query results. Usage of this element in a RUS extraction request results in the return of query result sets made by previous transaction.

- wsman:Filter
  Although RUS Core specification explained that the expression specified by "*rus:Filter*" may either be a Boolean predicate to return complete usage records or support ad-hoc projections to return fragmental usage information set, the GRUS message framework restricted the "*rus:Filter*" expression to be a Boolean predicate only to filter complete usage record instances, while reusing the "*wsman:Filter*" extension to specify projection information. The definition of "*wsman:Filter*" by WS-Management specification is same as "*rus:Filter*".

On successful execution of above example request message, a RUS service endpoint composes a response message as following example:

Example: The example response message that integrates control headers and enumeration extensions as defined within WS-Management specification

```
<s:Envelope ...>
<s:Header ...>
...
<wsman:TotalItemsCountEstimate>
xsd:nonNegativeInteger
</wsman:TotalItemsCountEstimate>
</s:Header>
<s:Body>
<rus:ExtractUsageRecordResponse>
<rus:OperationResult>
...
</rus:OperationResult>
<urf:UsageRecords />
<rus:EnumerateContext>
```

```
wsen:EnumerationContextType
<rus:EumerateContext>
<rus:Expires>xsd:DateTime|xsd:Duration<rus:Expires>
<wsman:Items>
<wsman:XmlFragment>
...
</wsman:XmlFragment>
</wsman:Items>
<wsman:EndOfSequence />
{xsd:any}
</rus:ExtractUsageRecordResponse>
</s:Body>
</s:Envelope>
```

The following definitions provide additional, normative constraints on the interpretation and processing of control headers and enumeration extensions as a RUS extraction response message.

- wsman:TotalItemsCountEstimate
  This optional header indicates that the client requested the total item count in request message, and includes the estimated total number of query results within the response message.
- rus:EnumerationContext
  If a RUS service endpoint supports enumeration, an enumeration context is established at service side and returned to the client with necessary information for follow-up query transactions.

- rus:Expires
  An instance of enumeration context has a limited lifetime, which is specified by the client in a RUS extraction request message or defaulted by a RUS service endpoint. Embedding this element in a response message helps the client understand how long the query results made would live.

- wsman:Items
  This optional element is defined as a container of one or more enumerable elements in WS-Management specification. The element used in GRUS message framework to contain fragmental query results only. Other complete usage records as query results should be placed in a schema-specific container. For example,

OGF UR instances should use "*urf:UsageRecords*" element, while OGF AUR instances should use "*aur:AggregateUsageRecords*" element.

- wsman:XmlFragment

  This optional element is used to contain a single fragmental query result. The main usage of this element in GRUS messaging framework is to wrap text fragments. A "*wsman:XmlFragment*" can only be a single fragment, and embedded as a child element of the "*wsman:Items*" element.

- wsman:EndOfSequence

  This element defined within WS-Management specification is used in GRUS message framework to notify the client that all query result set have been iterated.

Apart from these extensions of RUS messaging, GRUS messaging framework also defined a new service interface, the "*GRUS::listSupportedAggregateStrategies*", which is used query operation-strategy pairs implemented within a RUS service endpoint. The detailed GRUS messaging data type schema and Web service interface schema are given in Appendix D.

Entity Model Compiler

The implementation of WLCG-RUS system defined three data model objects, which are constructed by accepting standard OGF UR or OGF AUR instances, and are mapped to three predefined WLCG accounting schema through ORM. Each data model object has an associated DAO implementation that is triggered to save instantiated data model objects into relational database at the RUS insertion runtime. However these three data model objects are reversely engineered and hard coded based on WLCG accounting schemas, making them hard to be reused for other accounting systems. Besides, WLCG-RUS data model objects merely realise one-way transformation, i.e. transforming standard usage record instances into WLCG accounting data for the purpose of publishing accounting data. Rather than defining specific internal accounting data representation, the design of GRUS system is intended to enable high-level flexibility in allowing a RUS service endpoint to reuse any custom accounting data representations. To be more specific, a RUS implementation based upon GRUS framework should be able to transform usage record instances in XML format into relational accounting data representation at RUS insertion runtime, and render relational accounting data representation into standard external XML representation at RUS extraction runtime.

The GRUS system introduced a utility tool, the Entity Model Compiler (EMC), which provides a solution to XML persistence into relational backend. The EMC tool concentrates on following requirement and functionalities:

XSD Driven	XML instances must be validated against certain XML schema.
Relational Backend	XML data are to be persistent in a relational database.
Entity Oriented	A data model object must be of entity type, which has its own database identity. An entity may have one or more relationships to other entities, in particular one-to-one, many-to-one, one-to-many, many-to-many relationships.
Auto Generation	The EMC is a code generation engine that produces a list of interfaces, abstract classes that encapsulate runtime rendering functions, and DAO artefacts on per entity basis.
Customisation	Generated artefacts can be customised by developers to provide default entity model implementations and ORM mappings to local relational data formats.

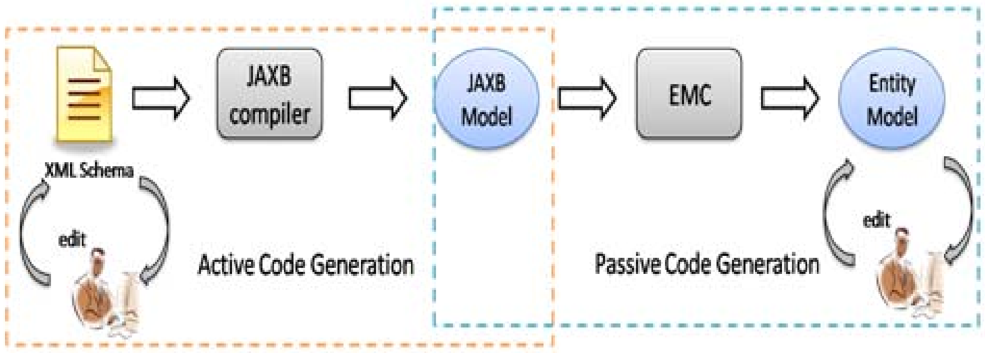


Figure 5.4: The EMC code generation pattern in combination with the active code generation pattern of JAXB binding compiler

There are two widely adopted techniques for code generation: active generation and passive generation. Both techniques involve a code generator component, that accepts an input and produce source code files, also known as artefacts. Common input sources includes code model represented in Unified Modelling Language (UML), data files (e.g.

XML files), and source code files. In an active generation system, the generated artefacts are only affected by modification of input source. Passive code generation, on the other hand, refers to the code generation process being one off and non-repeatable. The generated codes are normally imported into a project to be extended by developers. As Figure 5.4, the design of EMC uses passive code generation pattern that takes artefacts generated by JAXB compiler. Although JAXB compiles an XML schema into a set of Java classes, which are essential Plain Old Java Objects (POJO), these Java classes are not customisable. Therefore the EMC is intended to generate following extensible entity artefacts:

- An interface that contains a list of getter and setter methods;
- An abstract class contains:
    - Zero or more entity fields that have "one-to-one", "one-to-many", "many-to-one" or "many-to-many" relationships to current entity;
    - An empty constructor;
    - A constructor that takes the JAXB typed object as parameter;
    - A "toJaxbBindingType" method that returns JAXB binding type;
- An entity DAO interface;
- A DAO Factory abstract class with creator methods of each generated entity DAO;

The code generation process enforced by EMC is composed of two sub-processes, entity model generation and DAO model generation process. As Figure 5.5, the process of entity model generation starts from loading user inputs, including a list of entity qualified names, target full package path, and namespace-package mappings. The process tries to load the JAXB-generated Java class into memory and process JAXB field or property annotations by iterating every declared field in the JAXB class model. The processing of individual field and associated annotations results in adding setter and getter methods to the entity interface model, establishing relationships to other entities, and adding appropriate statements to constructor and "*toJaxbBindingType*" methods of abstract class model. The process of generating DAO models produces a DAO interface for each declared entity, and an abstract DAO factory class that contains creator methods for each generated entity DAO model. Finally these generated DAO and factory source codes are written into a specific source code directory specified by the user inputs.

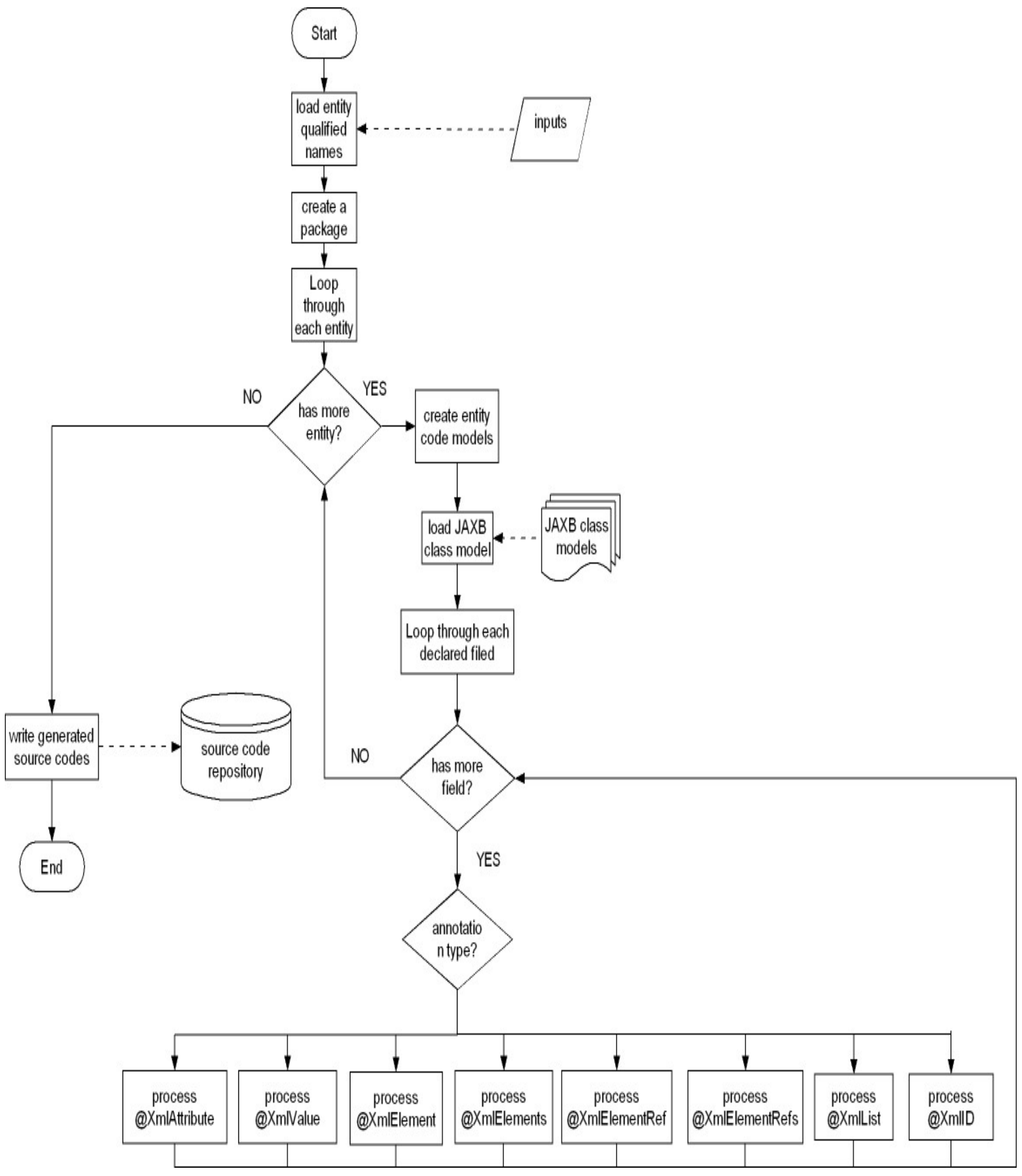


Figure 5.5: Flowchart of entity model generation process

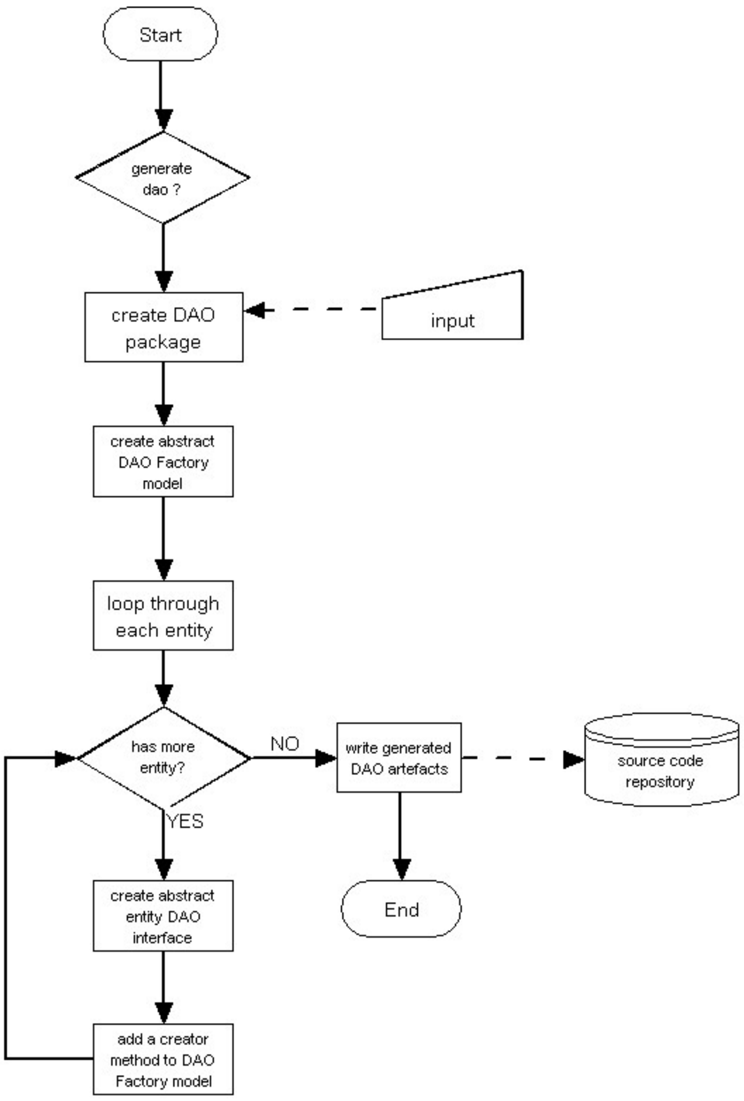

Start
generate dao ?
create DAO package
input
create abstract DAO Factory model
loop through each entity
has more entity?
NO
write generated DAO artefacts
source code repository
YES
create abstract entity DAO interface
End
add a creator method to DAO Factory model

Figure 5. 6: Flowchart of DAO model generation process

GRUS Core

The GRUS Core provides a development framework consisting of a package of abstract functional and loose-coupled components, each of which exhibits well-defined internal interfaces. A RUS service endpoint may provide custom RUS logic implementations by customising one or more functional components. As the class diagram given in Figure 5.7, these components are categorised and organised into five packages, each of which targets at accomplishing certain functionality. Like the design of WLCG-RUS runtime, the key component of GRUS core framework is the command, which interacts with other internal component implementations to fulfil RUS runtime logics, in particular RUS insertion, extraction, modification, and deletion operations. Therefore, the command factory class defines four creator methods to instantiate RUS operation-specific command implementations. A command exception class is also defined and throwable during the execution of a command instance. The GRUS core framework reuses the Authorisation and DAO components defined in WLCG-RUS system, with additional abstract methods defined within the Generic DAO interface mainly for data updating and deletion. Filter component introduced within GRUS core framework can be used in combination with RUS extraction, updating and deletion logics. There are two types of general-purpose filters defined, the query filter and update filter. The query filter can be further divided into two subcategories, projection-oriented filter and predicate filter. A predicate filter is used usage records according to certain predicate expressions, while the projection-oriented filter is used to get fragmental information set from filtered usage records. An implementation of predicate filter acts upon a DAO object and returns completed usage records by applying certain query terms. A SQL filter, for example, can be triggered at RUS extraction runtime to query usage records matched by evaluation of one or more "where" statements. The returned usage records can be further processed by a projection-oriented filter, e.g. XPath filter, to get fragmental usage information.

GRUS Admin

The design of GRUS Admin extends WLCG-RUS and provides additional VO management facilities for both system administrator and VO manager. As Figure 5.8, a user that takes the role of VO manager is redirected to VO management view through which new VO accounts can be created, managed, and deleted. These VO registration entries are to be fed into authorisation service at RUS extraction runtime, i.e. a VO manager can only access usage records of managed VOs. The system manager only also access VO management facilities and have full control of all registered VO accounts in a GRUS system.

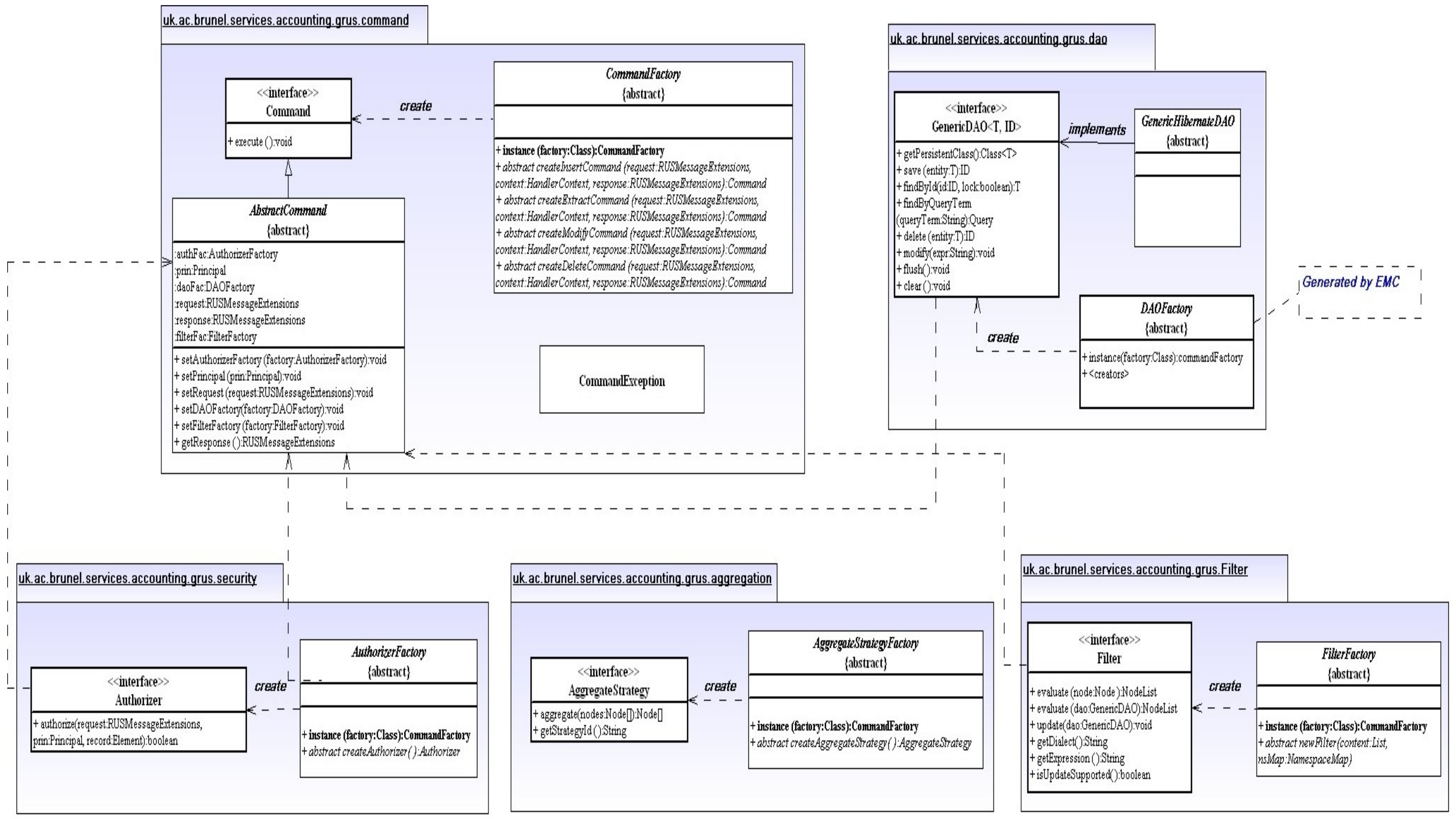

uk.ac.brunel.services.accounting.grus.command
<<interface>> Command
+execute ():void
create
CommandFactory {abstract}
+instance (factory:Class):CommandFactory
+abstract createInsertCommand (request:RUSMessageExtensions, context:HandlerContext, response:RUSMessageExtensions):Command
+abstract createExtractCommand (request:RUSMessageExtensions, context:HandlerContext, response:RUSMessageExtensions):Command
+abstract createModifyCommand (request:RUSMessageExtensions, context:HandlerContext, response:RUSMessageExtensions):Command
+abstract createDeleteCommand (request:RUSMessageExtensions, context:HandlerContext, response:RUSMessageExtensions):Command
AbstractCommand {abstract}
:authFac:AuthorizerFactory
:prin:Principal
:daoFac:DAOFactory
:request:RUSMessageExtensions
:response:RUSMessageExtensions
:filterFac:FilterFactory
+setAuthorizerFactory (factory:AuthorizerFactory):void
+setPrincipal (prin:Principal):void
+setRequest (request:RUSMessageExtensions):void
+setDAOFactory(factory:DAOFactory):void
+setFilterFactory (factory:FilterFactory):void
+getResponse ():RUSMessageExtensions
CommandException
uk.ac.brunel.services.accounting.grus.dao
<<interface>> GenericDAO<T, ID>
+getPersistentClass():Class<T>
+save (entity:T):ID
+findById(id:ID, lock:boolean):T
+findByQueryTerm (queryTerm:String):Query
+delete (entity:T):ID
+modify(expr:String):void
+flush():void
+clear ():void
implements
GenericHibernateDAO {abstract}
DAOFactory {abstract}
+instance(factory:Class):commandFactory
+<creators>
create
Generated by EMC
uk.ac.brunel.services.accounting.grus.security
<<interface>> Authorizer
+authorize(request:RUSMessageExtensions, prin:Principal, record:Element):boolean
create
AuthorizerFactory {abstract}
+instance (factory:Class):CommandFactory
+abstract createAuthorizer ():Authorizer
uk.ac.brunel.services.accounting.grus.aggregation
<<interface>> AggregateStrategy
+aggregate(nodes:Node[]):Node[]
+getStrategyId ():String
create
AggregateStrategyFactory {abstract}
+instance (factory:Class):CommandFactory
+abstract createAggregateStrategy ():AggregateStrategy
uk.ac.brunel.services.accounting.grus.Filter
<<interface>> Filter
+evaluate (node:Node):NodeList
+evaluate (dao:GenericDAO):NodeList
+update(dao:GenericDAO):void
+getDialect():String
+getExpression ():String
+isUpdateSupported():boolean
create
FilterFactory {abstract}
+instance (factory:Class):CommandFactory
+abstract newFilter(content:List, nsMap:NamespaceMap)

Figure 5.7: GRUS Core Runtime Component Class Diagram

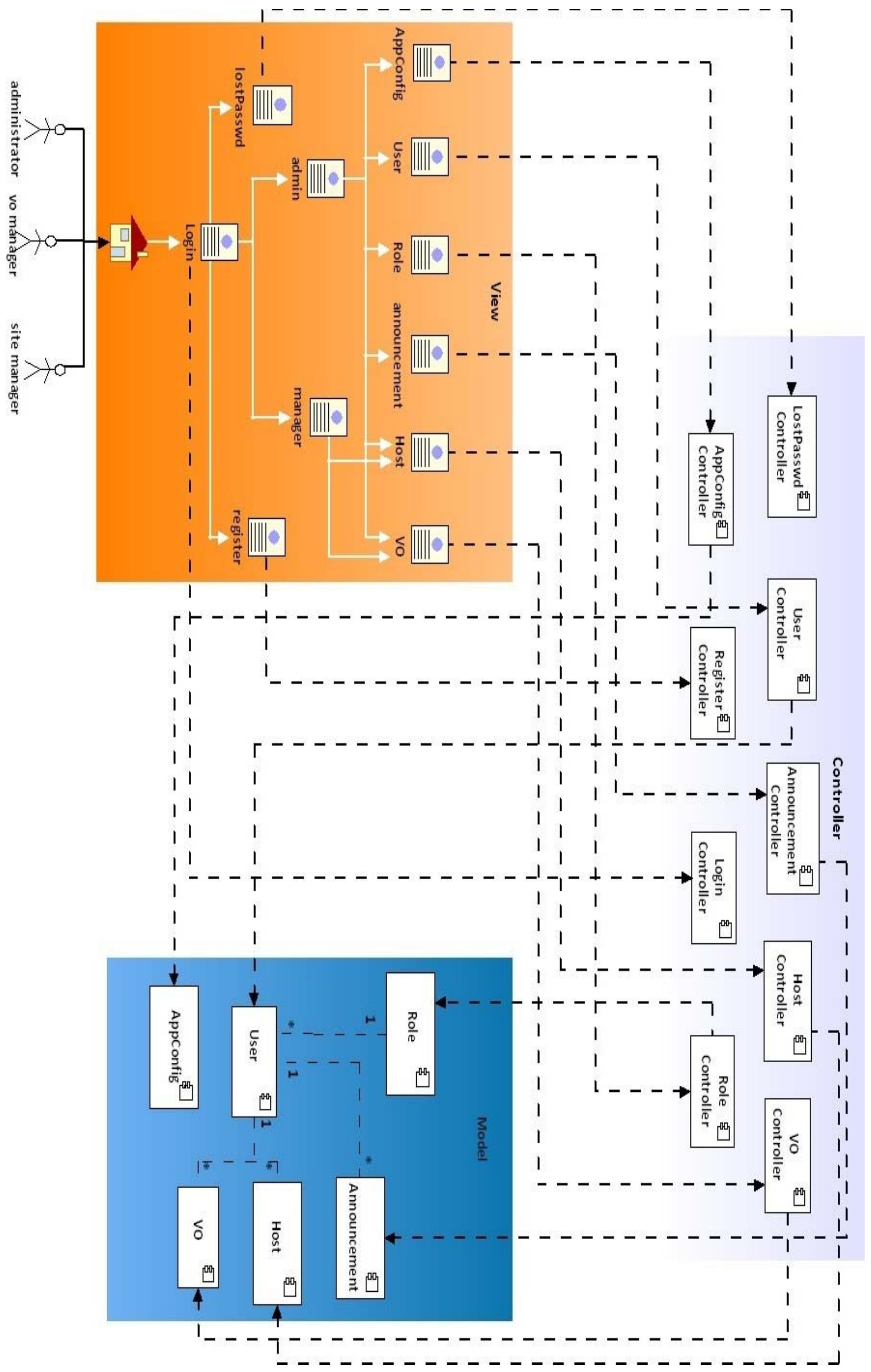


Figure 5.8: GRUS Admin MVC Model

5.4 Implementation

This section describes implementation details of composite components of GRUS system.

5.4.1 Entity Model Compiler

Synopsis

The EMC tool is implemented as a custom Ant [177] task that is to be invoked from the Ant build tool. The EMC task supports the following parameter attributes (Table 5-5).

Table 5-2: Parameter attribute list of EMC task

Attribute	Data Type	Description	Required
destDir	String	The root directory of source codes or artefacts to be generated	Yes
entityModelPkg	String	The package name of entity model artefacts	Yes
daoModelPkg	String	The package name of DAO artefacts	No
generateDAO	Boolean	If specific, the DAO artefacts will be generated and placed in specified DAO model package.	No

The EMC task also supports the following nested element parameters:

classpath The nested <classpath> element(s) is used to specify locations of JAXB-generated classes.

Example Syntax:

```
<classpath>
<pathelement path=”${classpath}”/>
<pathelement location=”lib/example.jar”/>
<classpath>
```

entity The nested <entity> element(s) is used to declare qualified names of target entities. These elements are loaded by EMC task to locate

JAXB-generated class models. The qualified name of an entity is a combination of a namespace and JAXB-generated class model name.

Example Syntax:

```
<entity namespace=”urn:namespace”>
Name of JAXB class name
</entity>
```

NsPkgMapping The element is used to declare custom JAXB namespace-package mappings. The syntax functions exactly as JAXB namespace-package mappings to declare custom packages other than reasoned from namespaces using default package name converter. Values of this element help the EMC compiler to locate appropriate JAXB class model. If this element is omitted, the default package name converter of JAXB is used.

Example Syntax:

```
<NsPkgMapping
namespace=”urn:namespace” prefix=”prefix”
package=”package.full.path” />
```

Worked Example

In order to use EMC tool in Ant build tool, the EMC Java ARchive (JAR) file is required to add class path in a build file and declare a task definition with following statements:

Example: EMC task definition of a build file requires specifying the class path referring to GRUS EMC package file.

```
<taskdef
name=”emc”
classname=”uk.ac.brunel.services.accounting.grus.tool.emc.EMCTask”
<classpath>
 <pathelement path=”${lib.dir}/grus-emc-1.0-SNAPSHOT.jar” />
<classpath>
</taskdef>
```

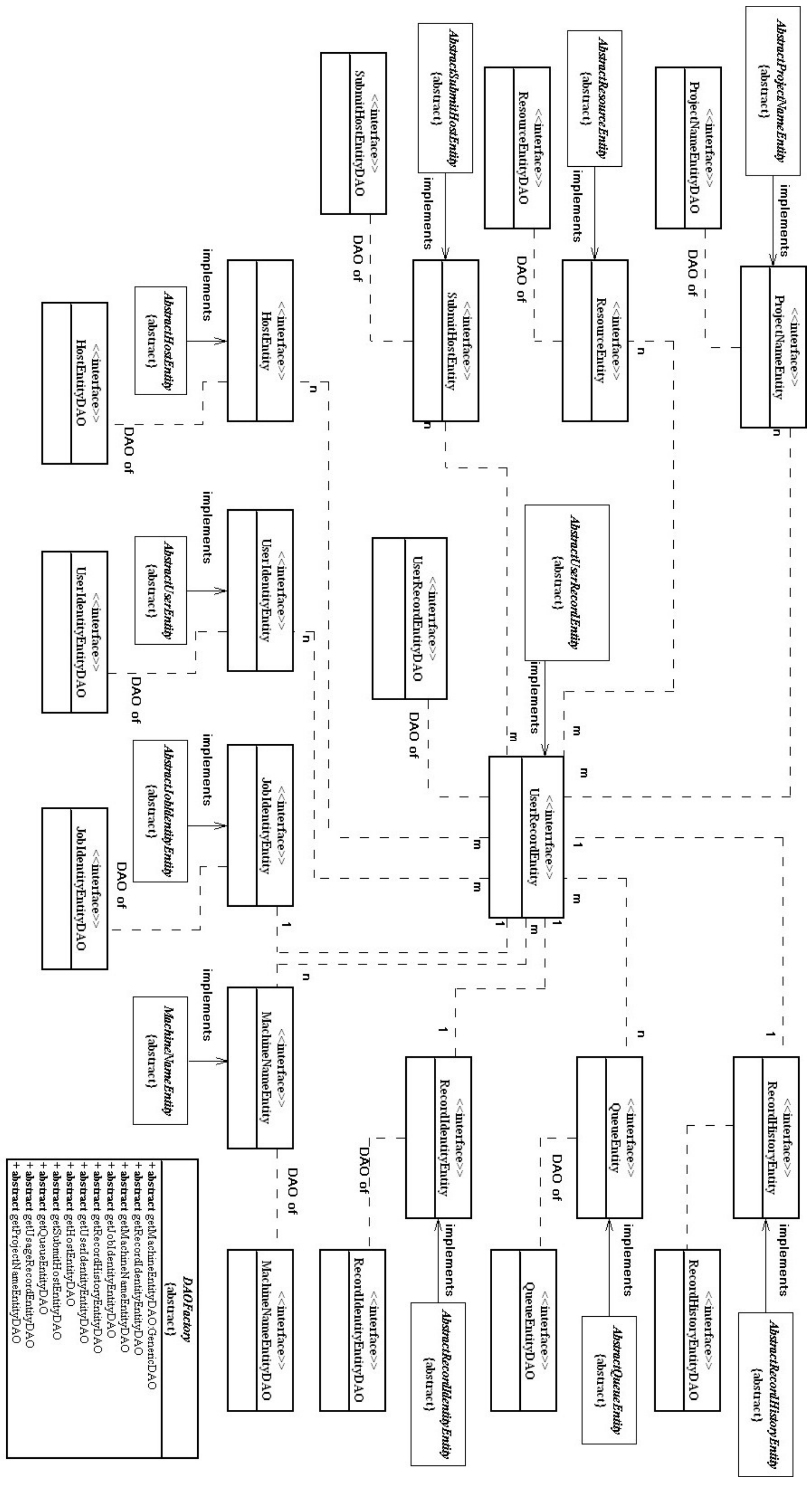
After defining the EMC task in a build file, an Ant target can be defined to generate entity model and DAO model artefacts by invoking EMC tasks, which accepts a set of user-defined parameters and embedded element parameters. The example below defines a “generateEntityDAOModels” target with following statements:

Example: The following example target definition uses EMC task to generate entity model and DAO model artefacts.

```
...
<target name=”genenateEntityDAOModels”
description=”generate Java entity and DAO artefacts”>
<emc destDir=”${src.dir}”
generateDAO=”true”
entityModelPkg=”uk.ac.brunel.services.accounting.grus.datamodel.
urf”
daoModelPkg=”uk.ac.brunel.services.accounting.grus.dao.urf”>
<classpath>
<fileset dir=”${build.dir}/classes” />
<include name=”*.class” />
</fileset>
</classpath>
<entity
namespace=”http://schema.ogf.org/urf/2003/09/urf” prefix=”urf”>
UsageRecordType</entity>
<entity
namespace=”http://schema.ogf.org/urf/2003/09/urf” prefix=”urf”>
Host</entity>
<entity
namespace=”http://schema.ogf.org/urf/2003/09/urf” prefix=”urf”>
SubmitHost</entity>
<entity
namespace=”http://schema.ogf.org/urf/2003/09/urf” prefix=”urf”>
UserIdentity</entity>
<entity
namespace=”http://schema.ogf.org/urf/2003/09/urf” prefix=”urf”>
RecordIdentity</entity>
<entity
namespace=”http://schema.ogf.org/urf/2003/09/urf” prefix=”urf”>
JobIdentity</entity>
<entity
namespace=”http://schema.ogf.org/urf/2003/09/urf” prefix=”urf”>
Resource</entity>
```

```
<entity
namespace=”http://schema.ogf.org/urf/2003/09/urf” prefix=”urf”>
ProjectName</entity>
<entity
namespace=”http://schema.ogf.org/urf/2003/09/urf” prefix=”urf”>
MachineName</entity>
<entity
namespace=”http://schema.ogf.org/urf/2003/09/urf” prefix=”urf”>
Queue</entity>
<entity
namespace=”http://schema.ogf.org/urf/2003/09/urf” prefix=”urf”>
Queue</entity>
<entity
namespace=”http://schemas.ogf.org/rus/2007/09/core/types”
prefix=”rus”>
RecordHistoryType</entity>
</emc>
</target>
...
```

The execution of the above example results in the generation of a set of entity and DAO model artefacts as the class model described in Figure 5.9. The example EMC task defines eleven embedded entities, including entities defined within the OGF UR schema and the record history entity defined within the RUS schema. For each entity, the EMC task generates an entity model interface and an abstract class model, which provides a runtime mapping framework between the instance of an entity model and the JAXB class model. The EMC task also establishes relationship between entities. In this example, a usage record entity has one-to-one relationship to the record identity entity, the job identity entity, and record history entity, while has many-to-many relationships to other generated entities. The DAO generation process also generated a DAO interface on per entity basis and added an associated factory method to the abstract DAO factory class.



AbstractProjectNameEntity {abstract}
<<interface>> ProjectNameEntityDAO
<<interface>> ProjectNameEntity
AbstractResourceEntity {abstract}
<<interface>> ResourceEntityDAO
<<interface>> ResourceEntity
AbstractSubmitHostEntity {abstract}
<<interface>> SubmitHostEntityDAO
<<interface>> SubmitHostEntity
<<interface>> HostEntity
AbstractHostEntity {abstract}
<<interface>> HostEntityDAO
AbstractUserRecordEntity {abstract}
<<interface>> UserRecordEntityDAO
<<interface>> UserRecordEntity
<<interface>> UserIdentityEntity
AbstractUserEntity {abstract}
<<interface>> UserIdentityEntityDAO
<<interface>> JobIdentityEntity
AbstractJobIdentityEntity {abstract}
<<interface>> JobIdentityEntityDAO
<<interface>> MachineNameEntity
MachineNameEntity {abstract}
<<interface>> MachineNameEntityDAO
<<interface>> RecordHistoryEntity
<<interface>> RecordHistoryEntityDAO
AbstractRecordHistoryEntity {abstract}
<<interface>> QueueEntity
<<interface>> QueueEntityDAO
AbstractQueueEntity {abstract}
<<interface>> RecordIdentityEntity
<<interface>> RecordIdentityEntityDAO
AbstractRecordIdentityEntity {abstract}
implements
DAO of
n
m
1
DAOFactory {abstract}
+ abstract getMachineEntityDAO:GenericDAO
+ abstract getRecordIdentityEntityDAO
+ abstract getMachineNameEntityDAO
+ abstract getJobIdentityEntityDAO
+ abstract getRecordHistoryEntityDAO
+ abstract getUserIdentityEntityDAO
+ abstract getHostEntityDAO
+ abstract getSubmitHostEntityDAO
+ abstract getQueueEntityDAO
+ abstract getUsageRecordEntityDAO
+ abstract getProjectNameEntityDAO

Figure 5.9: Example class models of artefacts generated by EMC

5.4.2 GRUS Core

The implementation of GRUS core is based on WiseMan (version 1.0) [178] platform, an open source Java^{TM} implementation of WS-Management specification. It provides a development framework as well as runtime environment for hosting WS-Management compatible Web services. Rather than using third-part Web service hosting environment, such as Axis or Java Web Service Developer Pack (JWSDP) [179], the WiseMan provides its own hosting environment in order to support WS-Addressing[57] compatible SOAP messaging framework and delegate incoming SOAP requests to appropriate request handler using WS-Addressing information. The implementation of GRUS Core extended Wiseman runtime framework and provided a set of support classes that help developers focus on designing custom RUS solutions without dealing with low-level messaging. These support classes provide following functionalities to the developer:

- Providing messaging facilities to marshal and unmarshal RUS messages and GRUS extensions;
- Managing lifecycle of requests being served;
- Runtime aggregation either by applying predefined aggregate strategies or instantiating dynamic aggregate strategy according to user inputs;
- Enumerating large volume of query results;
- Monitoring lifetime of enumeration context and perform clean up when expired;
- Filter query results;
- Mutual authentication and fine-grained access control on per usage records basis;
- Using custom XML-formatted accounting schema other than OGF UR and OGF AUR.

As Figure 5.10, the GRUS Core consists of a set of runtime components (items in blue) and abstract function components to be implemented by developers (items in red). The following list provides an overview of generic runtime events:

- The GRUS servlet keeps listening to transport-level requests. At startup, the servlet loads RUS Core schema and dependencies including GRUS extension schema, OGF UR, OGF AUR schema, etc. and instantiates a singleton GRUS agent instance. The servlet is responsible for serving both HTTP GET and POST requests. Client may query schema and WSDL files through HTTP GET request, while interrogating RUS logics through HTTP POST requests. On receiving a RUS request, the servlet forwards incoming request to the instantiated agent instance and passes a context object that encapsulates necessary information related to current transaction, including the client principal, GRUS handler object, command factory object, DAO factory object, etc., by loading system configurations.

- The GRUS agent acts as a request scheduler and maintains an internal pool for asynchronous tasks. When a RUS request is received, the agent validates request messages against loaded schemas. Once validated, a request dispatcher task is scheduled and placed into the task pool. The lifetime of the request dispatcher task is monitored by a specific timer task, which clean up the task and compose a "wsman:timeoutFault" message returned to the client when the task did not completed until the end of timeout value specified by the "wsman:OperationTimeout" control header.
- GRUS request dispatcher is implemented as a callable task. Its main responsibility is to delegate received requests to appropriate a GRUS handler implementation specified within the context object passed by GRUS servlet.

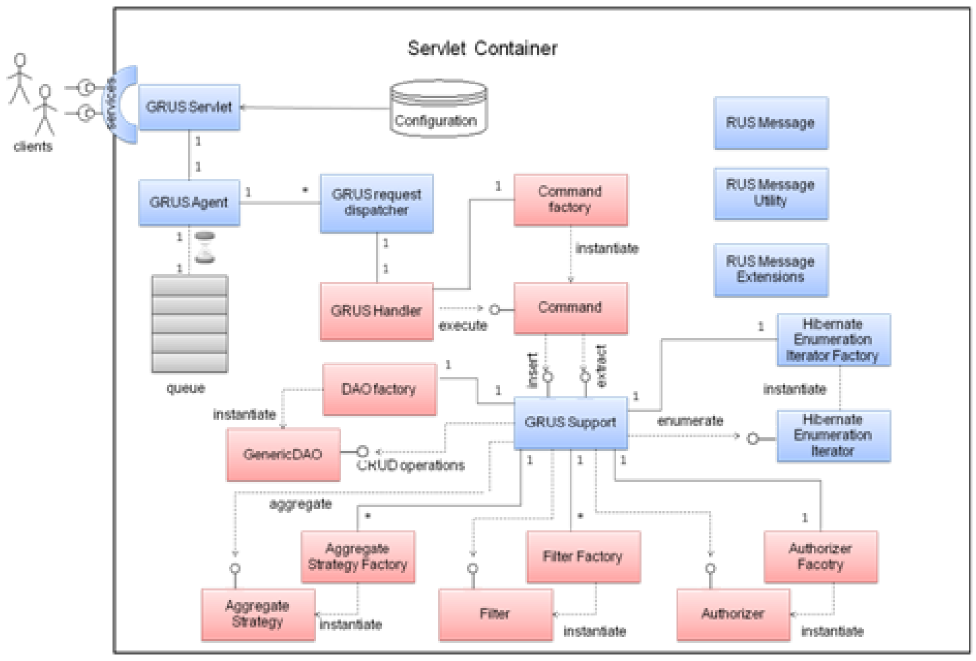


Figure 5.10: GRUS server architecture containing runtime implementations and interactions

- A GRUS handler provides a set of internal interfaces that are triggered according to the action specified within WS-Addressing [57] header information. Developer may provide custom handler solution by implementing the GRUS handler interface. A GRUS handler implementation may use support classes provided by WiseMan framework for real-time resource usage monitoring through standard RUS Core interfaces, or make use of utility functions supplied by GRUS support class for persistent accounting.
- The GRUS support is the main support class used by developers to provide support for their custom handler implementation in the context of persistent accounting. The support class interrogates GRUS runtime component implementations and provide utility functions.
- Finally the GRUS framework also provides a messaging framework consisting of Java representation of RUS Core messages and GRUS header blocks. A utility class is also provided and facilitate implementation developers to create RUS request and response messages.

5.4.3 GRUS Annotations

According to the RUS Core specification, a RUS compatible implementation must as least support XPath (version 1.0) [170] dialect for RUS extraction logic. In order to bridge the gap between XPath and relational backend, GRUS Core provides an XPath2Hql filter that implemented the Filter interface and converted standard XPath expression into Hibernate Query Language (HQL) [180] at runtime by consuming custom mapping of program elements of an entity model implementation to XML Schema construct. GRUS defined a set of mapping annotations based on the Java Specification Request 175 (JSR175) [181], a metadata facility for Java^{TM} programming language. The retention policy of all defined mapping annotations is the RetentionPolicy.RUNTIME, which allows introspection of mapping annotations by XPath2Hql Filter at runtime. These annotations are used in an entity model implementation for:

- Customising the mapping of an entity model to a global XML element;
- Referencing an entity property to another entity model;
- Customising the mapping of a non-entity property to a simple-typed XML element;
- Customising the mapping of a non-entity component to a complex-typed XML element;

The following gives detailed normative synopsis and mapping constraints of defined annotations.

@Entity

This class-level annotation is used to map an entity model class to an XML global element.

Synopsis

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
public @interface Entity{
String name();
String namespace ();
Boolean isRoot () default false;
}
```

Mapping The following mapping constraints must be enforced:

- This annotation is used as a class level annotation. A class model annotated with this annotation must be an entity class that extends one of abstract entity models generated by GRUS EMC.
- The @Entity.name () must be specified to the local name of the target global element.
- The @Entity. namespace () must be specified to the namespace of target global element.
- If isRoot ( ) is true, the entity class model is mapped to a root element.

@EntityRef

This property-level annotation is used to reference an entity property to another entity model.

Synopsis

```
@Target(ElementType.FIELD)
@Retention(RetentionPolicy.RUNTIME)
public @interface EntityRef{
Class<?> type ()
}
```

Mapping The following mapping constraints must be enforced:

- The @EntityRef.type( ) must specify the full class path of referenced entity model class.

@Property

This annotation is a property-level annotation that is used to map a non-entity property to an XML simple content.

Synopsis

```
@Target(ElementType.FIELD)
@Retention(RetentionPolicy.RUNTIME)
public @interface Property {
String name ()default “##default”
String namespace () default “##default”
PropertyType type ( ) default PropertyType.CHILD;
}
```

Mapping The following mapping constraints must be enforced:

- The @Property.name( ) may be specified to the local name (e.g. attribute name or child element name) of simple content to which the property is mapped. If @Property.name( ) is “##default”, the current property is the value of XML content model mapped to the parent entity class.
- The @Property.namespace( ) is used to specify the namespace of a simple content. If @Property.namespace ( ) is “##default”, the target namespace of this property is same as the @Entity.namespace ().
- The @Property.type ( ) is used to define the relationship between the simple XML content the property mapped to and the XML content model the parent entity class mapped to. The value of @Property.type( ) must be of the PropertyType, a Java enumeration class that defines three enumeration constants: attribute, value and the child. The default value of @Propety.type( ) is the PropetyType.CHILD.

@Component

This annotation is a property-level annotation that is used to map a non-entity property to an XML complex content.

Synopsis

```
@Target(ElementType.FIELD)
@Retention(RetentionPolicy.RUNTIME)
public @interface Component{
String relativeLocationPath () default “##default”;
String name ()“##default”;
String namespace () default “##default”;
Property[] properties () default {};
}
```

Mapping

The following mapping constraints must be enforced:

- The component property of an entity model implementation must be a JAXB binding type.
- The @Component.name() may be specified to the local name of the complex content to which the non-entity property is mapped to.
- The @Component.namespace( ) is used to specify the namespace of a simple content. If @Property.namespace ( ) is “##default”, target namespace of the complex content same as the @Entity.namespace().
- The only other additional GRUS mapping annotations allowed with @Component is the @Property to define the mapping annotations of properties defined within the JAXB binding type.
- An entity model implementation may alternatively use @Component.relativeLocationPath to define the location path relative to the context node the entity is mapped to.

The XPath language provides a common syntax and semantics mainly for addressing parts of an XML document by modelling it as a tree of nodes, while HQL is a full object-oriented query language. There are fundamental differences between these two languages. First of all, the XPath language defined a group axis names that allows flexible traversal over a tree of XML nodes. These axis names can be divided into two groups, the forward and reverse axes. Forward axes are used to traverse a specific context node to its children, descendants, and siblings, and reverse axes allows traversal from a specific context node to its parents and ancestors. However HQL takes the similar grammar as SQL, and only supports querying properties defined within a specified entity model class. In another word, a HQL only supports successive XPath axes, such as child and attribute. Besides, XPath also defines a set of functional call expressions, including node-set functions, string functions, boolean functions and number functions, most of which are not supported in HQL language. Finally there are no equivalent operators in HQL to some of those defined in XPath, in particular node-set operators and numeric operators. The supported features of XPath implemented in XPath2Hql filter are summarised in the table below (Table 5-3).

Table 5-3: Features of the XPath language supported in GRUS XPath2Hql filter

XPath Feature	Items
Axes	attribute, child
Abbreviated Axes	@
Relational Operators	= >=, <=, >, <, !=
Boolean Operators	AND, OR, |
Node-set Function	text ( )

Therefore, the XPath2Hql facility enforces a set of restrictions on standard XPath expression. These restrictions are rendered as following formulas:

```
[1]   Expr  :=  OrExpr

[2]   OrExpr   ::=  AndExpr | OrExpr 'or' AndExpr

[3]   AndExpr  ::=  EqualityExpr
                    | AndExpr 'and' EqualityExpr

[4]   EqualityExpr   ::=   NonAdditiveRelationalExpr
                           | EqualityExpr '=' NonAdditiveRelationalExpr
                           | EqualityExpr '!=' NonAdditiveRelationalExpr

[5]   NonAdditiveRelationalExpr  ::=  UnionExpr
                           | NonAdditiveRelationalExpr '>' UnionExpr
                           | NonAdditiveRelationalExpr '<' UnionExpr
                           | NonAdditiveRelationalExpr '>=' UnionExpr
                           | NonAdditiveRelationalExpr '<=' UnionExpr
```

```
[6]   UnionExpr ::= Location
                    | UnionExpr '|' LocationPath

[7]   LocationPath ::= RelativeLocationPath
                    | AbsoluteLocationPath

[8]   AbsoluteLocationPath ::=  '/' RelativeLocationPath?
                            | AbbreviatedAbsoluteLocationPath

[9]   RelativeLocationPath  ::=  Step
                            | RelativeLocationPath '/' Step
                            | AbbreviatedRelativeLocationPath

[10]  Step  ::=  AxisSpecifier NodeTest Predicate*

[11]  AxisSpecifier  ::=  AxisName '::'  | AbbreviatedAxisSpecifier

[12]  AxisName ::= 'attribute'  | 'child'

[13]  AbbreviatedAxisSpecifier  ::=  '@'?

[14]  NodeTest  ::=  NameTest  | NodeType '(' ')'

[15]  NameTest ::= QName

[16]  NodeType ::= 'text'  | 'node'

[17]  Predicate  ::=  '[' PredicateExpr ']'

[18]  PredicateExpr  ::=  Expr

[19]  AbbreviatedAbsoluteLocationPath  ::=  '//' RelativeLocationPath

[20]  AbbreviatedRelativeLocationPath  ::=  RelativeLocationPath '/' Step
```

In order to demonstrate how the XPath2Hql filter works, the following gives an example usage of the XPath2Hql facility that enables XPath query upon relational usage data repository. As discussed in section 5.4.1, the GRUS EMC provides an utility tool that generates a number of abstract entity and DAO artefacts. Implementations of some of generated artefacts with GRUS annotations are given to establish the mapping rules between XPath and HQL query languages.

Example: UsageRecordEntity implementation with GRUS annotation

```
@Entity (name="UsageRecord",
         namespace="http://schema.ogf.org/urf/2003/09/urf",
         isRoot=true)
public class UsageRecordEntity <UserEntity, HostEntity> extends
Abstract UsageRecordEntity<UserEntity, HostEntity>{

  @Component(name="status",
      Properties{
        @Property (name="description", type=PropertyType.ATTR),
        @Property (name="value", type=PropertyType.VALUE)})
  Status status

  @EntityRef (type=UserEntity.class)
  UserEntity user;

  @EntityRef (type=HostEntity.class)
  Host host;
```

Example: UserEntity implementation with GRUS annotation

```
@Entity (name="UserIdentity",
         namespace="http://schema.ogf.org/urf/2003/09/urf")
public class UserEntity <UsageRecordEntity> extends Abstract
UserEntity<UsageRecordEntity>{

  @Property (name="GlobalUserName"
             namespace=" http://schema.ogf.org/urf/2003/09/urf"
             type=PropertyType.CHILD)
```

```
    String globalUserName;

    @Property (name="LocalUserId"
               namespace=" http://schema.ogf.org/urf/2003/09/urf"
               type=PropertyType.CHILD)
    String localUserId;

    @Component(relativeLocationPath="ds:KeyInfo/ds:X509Data/ds:X509S
    ubjectName")
    String userDN;
  ...
  }
```

Example: HostEntity implementation with GRUS annotation

```
  @Entity (name="Host",
           namespace="http://schema.ogf.org/urf/2003/09/urf")
  public class HostEntity <UsageRecordEntity> extends Abstract
  HostEntity<UsageRecordEntity>{

    @Property (name="description"
               namespace=" http://schema.ogf.org/urf/2003/09/urf"
               type=PropertyType.ATTR)
    String description;

    @Property (name="primary"
               namespace=" http://schema.ogf.org/urf/2003/09/urf"
               type=PropertyType.ATTR)
    Boolean isPrimary

    @Property (type=PropertyType.VALUE)
    String value;
    ...
  }
```

A client query request must specify the XPath2Hql filter in the RUS:extract request message as the example below. In the GRUS system, each filter has an assigned global unique name. The XPath2Hql filter name is "urn:grus:filter:xpath-to-hql".

Example: RUS::extract request message using XPath2Hql filter

```
<s:Envelope ...>
<s:Header ...>
...
<wsa:Action mustUnderstand=”true”>
http://schemas.ogf.org/rus/2007/09/core/extractUsageRecordReque
st”>
...
</s:Header>
<s:Body ...>
<rus:ExtractUsageRecordsRequest>
<rus:Filter dialect=”urn:grus:filter:xpath-to-hql”>
/urf:UsageRecord[urf:Status=”finished”][urf:UserIdentity/ds:Key
Info/ds:X509Data/ds:X509SubjectName=”xiaoyu
chen”][urf:Host[@primary=true]=’octopussy.brunal.ac.uk’]
</rus:Filter>
</rus:ExtractUsageRecordsRequest>
</s:Body>
</s:Envelolpe>
```

On receiving the request message, the GRUS endpoint is create an XPath2Hql filter instance, which is then render the XPath expression into HQL expression step-by-step:

- An XPath2Hql filter instance firstly normalise an XPath expression as:

```
/urf:UsageRecord[urf:Status=”finished”][urf:UserIdentity/ds:KeyIn
fo/ds:X509Data/ds:X509SubjectName=”xiaoyu
chen”][urf:Host/@primary=true][urf:Host.text()=’octopussy.brunel.
ac.uk’]
```

- The filter then finds the implementation of root element (i.e. urf:UsageRecord) and generates an initial HQL statement:

```
SELECT FROM UsageRecordEntity AS entity
```

- For each XPath predicates, the filter generates conditional clauses and appends them to the genreated HQL statement. In this example, it traverses the GRUS annotations of the “UsageRecordEntity” class and learns mapping rules. If the @EntityRef annotation encountered, it loads the referenced entity class into memory and analyses in-depth mapping rules. Finally the filter generates an HQL expression as:

```
SELECT FROM UsageRecordEntity AS entity
   WHERE entity.status.value='finished' AND
   entity.user.userDN='xiaoyu chen' AND
   entity.host.isPrimary=true AND
   entity.host.value='octopussy.brunel.ac.uk'
```

The XPath2Hql facility of GRUS is implemented based on Simple API for XPath (SAXPath) [182] and Java APIs for XPath engine (Jaxen)[183]. SAXPath is modeled closely on the structure used by Simple API for XML (SAX)[184], and involves two generic interfaces, including a reader that parses an XPath expression and a handler that receives handles parse events received from the reader. Jaxen is an open source XPath engine that leverages various object models, such as Document Object Model (DOM)[185], XML Object Model (XOM)[186], and so on. Jaxen uses SAXPath and provides default SAXPath reader and handler implementation that parse textual XPath expression and build Jaxen expression trees that can walk through different object models. As the components and runtime events outlined in Figure 5.11, to start the XPath2Hql process, an instance of the XPath2Hql factory class that implements the FilterFactory

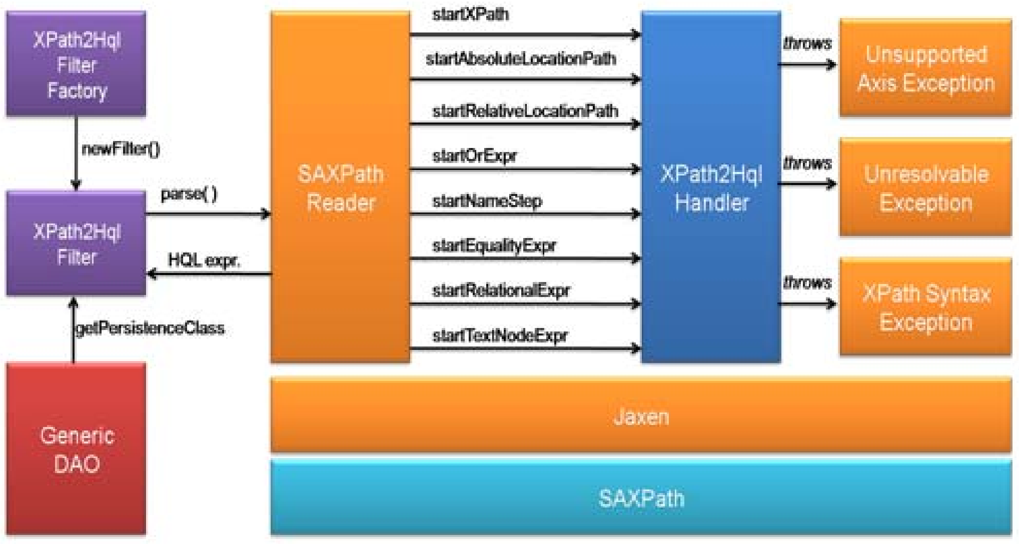


Figure 5.11: The components and runtime events implemented based on Jaxen and SAXPath for rendering an XPath expression to HQL statement.

interface is used to create an instance of XPath2Hql filter that implements the Filter interface. The XPath2Hql filter instance wraps a SAXPath reader object provided by Jaxen. When the filter instance is invoked, the reader parses textual XPath expressions and triggers one of several callback methods implemented within in the XPath2Hql handler. These methods, such as “startXPath”, “startAbsoluteLocationPath”, “startRelativeLocationPath” and so on, are implemented by the default Jaxen handler and are overridden by the XPath2Hql handler to render the Jaxen expression trees into HQL expression trees by processing mapping annotations of entity model classes. During the parsing process, when an unsupported Axis or XPath expression is detected, the XPath2Hql handler stops the parsing process and throws an UnsupportedAxisException and XPathSyntaxException. On successful completion, the XPathHandler instance returns an HQL expression that is used by XPath2Hql filter instance to query usage records through the “findByQueryTerm( )” interface of an GenericDAO object.

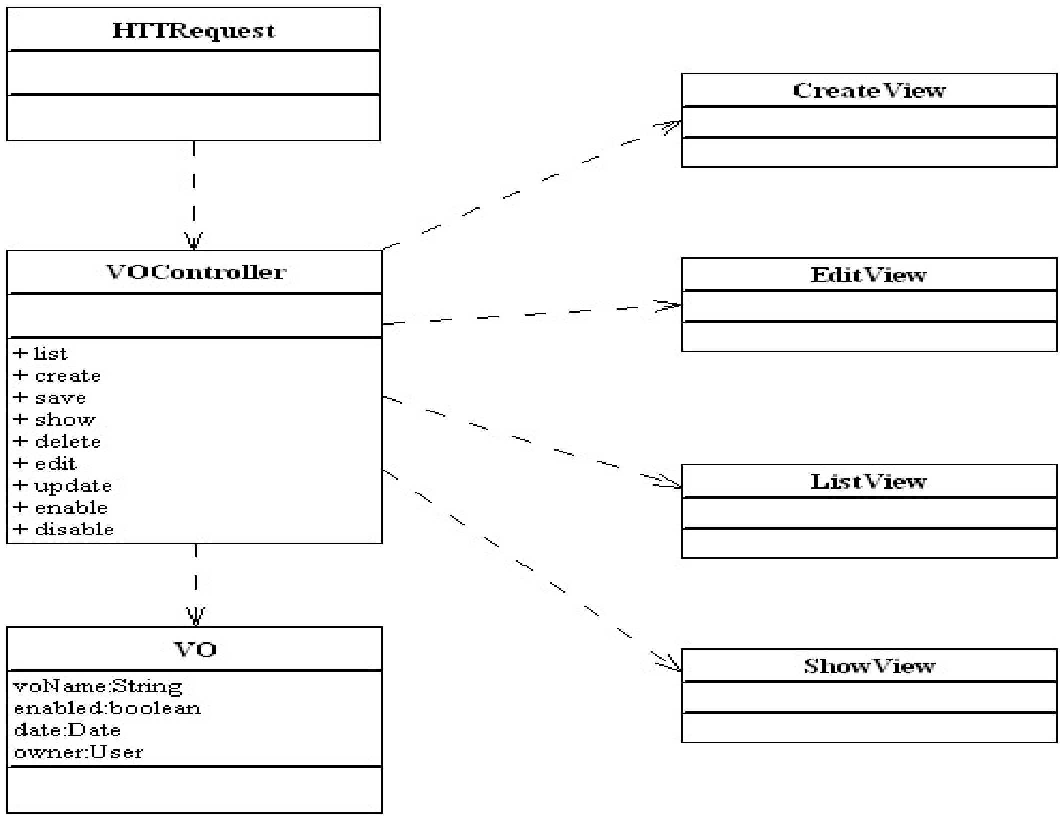


Figure 5.12: Class Diagram of the VO Management Implementation

5.4.4 GRUS Admin

The implementation of GRUS Admin reuses and extends the WLCG-RUS Admin Web application with additional VO management facilities.

VO Management

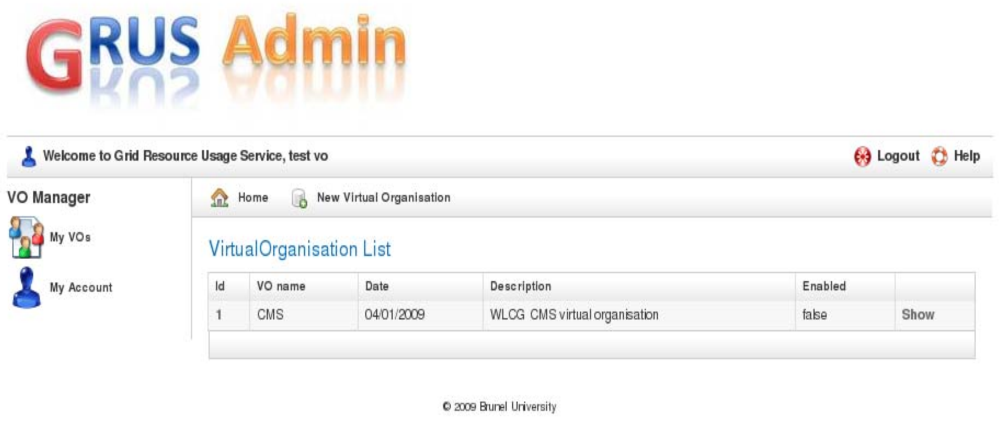
In consistent to other management functionalities implemented in WLCG-RUS Admin, the VO management is implemented based on MVC pattern. As illustrated in Figure 5.12, the implementation of VO management consists of a VO controller, a VO model class, and a set of view pages. An instance of VO model encapsulates meta-information of a VO as a registry entry, including the VO name, owned VO manager, registration date, and status. Similar to Host management facility implemented within WLCG-RUS Admin, the VO controller provided four "do-GET" methods in the host controller, which allows a VO manager to "list", "create", "edit", and "show" managed VO accounts. Each "do-GET" method has also has a view page that provides a Web-based interface to end users. The implementation of VO management defined distinguished authorities for VO managers and the system administrator. For example, a system administrator has full view authorities of all VO accounts registered to a GRUS endpoint, while a VO manager can only view managed VO accounts. Besides, the status of newly created VO account is set to "disabled" and can only be "enabled" by the system administrator.

5.4.5 User Interface

Like the design of WLCG-RUS system, the GRUS provides a command-line interface for interacting RUS service endpoint and a Web-based interface for GRUS system administration.

Command Interface

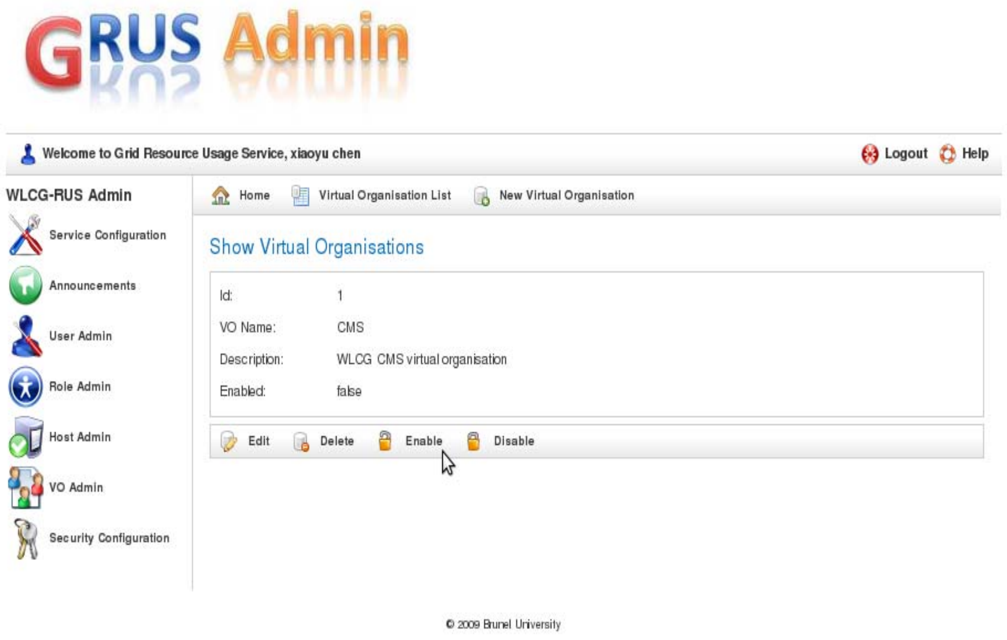
The implementation of GRUS command interface extends WLCG-RUS command-line client and allows a user to view configuration information of a RUS service endpoint, in particular supported aggregate strategies, operation-dialect pairs, and mandatory usage record elements, insert, query, modify, delete and audit usage records through standard



VirtualOrganisation List

Id	VO name	Date	Description	Enabled	
1	CMS	04/01/2009	WLCG CMS virtual organisation	false	Show

Figure 5.13: VO manager view of GRUS Admin Web application



Show Virtual Organisations

Id:	1
VO Name:	CMS
Description:	WLCG CMS virtual organisation
Enabled:	false

Figure 5.14: System administrator view of GRUS Admin Web application

RUS core and GRUS extension interfaces. The Java client is wrapped by a shell script, which accepts arguments as shown in Appendix D.2. At least one of the set of options: list, insert, extract, modify, delete, and audit must be used every time along with the target service endpoint URI. An example command is as shown below to query aggregate strategy list supported by a GRUS endpoint.

```
grus -s <service-endpiont-uri> –list --aggregate-strategies
```

Web Interface

The GRUS Web interface provided enhanced VO management facilities for a VO manager and system administrator. As the screen shot presented in Figure 5.13, a VO manager is redirected to the VO management view where a VO account can be added or removed. By default, a newly created VO account is not enabled until the system administrator approved its validity. Once approved, the VO manager may query usage records belong to owned VOs through a RUS service endpoint. The GRUS system administrator has full view and control of all VO accounts registered. As illustrated in Figure 5.14, the system administrator can edit, delete, create, activate and deactivate a VO account through the administrator view. The presentation of GRUS Admin Web application shares the same layout of WLCG-RUS Admin except replacing WLCG-RUS logo with a new GRUS logo.

5.5 Summary

This chapter presented a middleware solution, the GRUS, which aims at assisting developers in implementing a RUS compatible accounting service. The GRUS design is based on the JISC-proposed accounting framework and consists of four main components: the GRUS EMC, GRUS Core, GRUS annotations, and GRUS Admin Web applications. The EMC is implemented as a utility class that is used to generate abstract entity models and DAO objects. Generated artefacts provide runtime mapping between implementation of entity models and JAXB binding types, and are to be implemented by developers and bounded to custom relational backends through ORM mapping configurations. The GRUS Core provides RUS messaging framework and contains a set of abstract functional components for authorisation, data access, filtering and runtime aggregation. A RUS service endpoint may implement one or more GRUS Core components to provide a custom RUS implementation. GRUS Core also provides implementations of some components, such as the XPath2Hql filter that enables query relational usage data using standard XPath expression, and the dynamic aggregate strategy that is mainly used to summarise query results according to the grouping criteria specified by a user. In order to save development effort, a RUS implementation can reuse functionalities implemented by a helper class, the GrusSupport. GRUS framework defined a set of mapping annotations that are used to customise mapping rules of entity program elements to XML schema constructs. These annotations are embedded within an entity model implementation and fed into XPath2Hql filter that renders standard an XPath expression to HQL statement. The GRUS Admin extended functions implemented by WLCG-RUS Admin with additional VO management facilities allowing a VO manager or system administrator to manage owned or system-wide VO accounts. The GRUS software stacks[1] are hosted at the SourceForge.net as open source software.

[1] http://grus.sourceforge.net/

Chapter 6

Conclusions

This thesis presented systematic researches on Grid accounting including reviews of accounting in the Grid, prototypical development of RUS system in such multi-Grid environment as WLCG, and design and implementation of GRUS middleware. This chapter concludes the research results of this thesis and recommended future works on standards and possible further implementations according to the evolvement of those standards. Relevant publications of this chapter include [188].

6.1 Research Outcomes

This thesis presented a three-year research on design and implementation of Grid accounting systems in multi-Grid environments. The following summarises research outcomes including lessons learned and reflections of research outputs.

6.1.1 Lessons Learned

Throughout researches conducted in this thesis towards developing a standard and interoperable Grid accounting system, there were many problems encountered mainly due to four main factors: lacking of comprehensive understanding of accounting requirements; diverse project-specific requirements; confusion of interoperability and interoperation; tremendous duplicate re-engineering tasks. This section summarises lessons learned during the course of this research.

At the beginning of this research, Grid accounting along with its concept and usage scenarios was new to many. Although there were many definitions and concepts proposed, they were defined based on certain use cases identified in a project-specific manner. Lacking of comprehensive understanding of accounting requirements on heterogeneous usage scenarios across Grid project boundaries was the first and most significant issue in building interoperable accounting systems. Early efforts on developing Grid accounting systems focused on diverse Grid-specific requirements and resulted in the complexity of enabling interoperability between these Grid accounting systems. However the emergence of ever-increasing collaborations requires resource sharing across Grid infrastructures and provisioning a multi-Grid view of resource usage.

In order to cope with the interoperability issue, significant efforts have been put on standardisation. In 2003, the first standard accounting data format was proposed by OGF UR working group, aiming at provisioning common representation usage information at batch job level. The first RUS specification emerged in 2005 and re-designed in 2007 to provide standard interface definitions of a Grid accounting system. These two standards contributed to data and service interoperability between heterogeneous accounting systems. However the adoption of these standards is not as easy as it seems to be. This is because of three main reasons. First of all, the current status of both OGF UR and OGF

RUS specifications are not mature enough to accommodate various accounting usage scenarios. Besides, the evolving nature of these standards slows down the adoption process. Finally and most importantly, the enablement of interoperation is far more than defining interoperable accounting data representation and service interfaces. It also involves a lot of re-engineering tasks, which can break current accounting process enforced by pre-existing accounting systems. This could be better explained by the WLCG accounting process. The current WLCG accounting process involves metering and streaming accounting data from three Grid infrastructures, each of which has custom accounting solution deployed. The interoperation between these accounting systems is enabled by three different communication protocols (section 4.1). The migration to be OGF standard compatible would result in tremendous re-engineering tasks for each pre-existing accounting systems as well as communication protocols and risking existing accounting processes. Therefore such migration becomes a hard decision unless there is an obvious reason and a consistent solution to minimise re-engineering tasks while ensure data consistency.

6.1.2 Reflections

This thesis starts from reviewing current practices by interviewing stakeholders from different groups, including international and national Grid service providers, regional Grid service providers, campus Grid service providers, standard bodies, accounting solution developers, and end users, through face-to-face meetings, Tele-conferences, and questionnaires. During the three-month interview, a list of use cases were identified and categorised into four major usage scenarios (section 3.2). Such review that has not done by others before contributed to a comprehensive view of Grid accounting, including its technical concept, classifications of accounting models, and technical requirements. It also provides systematic reviews of current accounting solutions deployed in production Grid projects as well as standardisation efforts. The review ends up with a proposed accounting framework that abstracts common accounting requirements while customisable to accommodate advanced accounting purposes in a standard compatible manner. In the final review report, a list of prioritised recommendations were proposed to JISC to fund further efforts on standardisation and development tasks for fulfilment of the functionalities of the proposed accounting framework. These recommendations along with the proposed accounting framework were completely accepted by JISC such that following funding calls were released exactly as recommended.

The thesis further described the WLCG-RUS prototype system (section 4), an OGF-RUS implementation based on a loosely-coupled component architecture, which is similar to the proposed accounting framework. The WLCG-RUS prototype system is designed to provide an alternative, but standard compatible, solution for sharing WLCG accounting data across Grid infrastructures to GOC. The deployment of WLCG-RUS system in the WLCG, a production multi-Grid environment, successfully proved the concept that standardisation is of great importance in the interoperability among heterogeneous Grid systems. However the development of WLCG-RUS system also exposed the inefficiency of current standards. The main issue is that both OGF UR and OGF RUS standards were designed for job accounting purpose, therefore does not support the WLCG aggregate accounting models. Besides, there are some common usage properties missing in the OGF UR standard, such as VO name and executing site information, which are important for VO- and site-level accounting.

Based on the experiences gained during the development of WLCG-RUS prototype system, the thesis finally presented the GRUS middleware (section 5), which provides a full implementation of features defined within the proposed accounting framework [132]. Rather than provide a homogeneous accounting solution, the GRUS is intended to provide a development platform for custom implementation of a RUS service endpoint. By using GRUS middleware, existing accounting systems can be easily migrated to be standard compatible with minimum re-engineering efforts, while ensuring back-compatibility to existing accounting processes. Given the evolving nature of accounting standards, the GRUS is designed in a schema-independent manner. In this sense, the GRUS middleware is adaptive to changes of existing accounting schemas as well as emergence of new accounting schemas.

Finally the research work of this thesis also contributed to the evolution of accounting standards. In 2006, the first draft of the Aggregate Usage Record (AUR) presentation specification [158] was submitted to the OGF UR working group. It was then refined according to initial user feedbacks in 2007. At the end of 2007, a new RUS core specification [171] was also proposed based on the implementation of GRUS middleware, making it more flexible to enable various accounting models.

6.2 Recommendations

This section recommends possible future works for both standardisation and development.

6.2.1 Recommendations on Standards

In order to investigate the effectiveness of existing OGF UR [137] standard, the OGF UR working group conducted an evaluation according to user experiences [188] inputs from production Grid projects that uses the OGF UR format for accounting data representation. Based on initial evaluation results, there are some significant issues observed. First of all, the OGF UR format focuses on usage representation of the finest UoW, the “batch” job. Besides, there are still many base properties absent, typically executing site and general VO information. Although these properties can be defined using OGF UR extension framework, they are semantically incorrect. Furthermore using UR extension framework undermines interoperability. Moreover usage metrics defined in OGF UR format 1.0 are not enough to support accounting of resource types other than computational resource, such as data, network and application service resources. Finally most of commercial Grid or cluster systems are using industry accounting data model, for example, the metric sub-model as defined within the DMTF’s Common Information Model (CIM) [86]. It is difficult make these industry standard adopters to use OGF UR to achieve interoperability.

Based on feedbacks received from user experiences, the OGF UR working group defined a new roadmap towards OGF UR 2.0 in OGF 21 conference. As illustrated in Figure 6.1, the OGF UR 2.0 proposed a hierarchical data model with a core information model that abstract common properties including record creator, resource/service consumer, time period and charges information. This core model forms the basis of usage information models of computational, storage and network resource usage records. A composite usage information model is also proposed to representation consumption of a single UoW, which could be a single batch job, a workflow or service transaction. The summary/aggregate usage information model is used to represent total resource usage and costs by summarising multiple composite records. Definitions of various data models in the UR 2.0 roadmap will reuse existing usage metrics and properties defined in current

OGF UR standard where appropriate to ensure backward compatibility.

Although the newly proposed RUS Core specification solved the issues related to system performances and fault tolerance by introducing enumerating query results, the specification cannot be finalised unless other issues are solved. One of the significant issues would be enabling higher flexibility on the RUS service interface definitions so that a RUS service endpoint is able to accept emerging OGF UR 2.0 compatible record instances. Runtime aggregation is another important feature that should be enabled along with RUS insertion or extraction logics. This can be realised either through specific aggregation service interface definitions or normative header information as proposed by GRUS messaging framework. Finally, the usefulness of RUS updating and deletion should be carefully evaluated. If these two service interfaces are not necessary for common use cases, they should be removed from RUS Core specification and defined as an optional or advanced RUS features.

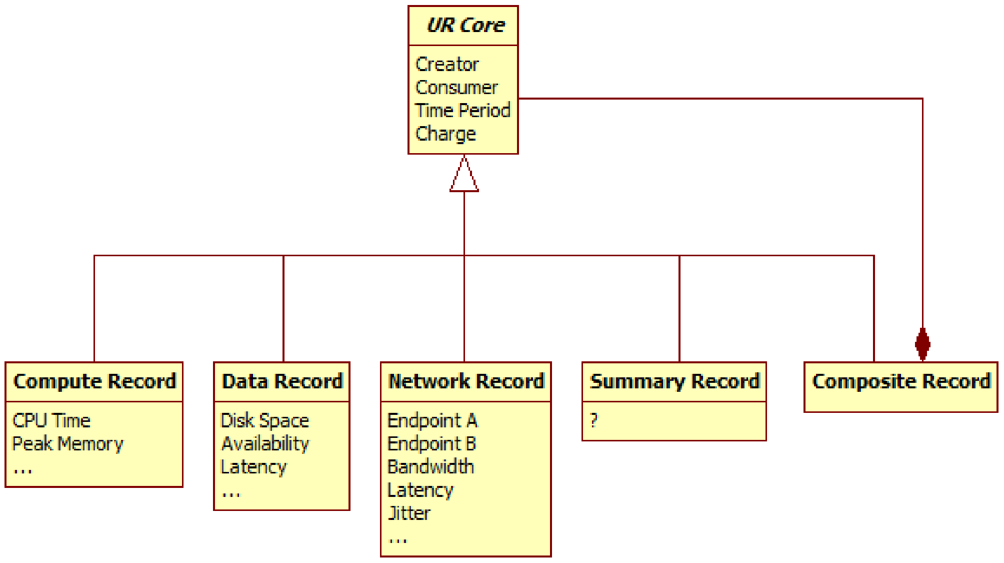


Figure 6.1: The Diagram of UR 2.0 Zoo.

6.2.2 Recommendations on Development

The GRUS framework implemented RUS Core messaging and exhibits an extensible framework for developers to provide custom RUS solutions. With the evolution of RUS

Core specification, the GRUS messaging framework is likely to be changed so as to adapt to possible changes of the RUS Core specification. Besides, the current helper class, the GRUS support, can be used by a RUS implementation for passive accounting models only on relational backend. In the future, possible extensions may be implemented to provide supports for implementing active accounting models as well depending on user feedbacks. Other possible further works that can be done based on GRUS framework include advanced aggregate strategy for OGF UR 2.0 summary record model, filter implementations that support emerging XUpdate Query facility [195], and etc.

Appendix A

Stakeholder Reviews

The following lists the review results of use cases in production Grid projects through interviewing different group of stakeholders.

A.1 National Grid Service

NGS aims to provide computational and data based resources and facilities to UK researchers, independent of resource or researcher location. This is currently achieved using resources (both compute and data) at four core sites (RAL, Oxford, Leeds and Manchester), and a growing number of partner and affiliate sites, together with the provision of software and services, to enable a consistent method of access to any resource from any location. As resources may have different 'owners', each of whom may have different charging policies, it is essential there is a reliable mechanism to account for all aspects of use, in an environment with dynamically varying resources and services.

The NGS already has a sophisticated accounting system in operation and needs to extend the functionality and scope to meet its objectives and address future service requirements. There is a strong desire to use a standard approach maximizing interoperability with other services, and enabling straightforward deployment on sites wishing to partner with the NGS. Major stakeholders to the NGS in the context of accounting and usage monitoring are the grid operations support centre, software developers and standards bodies, current and potential partner and affiliate sites (including campus grids and SRIF funded clusters), funding bodies and end users.

Key requirements

- Performance
- Interoperability – clearly defined APIs or protocols to enable exchange of information with:
    - partner/affiliate sites and dataset providers

    - large scale grid projects such as GridPP/WLCG
- Ease of deployment
- Ability to trace individual jobs; legal requirement for auditability to an individual
- Ability to view historical usage data at user, VO and resource levels
- Metrics:
    - Required – CPU time, Wall time, permanent storage, data services
    - Desirable – executable, memory usage, network usage, QoS
    - Not generally of concern – temporary storage
- Resource allocation and policing
- Custom charge rates for QoS, e.g. advanced reservation
- Integration with user/project management system
- Integrity of accounting data through automated monitoring/notification systems

Key concerns/issues

- RUS querying currently not functional
- Current accounting methodologies and practices are batch job centric
- Interfaces alone should be standardized, allowing site-specific implementation
- Significant investment in current system – would need to see clear benefit in change
- Partner/affiliates not wishing to entrust their data to a centralized site

A.2 Grid for Particle Physics

GridPP is a collaboration of particle physicists and computer scientists from the UK and CERN, with distributed compute resources spanning 17 UK institutions. GridPP has a number of key stakeholders – it is the UK's contribution to worldwide Large Hadron Collider (LHC) Grid (WLCG), overseeing the Tier 1 facility at RAL and the Tier 2 organisations of ScotGrid, NorthGrid, London and SouthGrid, and also contributes to the interdisciplinary project EGEE - Enabling Grids for E-sciencE.

LCG is a production-level grid and GridPP has a contractual obligation to provide accounting data as part of the LCG project. At present over 150 sites worldwide are publishing accounting data to the Grid Operations Centre (GOC) at RAL making aggregation, scalability and validation of accounting data critical concerns.

Key requirements

- Performance and scalability
- Ability to view historical usage data at VO, resource, country and EGEE region levels
- Metrics:
  - Required – CPU (normalized to reflect "work done")/Wall time
  - Desirable – permanent storage
  - Not generally of concern – memory usage, network usage
- Interoperability across international production grids
- Integrity of accounting information through automated monitoring/notification systems
- Ability to modify records e.g. SiteName change does not break historical querying

Key concerns/issues

- Significant investment in current system – 150 sites publishing via APEL/R-GMA
- Scalability of RUS – XML only useful as an exchange format
- CPU normalization and benchmarking needs to be addressed
- Sharing of accounting data across different grids poses difficulties in terms of data protection
- Charging mechanism should be separate and require digital signatures and auditability.

A.3 Campus Grids

The accounting requirements of campus grids across the UK academic sector range from simple "best effort" usage statistics from condor pools to sophisticated job-level accounting across a range of disparate resources. In cases where departmental resources or SRIF-funded hardware are available to the grid there is a more urgent requirement for accounting as a direct consequence of the fEC model (see Other Compute Services, to follow). Less mature campus grids can see immediate benefit from the development of a clearly defined accounting framework and tools to prevent further duplicity of effort.

Key requirements

- Performance
- Interoperability with NGS / other grids but flexibility to allow site-specific access control policies
- Ability to trace individual jobs
- Ability to view historical usage data at user, project, School, and resource levels
- Resource allocation and policing
- Charging mechanisms for fEC (especially HPC component)
- Metrics:
    - Required – CPU time, Wall time, permanent storage
    - Desirable – memory usage, full job command line
    - Not generally of concern – temporary storage, network usage

Key concerns/issues

- Performance of XML database
- Interfaces alone should be standardized, allowing site-specific implementations
- RUS aggregation needs attention
- Wide range of job managers: Linux/Windows Condor, Windows Compute Cluster, PBS, TORQUE, LSF
- Solution should be lightweight and not be tied to a specific project

A.4 Regional Grids

Most regional grids currently operate fairly homogeneous systems at different sites and thus can provide the service with a limited range of software such as a single batch system, and therefore do not, as yet, require the same degree of flexibility as NGS or some campus grids.

Key requirements

- Contractual obligations to provide accounting data to specific large scale projects, e.g. GridPP
- Interoperability with campus grids
- Ability to trace individual jobs

- Ability to view historical usage data at user, project/VO, University, and resource levels
- Resource allocation and policing
- Devolution of allocation management to PIs
- Charging mechanisms required in the future
- Required metrics:
    - Required – CPU time, Wall time
    - Desirable – permanent storage
    - Not generally of concern – temporary storage, network usage

Key concerns/issues

- Data protection

A.5 Other Compute Services

There is an increasing number of universities providing or starting to provide large scale local compute services, particularly after the recent SRIF funding programmes. In many cases this has resulted in a 'standalone' service, typically for local high performance computing (HPC), even at sites where there is or has been campus grid activity, such as Oxford, Cambridge and UCL. Many such services are influenced by fEC and thus need to manage and report on usage. While it may be relatively simple for such services to use resource management or batch engine software to address the accounting requirements, it may be at the cost of interoperability or extensibility for future services. Nevertheless some such services are developing their own accounting and user management systems not tied to a specific supplier, thus providing greater long-term flexibility, but also requiring significant development effort. Thus the objectives for the grid communities, in providing a standard approach for usage data metering, storage and sharing, could be of great value to these other specialist services.

It is recognised that where significant effort has already been invested and the service requirements fully met, such as the national HPC services, there is unlikely to be a good reason for changing existing practices in the short-term. However it would be hoped that such services would see the long-term benefit of a co-ordinated approach, ideally resulting in convergence in development. It is known for example that the developers of the ‘SAFE’ system used by the national HPC services, are developing a generator for converting SAFE-specific usage information into OGF-UR records, and are in the process of implementing a RUS service. It should be made clear that the UR format is not useful only for grid environments – it is a standard format for storing job usage information, which may be used for accounting on any system.

Key requirements

- Job tracing
- Historical usage monitoring at project and user levels
- Management of project resources (sub-allocation)
- Automated policing
- Integration with user management system
- Accuracy of accounting data critical – charging
- Auditability

Key concerns/issues

- Independent contractual arrangements regarding data protection
- Significant investment in current accounting system(s)

A.6 End user

Key requirements

- Intuitive interface, preferably integrated with user management interface
- Job tracing
- Historical usage monitoring at VO and user levels
- Management of project/VO resources (sub-allocation)
- Confidence in the accuracy of accounting data – critical if being charged

A.7 Standard Bodies

There appears to be general support in the grid communities for the OGF-UR and RUS specifications as standards for storing and sharing usage information. OMII-Europe, who is concerned with interoperability between different Grid systems through the implementation of common standard interfaces, are evaluating the implementation of RUS interfaces for the gLite, Globus and UNICORE middleware stacks. To this end, preliminary design documents have been prepared for the SGAS, DGAS and UNICORE accounting systems.

Key requirements

- Acceptance and rollout of OGF usage record format
- Support for development and adoption of aggregated usage record format
- Support for development and adoption of storage usage record format
- Support for development and adoption of network usage record format
- Understanding of more complex use cases
- Hierarchical and P2P RUS deployments

Key concerns/issues

- Site implementations not strictly standards compliant
- Standards not flexible enough to cater for individual accounting requirements
- Standards too bloated for individual requirements
- Issues regarding RUS specification querying interface
- Is Xpath querying expressive enough?

A.8 Data Service Providers

There are a large number of data based services funded by JISC, including the MIMAS and EDINA services. There is an increasing interest in the ‘grid enablement’ of these services, which includes the management of security through grid mechanisms; the ability to combine and analyse data in distributed datasets; and the ability to access grid based (compute) resources dynamically at periods of high loads. There have been a number of grid enablement pilot projects including Gemeda, GEMS (1 and 2), GESSE and SEE-GEO but there are few if any production grid based data services. Authentication and authorisation are key issues in this context – the services currently use ATHENS or Shibboleth, rather than grid certificate authentication.

Most of the data services are required to provide accounting details to JISC on a regular basis as defined by SLAs. The statistics reported are primarily concerned with the number of accesses and searches, on a per site basis, as well as service availability. In addition the service providers need to ensure that accesses are restricted to licensed users (whether individual or site based), so the ability to identify the user of the service is crucial.

Thus most of the data based services are required to provide service usage accounting, rather than resource usage accounting typically required by grid (and other compute based) services; However there are some specialist services, such as the satellite image service, which do have significant resource usage requirements.

While the NGS, for example, does see a long term need for service usage accounting, recognising that such services may be provided through NGS itself, even though the data is hosted elsewhere, there is little in the present standards framework to address this type of accounting. It is not clear to the reviewers how best this should be addressed. It should in principle be possible to define such metrics, but whether it is appropriate or desirable to extend the UR specification, for example, for this purpose is certainly questionable: the UR has been designed with resource based accounting in mind, not service accounting.

In addition, it is clear that many of the current services are well established, and the mechanisms used for collecting the statistics frequently closely integrated with the service itself. The adoption of a new approach for the collection of the statistics across a range of services would probably not be considered favourably. Thus, the reviewers believe it is outside the scope of this review to provide tangible recommendations in this context, although it is felt that such issues should be addressed through further exploratory projects in setting up ‘grid enabled’ services, and subsequently establishing new grid based services as required, rather than adapting accounting mechanisms in existing services.

With respect to some services such as the satellite image service, very large amounts of data must be stored, analysed, and possibly downloaded, and JISC may request information on resource usage to demonstrate a requirement of the service, in order to justify funding streams. The focus is on service access to justify the provision of the service. It is likely that there will be an increase in resource usage associated with these and other data based services, particularly when utilising multiple distributed datasets – something that has not easily been possible previously. This is likely to result in additional accounting requirements, although it does depend (at least partly) on the funding bodies - for example on whether JISC continues to focus on service usage accounting, with little direct interest in details of compute, storage and network usage. However if the service is grid based, with significant storage, network traffic, and high compute requirements possibly at hosts determined dynamically, the owner of the resources will need to be able to charge for use of these resources. Thus it seems essential in the long-term that a mechanism is developed to account for all of these activities. The approaches adopted in the grid accounting context should be applicable to these types of services, bearing in mind the work and time still required to address usage of resources involving storage and network activities.

Key requirements

- Data security, authentication/authorization
- Accounting in workflows: single access/instance may involve multiple services
- Metrics:
    - Required: number of logins, searches, amount of data downloaded, nature of data downloaded
    - Desirable: permanent storage (resource provider end) and network usage if significant downloads are performed
    - Not generally of concern : temporary storage

Key concerns/issues

- Charging model does not fit easily in job-level accounting schema
- Distribution of datasets presents difficulties with respect to licensing
- Grid enablement still in its infancy

Appendix B

Accounting Schema Mapping and Extensions

Table A-1: NGS UAS Accounting Schema mapping to OGF-UR

OGF UR	NGS UAS Schema	
Metric Context Node (XML)	Metric Name	Base Data Type (SQL)
//urf:RecordIdentity@urf:recordId	RecordId	VARCHAR
//urf:JobIdentity/urf:LocalJobId	LocalJobID	VARCHAR
//urf:UserIdentity/urf:LocalUserId	LocalUserId	VARCHAR
//urf:UserIdentity/ds:KeyInfo/ds:X509Data/ ds:X509SubjectName	X509SubjectName	VARCHAR
//urf:JobName	JobName	VARCHAR
//urf:Status	Status	VARCHAR
//urf:WallDuration	WallDuration	NUMBER
	wallTimeRequested	NUMBER
//urf:CpuDuration	CpuDuration	NUMBER
	cpuTimeRequested	NUMBER
//urf:EndTime	pbsLogDate	DATE
//urf:StartTime	timeGlobusSubmitted	DATE
//urf:MachineName	MachineName	VARCHAR
//urf:SubmitHost	SubmitHost	VARCHAR
//urf:Processors	Processors	NUMBER

Table A-2: APEL Accounting Schema mapping to OGF-UR

OGF UR Schema	APELSchema	
Metric Context Node (XML)	Metric Name	Base Data Type (SQL)
//urf:RecordIdentity@urf:recordId	RecordIdentity	VARCHAR
//urf:RecordIdentity@createTime	MeasurementDate	DATE
	MeasurementTime	TIME
//urf:JobIdentity/urf:GlobalJobId	LCGJobID	VARCHAR
//urf:JobIdentity/urf:LocalJobId	LocalJobID	VARCHAR
//urf:UserIdentity/urf:LocalUserId	LocalUserId	VARCHAR
//urf:Useridentity/GlobalUserName	LCGUserID	VARCHAR
//urf:WallDuration	ElapsedTime	VARCHAR
	ElapsedTimeSeconds	INT
//urf:CpuDuration	BaseCpuTime	VARCHAR
	BaseCpuTimeSeconds	INT
//urf:EndTime	StopTime	VARCHAR
	StopTimeUTC	VARCHAR
	StopTimeEpoch	INT
//urf:StartTime	StartTime	VARCHAR
	StartTimeUTC	VARCHAR
	StartTimeEpoach	INT
//urf:Host	ExecutingCE	VARCHAR
//urf:Memory	MemoryReal	INT
	MemoryVirtual	INT
//urf:TimeInstant	EventDate	DATE
	EventTime	TIME

Table A- 3: DGAS Accounting Schema mapping to OGF-UR

OGF UR Schema	DGAS Schema	
Context Node (XML)	Metric Name	Base Data Type (SQL)
//urf:RecordIdentity@urf:recordId	id	BIGINT
//urf:RecordIdentity@createTime	date	DATETIME
//urf:Charge	amount	SMALLINT
//urf:JobIdentity/urf:GlobalJobId	LCGJobID	VARCHAR
//urf:JobIdentity/urf:LocalJobId	lrmsId	VARCHAR
//urf:UserIdentity/ds:KeyInfo/ds:X509Data/ ds:X509SubjectName	acl	VARCHAR
//urf:UserIdentity/urf:GlobalUserName	gridUser	VARCHAR
//urf:UserIdentity/urf:LocalUserId	localUserId	VARCHAR
//urf:WallDuration	wallTime	INT
//urf:CpuDuration	cpuTime	INT
//urf:EndTime	end	INT
//urf:StartTime	start	INT
//urf:MachineName	gridResource	VARCHAR
//urf:Memory	pmem	INT
	vmem	INT

Table A-4: Gratia Accounting Schema mapping to OGF-UR

OGF UR Schema	Gratia Schema	
Metric Context Node (XML)	Metric Name	Base Data Type (SQL)
//urf:RecordIdentity@urf:recordId	recordId	BIGINT
//urf:RecordIdentity@createTime	CreateTime	DATETIME
	CreateTimeDescription	VARCHAR
//urf:JobIdentity/urf:GlobalJobId	GlobalJobId	VARCHAR
//urf:JobIdentity/urf:LocalJobId	LocalJobID	VARCHAR
//urf:Jobidentity/urf:ProcessId	ProcessIds	VARCHAR
//urf:JobName	JobName	VARCHAR
//urf:JobName@urf:description	JobNameDescription	VARCHAR
//urf:UserIdentity/urf:LocalUserId	LocalUserId	VARCHAR
//urf:Useridentity/GlobalUserName	GlobalUserName	VARCHAR
//urf:UserIdentity/ds:KeyInfo@ds:id	KeyInfoId	VARCHAR
//urf:UserIdentity/ds:KeyInfo	KeyInfoContent	BLOG
//urf:Charge	Charge	FLOAT
//urf:Charge@urf:unit	ChargeUnit	VARCHAR
//urf:Charge@urf:formula	ChargeFormula	VARCHAR
//urf:Charge@urf:description	ChargeDescription	VARCHAR
//urf:Status	Status	VARCHAR
//urf:Status@urf:description	StatusDescription	VARCHAR
//urf:WallDuration	WallDuration	VARCHAR
//urf:WallDuration@urf:description	WallDurationDescription	VARCHAR
//urf:CpuDuration	CpuUserDuration	VARCHAR
	CpuSystemDuration	VARCHAR
//urf:CpuDuration@urf:description	CpuUserDurationDescription	VARCHAR
	CpuSystemDurationDescription	VARCHAR

//urf:NodeCount	NodeCount	VARCHAR
//urf:NodeCount@urf:metric	NodeCountMetric	VARCHAR
//urf:NodeCount@urf:description	NodeCountDescription	VARCHAR
//urf:Processors	Processors	INT
//urf:Processors@urf:metric	ProcessorsMetric	VARCHAR
//urf:Processors@urf:consumptionRate	ProcessorsConsumptionRate	FLOAT
//urf:Processors@urf:description	ProcesorsDescription	VARCHAR
//urf:StartTime	StartTime	DATETIME
//urf:StartTime@urf:description	StartTimeDescription	VARCHAR
//urf:EndTime	EndTime	DATETIME
//urf:EndTime@urf:description	EndTimeDescription	VARCHAR
//urf:MachineName	MachineName	VARCHAR
//urf:MachineName@urf:description	MachienNameDescription	VARCHAR
//urf:SubmitHost	SubmitHost	VARCHAR
//urf:SubmitHost@urf:description	SubmitHostDescription	VARCHAR
//urf:Queue	Queue	VARCHAR
//urf:Queue@urf:description	QueueDescription	VARCHAR
//urf:Host	Host	VARCHAR
//urf:Host@urf:description	HostDescription	VARCHAR

Table A-5: Custom Metrics as Extensions to OGF-UR

OGF-UR Extension Framework	**APEL Extension**	**DGAS Extension**	**Gratia Extension**	**SGAS Extension**	**UNICORE Accounting Extensions**	**Description**
//urf:Resource	LCGUserVO	UserVO	VOName	VOName	VOName	Virtual organisation identity
			Reportable VOName			VO Name that is actually when reporting the usage records
			ProbeName			The probe identity that meters resource usage
	ExecutingSite	SiteName	SiteName	SiteName	SiteName	The site name on which the job recorded is executed
		iBenchType				(Integer) performance benchmark specification type
	SpecInt2000	iBench				The GLUE host benchmark (SI00)
		fBenchType				(float) performance benchmark specification type
	SpecFloat2000	fBench				The GLUE host benchmark (SF00)
		userGroup				The user group name
	UserFQAN	userFQAN				Full Qualified Attribute Name
		localGroup				Local group name
		remoteHlr				Home Local Resource server URL

Table A-6: WLCG summary schema mapping to proposed OGF-AUR draft

OGF AUR Schema	WLCG Summary Schema	
Metric Context Node (XML)	Metric Name	Base Data Type (SQL)
//urf:RecordIdentity@urf:recordId	recordId	BIGINT
//urf:RecordIdentity@createTime	CreateTime	DATETIME
	CreateTimeDescription	VARCHAR
//urf:JobIdentity/urf:GlobalJobId	GlobalJobId	VARCHAR
//urf:JobIdentity/urf:LocalJobId	LocalJobID	VARCHAR
//urf:Jobidentity/urf:ProcessId	ProcessIds	VARCHAR
//urf:JobName	JobName	VARCHAR
//urf:JobName@urf:description	JobNameDescription	VARCHAR
//urf:UserIdentity/urf:LocalUserId	LocalUserId	VARCHAR
//urf:Useridentity/GlobalUserName	GlobalUserName	VARCHAR
//urf:UserIdentity/ds:KeyInfo@ds:id	KeyInfoId	VARCHAR
//urf:UserIdentity/ds:KeyInfo	KeyInfoContent	BLOG
//urf:Charge	Charge	FLOAT
//urf:Charge@urf:unit	ChargeUnit	VARCHAR
//urf:Charge@urf:formula	ChargeFormula	VARCHAR
//urf:Charge@urf:description	ChargeDescription	VARCHAR
//urf:Status	Status	VARCHAR
//urf:Status@urf:description	StatusDescription	VARCHAR
//urf:WallDuration	WallDuration	VARCHAR
//urf:WallDuration@urf:description	WallDurationDescription	VARCHAR
//urf:CpuDuration	CpuUserDuration	VARCHAR
	CpuSystemDuration	VARCHAR
//urf:CpuDuration@urf:description	CpuUserDurationDescription	VARCHAR
	CpuSystemDurationDescription	VARCHAR

Appendix C

Use Cases

B.1 WLCG-RUS Use Cases

Use Case	Insert usage records
Description	Publish resource usage information to WLCG RUS through standard RUS::insertUsageRecords interface.
Actors	Host
Assumptions	• Requestor holds a valid grid certificate; • Accounting data to be uploaded are correct and trustworthy;
Steps	1. check host's permission to execute "RUS::insertUsageRecord" operation on per usage record basis; 2. validate usage record inputs against mandatory elements configuration; 3. render standard usage record format to appropriate data format; 4. save usage records into database; 5. compose response message with operation results;
Variations	5. if trying to insert job usage records into summary record database, appropriate aggregate strategy must be applied
Non-Functional	Security: authorisation and data privacy Performance: usage records should be inserted in bulk if possible.
Issues	1. Trying to insert usage records that already exist;

Use Case	List mandatory usage record elements
Description	Query mandatory element configuration of a specific WLCG RUS instance.
Actors	Host, Administrator, Site Manager, Grid User, VO Manager
Assumptions	1. Requestor holds a valid grid certificate;
Steps	1. Find out mandatory usage record element configuration; 2. Compose RUS::listMandatoryUsageRecordElements response message;
Variations	
Non-Functional	
Issues	Mandatory usage record element configuration infoset not found

Use Case	Create a host account
Description	Register a new host account
Actors	Site Manager, Administrator
Assumptions	Requestor hold a valid grid certificate;
Steps	1. Check user’s permission to register; 2. Check validity of requestor’s account; 3. Create a new host account 4. Email requestor a confirmation message
Variations	
Non-Functional	Security: only registered user with an active account is allowed to create a new host account
Issues	1. The registry entry of host account already exists;

Use Case	Delete a host account
Description	Remove a host account from registry entry
Actors	Site Manager, Administrator
Assumptions	Requestor hold a valid grid certificate;
Steps	1. Check user’s permission to register; 2. Check validity of requestor’s account; 3. Find out host account on requestor’s account; 4. Remove the host account from registry;
Variations	
Non-Functional	Security: only registered user and the owner of an active host account is allowed to remove a new host account
Issues	Trying to delete an host account that is publishing data

Use Case	View host account information
Description	View registration details of host accounts
Actors	Site Manager, Administrator
Assumptions	Requestor hold a valid grid certificate;
Steps	1. Check user’s permission to register; 2. Check validity of requestor’s account; 3. Find out host account on requestor’s account; 4. Display host account details;
Variations	
Non-Functional	Security: administrator can view all host account details, while site manger can only view owned host account details;
Issues	The registry entry of specific host account does not exist.

Use Case	Activate a host account
Description	Activate a host account
Actors	Administrator
Assumptions	Requestor hold a valid grid certificate;
Steps	1. Check user’s permission ; 2. Find host account on requestor’s account; 3. Activate the host account; 4. Email host owner an activation message;
Variations	
Non-Functional	Performance: activation should be completed in reasonable short period.
Issues	The registry entry of specific host account does not exist.

Use Case	Edit host account
Description	Edit host account’s details
Actors	Site manager, Administrator
Assumptions	Requestor hold a valid grid certificate;
Steps	1. Check requestor’s permission to edit a host account; 2. Edit host account; 3. Set activeness of current host account to false; 4. Email host owner a confirmation message
Variations	
Non-Functional	Security: Administrator can edit all user account details while account owner can edit its own user account details;
Issues	

Use Case	User account registration
Description	Register a new user account
Actors	Site manager
Assumptions	Requestor hold a valid grid certificate;
Steps	1. Check requestor’s permission to execute user registration; 2. Create a new user account 3. Email registered user confirmation message
Variations	
Non-Functional	Performance: user registration should be completed in reasonable short period.
Issues	1. A registry entry of user account already exists;

Use Case	Delete a user account
Description	Remove a user account from registry
Actors	Administrator
Assumptions	Requestor hold a valid grid certificate;
Steps	1. Check user’s permission; 2. Find user account; 3. Remove the user account from registry; 4. Email deleted user;
Variations	
Non-Functional	Security: only administrator is allowed to remove a new user account
Issues	Trying to delete a non-existent user account

Use Case	View user account information
Description	View detailed user account information
Actors	Site Manager, Administrator
Assumptions	Requestor hold a valid grid certificate;
Steps	1. Check user's permission to view user account(s) 2. Find user account; 3. Display user account details;
Variations	
Non-Functional	Security: Administrator can view all user accounts' details, while site manager can only view its own account details.
Issues	The registry entry of specific user account does not exist.

Use Case	Activate a user account
Description	Activate a host account
Actors	Administrator
Assumptions	Requestor hold a valid grid certificate;
Steps	1. Check user's permission ; 2. Find user account; 3. Activate the user account; 4. Email account owner an activation message;
Variations	
Non-Functional	Performance: activation should be completed in reasonable short period.
Issues	The registry entry of specific user account does not exist.

Use Case	Edit a user account
Description	Update a user account's details
Actors	Administrator, Site Manager
Assumptions	Requestor hold a valid grid certificate;
Steps	1. Check user's permission ; 2. Find user account; 3. Update user account details; 4. Email account owner a confirmation message;
Variations	
Non-Functional	Security: administrator can edit any user accounts, while site manager can edit its own account only.
Issues	The registry entry of specific user account does not exist.

B.2 GRUS Use Cases

Use Case	List supported aggregate strategies
Description	Query supported aggregate strategies of a RUS service endpoint
Actors	Administrator, Site manage, VO manager, Grid User
Assumptions	1. Requestor holds a valid grid certificate;
Steps	1. Find out supported aggregate strategies configuration; 2. Compose a response message and return to client;
Variations	
Non-Functional	
Issues	Supported dialects configuration not found

Use Case	Query job usage records
Description	Query OGF UR instances through the RUS::extractUsageRecord interface of a RUS service endpoint
Actors	Administrator, Site manage, VO manager, Grid User
Assumptions	1. Requestor holds a valid grid certificate;
Steps	1. Check specified query dialect against supported dialects of a RUS service endpoint; 2. Get query results that match query term from underlying persistent storage; 3. Rendering query results into OGF UR instances; 4. Check user permission on individual OGF UR instance; 5. Compose a response message and send it back to client;
Variations	5. Compose a response message and returns a context for enumeration
Non-Functional	Security: authorisation and data privacy Performance: enumerating query results if the value of maximum elements is specified within the request message.
Issues	Supported dialects configuration not found

Use Case	Query aggregate usage records
Description	Query OGF AUR instances through the RUS::extractUsageRecord interface of a RUS service endpoint
Actors	Administrator, Site manage, VO manager, Grid User
Assumptions	1. Requestor holds a valid grid certificate;
Steps	1. Check specified query dialect against supported dialects of a RUS service endpoint; 2. Get query results that match query term from underlying persistent storage; 3. if underlying accounting data type is job usage records, apply aggregate strategy specified in the request message; 4. Rendering aggregate results into OGF AUR instances; 5. Check user permission on individual OGF AUR instance; 6. Compose a response message and send it back to client;
Variations	6. Compose a response message and returns a context for enumeration
Non-Functional	Security: authorisation and data privacy Performance: enumerating query results if the value of maximum elements is specified within the request message.
Issues	Supported dialects configuration not found

Use Case	Audit
Description	Query history of a specific usage record
Actors	Host, Administrator, Site Manager, VO manager
Assumptions	1. Requestor holds a valid grid certificate;
Steps	1. Get usage record identified by record identity specified in the request message; 2. Check user’s permission on the usage record; 3. Get record history associate with the usage record; 4. Compose response message and return it to the client;
Variations	
Non-Functional	Security: authorisation and data privacy
Issues	The requested usage record does not exist.

Use Case	create a VO account
Description	Add a new VO account
Actors	Host Manager, Administrator
Assumptions	Requestor hold a valid grid certificate;
Steps	1. Check user’s permission; 2. Check mandatory VO account information; 3. Create a new VO account; 4. Email client a confirmation message
Variations	
Non-Functional	Security: only registered user with an active account is allowed to create a new VO account
Issues	1. The VO account already exists;

Use Case	View VO account
Description	View account information of a specific created VO
Actors	VO Manager, Administrator
Assumptions	Requestor hold a valid grid certificate;
Steps	1. Check user’s permission; 2. Display VO account details on screen;
Variations	
Non-Functional	Security: administrator can view all host account details, while a VO manger can only view an owned VO account;
Issues	The specific VO account does not exist.

Use Case	Edit VO account(s)
Description	Edit a VO account information
Actors	VO manager, Administrator
Assumptions	Requestor hold a valid grid certificate;
Steps	1. Check requestor’s permission to edit a VO account; 2. Edit VO account details; 3. Set activeness of current VO account to false; 4. Email VO owner a confirmation message
Variations	Security: Administrator can edit all VO account details while account owner can edit its own user account details;
Non-Functional	
Issues	VO manager is not allowed to change owner;

Use Case	Activate a VO account
Description	Query mandatory element configuration of a specific WLCG RUS instance.
Actors	Administrator
Assumptions	Client hold a valid grid certificate;
Steps	5. Check user's permission; 6. Find VO account; 7. Activate the VO account; 8. Email VO owner an activation message;
Variations	
Non-Functional	Performance: activation should be completed in reasonable short period.
Issues	The VO account does not exist.

Use Case	Delete a VO account
Description	Query mandatory element configuration of a specific WLCG RUS instance.
Actors	VO manager, Administrator
Assumptions	Requestor hold a valid grid certificate;
Steps	5. Check user's permission; 6. Find VO account; 7. Remove the VO account; 8. Email VO account owner;
Variations	
Non-Functional	Security: VO manager can only remove a owned VO account
Issues	Trying to delete a non-existent user account

Appendix D

Command Line Parameters

C.1 WLCG-RUS Command Line Parameters

```
July 2007(User Commands)                  July 2007(User Commands)

NAME
       wlcgrus - manual page

DESCRIPTION
       usage: wlcgrus [-h<help> | -list | -insert]

              [-s<service-endpiont>]

       WLCG-RUS version 0.1 CLI, copyright 2007 Brunel.

       -h,--help
              print usage information

       -insert,--insert
              insert usage records

       -list,--list
              list mandatory elements

       --max-elements
              The maximum number of usage records per insertion

       -s,--service-endpoint
```

```
              service endpoint address

       usage: wlcgrus [-h<help> | -list | -insert]

              [-s<service-endpiont>]

       WLCG-RUS version 0.1 CLI, copyright 2007 Brunel.
```

SEE ALSO

The full documentation for WLCG RUS is maintained as a Text info manual.

If the info and WLCG RUS programs are properly installed at your site, the command

```
              man wlcgrus
```

should give you access to the complete manual.

July 2007(User Commands)

C.2 GRUS Command Line Parameters

```
June 2009(User Commands)                    June 2009(User Commands)

NAME
       grus - manual page

DESCRIPTION
       usage:  grus  [-h<help>  |  -list  |  -insert  |  -extract  |  -
modify | -delete]

              [-s<service-endpiont>] [-t<timeout>]

       GRUS version 1.0 CLI, copyright 2009 Brunel.

       -audit,--audit
              extract record history

       -delete,--delete
              delete usage records

       -extract,--extract
              extract usage records

       -h,--help
              print usage information

       -insert,--insert
              insert usage records

       -list,--list
              list GRUS configuration information

       -modify,--modify
              modify usage records

       -s,--service_endpoint
              service endpoint address
       -t,--timeout <arg>
              timeout in millisecs

       for more instructions, see http://grus.sourceforge.net

       usage:  grus  [-h<help>  |  -list  |  -insert  |  -extract  |  -
modify | -delete]

              [-s<service-endpiont>] [-t<timeout>]

       GRUS version 1.0 CLI, copyright 2009 Brunel.

       for more instructions, see grus.sourforge.org
```

SEE ALSO

```
       The  full   documentation  for  invalid  is  maintained  as  a
Texinfo manual.
       If the info and invalid programs are properly installed at
your  site,
       the command

              man grus

       should give you access to the complete manual.

                                   September 2009(User Commands)
```

Appendix E

Schemas

D.1 GRUS Data Type Definitions

```
<?xml version="1.0" encoding="utf-8" ?>

<!--*************************************************************
Copyright @ 2007-2009 Brunel University. All rights reserved.
Permission to copy, display, perform, modify and distribute
the GRUS extensions to OGF RUS-Core WS-I rendering specification.
**************************************************************-->

<xsd:schema

targetNamespace="http://schemas.brunel.ac.uk/services/accounting/grus/typ
es"

xmlns:grus="http://schemas.brunel.ac.uk/services/accounting/grus/types"
        xmlns:xacml="urn:oasis:names:tc:xacml:1.0:policy"
        xmlns:urf="http://schema.ogf.org/urf/2003/09/urf"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        attributeFormDefault="qualified" elementFormDefault="qualified">

<xsd:annotation>
<xsd:documentation xml:lang="en">
The data type and elements defined in this schema document provides
header extensions to the RUS::insertUsageRecords and
RUS::extractUsageRecords messages as defined in OGF RUS-Core WS-I
rendering specification. Using headers defined here allows runtime
aggregation during the execution of RUS insertion and extraction
operations.
```

```
</xsd:documentation>
</xsd:annotation>

<xsd:import namespace="http://schema.ogf.org/urf/2003/09/urf"
                        schemaLocation="urf.xsd" />
<xsd:element name="AggregateStrategies">
<xsd:complexType>
<xsd:sequence>
<xsd:element ref="grus:AggregateStrategy" minOccurs="0"
maxOccurs="unbounded" />
</xsd:sequence>
</xsd:complexType>
</xsd:element>

<xsd:element name="AggregateStrategy"
             type="grus:AggregateStrategyType" />

<xsd:complexType name="AggregateStrategyType">
<xsd:sequence>
<xsd:element name=”Interval”>
<xsd:simpleContent>
<xsd:
<xsd:element name="Entity"
             type="grus:EntityType"
             maxOccurs="unbounded"
             minOccurs="0" />
</xsd:sequence>
<xsd:attribute name="AggregateStrategyId"
               type="xsd:anyURI"
               use="optional" />
</xsd:complexType>

<xsd:complexType name="EntityType">
<xsd:simpleContent>
<xsd:extension base="xsd:QName">
<xsd:anyAttribute namespace="##any" processContents="lax"/>
</xsd:extension>
</xsd:simpleContent>
</xsd:complexType>

<xsd:element name="ListSupportedAggregateStrategiesRequest"
        type="grus:ListSupportedAggregateStrategiesRequestType" />
```

```
<xsd:complexType name="ListSupportedAggregateStrategiesRequestType">
<xsd:sequence>
<xsd:any namespace="##other"
         minOccurs="0"
         maxOccurs="unbounded"
         processContents="lax" />
</xsd:sequence>
</xsd:complexType>

<xsd:element name="SupportedAggregateStrategy">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="Operation" type="xsd:anyURI" />
<xsd:element name="AggregateStrategy"
             type="grus:AggregateStrategyType"
             minOccurs="1"
             maxOccurs="unbounded" />
</xsd:sequence>
</xsd:complexType>
</xsd:element>


<xsd:element name="ListSupportedAggregateStrategiesResponse"
       type="grus:ListSupportedAggregateStrategiesResponseType" />

<xsd:complexType name="ListSupportedAggregateStrategiesResponseType">
<xsd:sequence>
<xsd:element ref="grus:SupportedAggregateStrategy"
             minOccurs="0"
             maxOccurs="unbounded"/>
</xsd:sequence>
</xsd:complexType>

</xsd:schema>
```

D.2 GRUS Service Interface Definitions

```
<?xml version="1.0" encoding="UTF-8"?>

<definitions
xmlns:tns="http://schemas.brunel.ac.uk/services/accounting/grus"

xmlns:types="http://schemas.brunel.ac.uk/services/accounting/gru
s/types"

xmlns:wsen="http://schemas.xmlsoap.org/ws/2004/09/enumeration"
        xmlns:wse="http://schemas.xmlsoap.org/ws/2004/08/eventing"

xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
xmlns:rus="http://schemas.ogf.org/rus/2007/09/core/types"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:wsoap12="http://schemas.xmlsoap.org/wsdl/soap12/"

targetNamespace="http://schemas.brunel.ac.uk/services/accounting/g
rus">


<!--*****************************************************************
*               Import third-party WSDL files                       *
*****************************************************************-->

<import
namespace="http://schemas.xmlsoap.org/ws/2004/09/enumeration"
location="enumeration.wsdl" />
<import
namespace="http://schemas.ogf.org/rus/2007/09/core/types"
location="rus-core.wsdl" />
<!-- ****************************************************************
*                           Type definitions                        *
**************************************************************** -->
<types>
<xsd:schema
targetNamespace="http://schemas.brunel.ac.uk/services/accounting/g
rus"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:types="http://schemas.brunel.ac.uk/services/accounting/grus/
types">

<xsd:import
namespace="http://schemas.ogf.org/rus/2007/09/core/types"
schemaLocation="../schemas/rus-core.xsd" />

<xsd:import
namespace="http://schemas.brunel.ac.uk/services/accounting/grus/ty
pes"
schemaLocation="../schemas/grus.xsd" />

<xsd:import
namespace="http://schema.ogf.org/urf/2006/07/aur"
schemaLocation="../schemas/aur.xsd" />
```

```
</xsd:schema>
</types>


<!--************************************************************
*                      Message Definitions                     *
************************************************************-->
<message name="ListSupportedAggregateStrategiesRequestMessage">
<part
name="ListSupportedAggregateStrategiesRequest"
element="types:ListSupportedAggregateStrategiesRequest" />
</message>

<message name="ListSupportedAggregateStrategiesResponseMessage">
<part name="ListSupportedAggregateStrategiesResponse"
element="types:ListSupportedAggregateStrategiesResponse" />
</message>

<!--*************************************************************
*                   Port Type Definitions                       *
*************************************************************-->

<portType name="GridResourceUsageServicePortType">
<operation name="ListSupportedAggregateStrategies">
<input
message="tns:ListSupportedAggregateStrategiesRequestMessage"
wsa:Action="http://schemas.brunel.ac.uk/services/accounting/grus/l
istSupportedAggregateStrategies" />
<output
message="tns:ListSupportedAggregateStrategiesResponseMessage"
wsa:Action="http://schemas.brunel.ac.uk/services/accounting/grus/l
istSupportedAggregateStrategiesResponse" />
</operation>
</definitions>
```

Bibliography

[1] H. H. Goldstine, and A. Goldstine, “The Electronic Numerical Integrator and Computer (ENIAC)”, 1946 (reprinted in *The Origins of Digital Computers: Selected Papers*, Springer-Verlag, New York, 1982, pp. 359-373)

[2] Moore’s Law- Wikipedia, http://en.wikipedia.org/wiki/Moores_law

[3] M. V. Wilkes. “Automatic Digital Computers”, *New York: John Wiley & Sons*. pp. 305 pages. QA76.W5 1956.

[4] N. Beth Stern, “From Eniac to UNIVAC: An Appraisal of the Eckert-Mauchy Computers”, ISBN 0932376142.

[5] R. R. Schaller, “Moore's law: past, present and future”, *Spectrum IEEE*, Vol.34, Jun. 1997, pp.52-59.

[6] T. Aita and Y. Husimi, “Fitness landcape of biopolymers and efficient optimization strategy in evolutionary molecular engineering”, *Proc. of 6th Int. Sympo. on A-life and Robotics*, Vol.6, 2001, pp.365-368.

[7] S. Lee, L. R. Hook, “Towards the Design of a Nanocomputer”, *Proc. of Electrical and Computer Engineering Conference 2006 (CCECE ‘06’),* May 2006, pp.74-77.

[8] S. E. Lysheyski, “Nanotechnology, quantum information theory and quantum computing”, *Proc. of 2nd IEEE Nanotechnology 2002 (IEEE-NANO 2002)*, 26-28 August, 2002, pp.309-314.

[9] C. Joach, “Towards a molecule-computer? Resources and Technologies to compute within a single molecule”, *Proc. of 31th Solid-State Circuits Conference 2005 (ESSCIRC*

2005), France, September 2005, pp.27-28.

[10] Flynn’s taxonomy-Wikipedia, http://en.wikipedia.org/wiki/Flynn’s_taxonomy

[11] M. J. Flynn, “Some Computer Organizations and Their Effectiveness”, *IEEE Trans. on Computers*, Vol. C-21, pp.948-960, 1972.

[12] G. Amdahl, “The validity of the single processor approach to achieving large-scale computing capabilities”, *Proc. of AFIPS Spring Joint Computer Conference*, Atlantic City, N. J. AFIPS Press, pp.483-485.

[13] M. C. August, G. M. Brost, C. C Hsiung and A. J. Schiffleger, “Cray X-MP: the birth of a supercomputer”, *Trans. on IEEE computer*, vol. 22, pp. 45-52, January 1989.

[14] T. Blank, “The MasPar MP-1 architecture”, *Proc. of 35th IEEE Computer Society International Conference*, pp.20-24, March 1990.

[15] S. F. Reddaway, “DAP - a distributed array processor”, *Proc. of the 1st annual symposium on Computer Architecture*, ACM Press New York, Gainesville, Florida, pp 61-65, 1973.

[16] Top 500 Supercomputing Sites. http://top500.org

[17] B.Chapman, G. Jost, R. vanderPas, D.J. Kuck, *Using OpenMP: Portable Shared Memory Parallel Programming*. The MIT Press, October 31, 2007.

[18] M. Snir, S. Otto; S. Huss-Lederman, D. Walker, and J. Dongarra, *MPI: The Complete Reference*. The MIT Press, 1995.

[19] P. L. Springer, “PVM Support for Clusters”, *Proc. of International Conference on Cluster Computing*, 2001, pp.183-186.

[20] P. Uthayonpas, T. Angskun, and J. Maneesilp, “On the Building of the Next Generation Integrated Environment for Beowulf Clusters”, *Proc. of International Symposium on Parallel Architectures, Algorithms and Networks (I-SPAN)*, pp.139-144, 2002

[21] C. Motorola, *The Basics Book of X.25 Packet Switching*, 2nd edition, Addison Wesley Press, 1992.

[22] T. Berner-Lee and R. Cailliau, “WorldWideWeb: Proposal for a HyperText Project”, 1990. Available: http://www.w3.org/Proposal.html

[23] R. Fielding, J. Getty, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, “*Hypertext Transfer Protocol -- HTTP/1.1*”, RFC 2616, Jun. 1999. Available online at: http://www.ietf.org/rfc/rfc2616.txt

[24] R. Srinivasan, “RPC: Remote Procedure Call Protocol Specification version 2”, RFC 1831, August, 1995. Available online at: http://tools.ietf.org/html/rfc1831

[25] W. Grosso, *Java RMI*, O’Reilly, October 2001. ISBN: 1-56592-452-5

[26] OMG, “Common Object Request Broker Architecture: Core Specification”, March 2004, available: http://www.omg.org/docs/formal/04-03-12.pdf

[27] T. L. Thai, “Learning DCOM”, *O’Reilly Press*, April 1999. ISBN:978-1-56592-581-6

[28] D. Box, D. Ehnebuske, G. Kakivaya, et. al, “Simple Object Access Protocol 1.1”, *W3C*, May 2000. Available online at: http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

[29] T. Bellwood, “UDDI version 2.04 API specification”, *OASIS UDDI TC*, July 2002, Availabe online at: http://uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.pdf

[30] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, “Web Service Description Language 1.1”, *W3C*, March 2001, Available online at: http://www.w3.org/TR/wsdl

[31] I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Architecture*, 1998. ISBN: 978-1558604759.

[32] Internet System Consortium, http://www.isc.org/

[33] I. Foster and C. Kesselman, *The Grid2: Blueprint for a New Computing Architecture*, The Elsevier Press, November 2003. ISBN: 978-1558609334.

[34] NIST, “Definition of Cloud Computing”, *NIST*, Available online at: http://csrc.nist.gov/groups/SNS/cloud-computing/

[35] J. Taylor, “Defining e-Science”, *Research Council e-Science Core Programme*, 2000. Available online at: http://www.nesc.ac.uk/nesc/define.html

[36] M. Atkinson, “What is e-Science?”, *e-Science Envoy*, 2001, Available online at: http://www.rcuk.ac.uk/escience/default.htm

[37] ATLAS Experiment, Available online at: http://cern.ch/atlas

[38] CMS Experiment, Available online at: http://cmsdoc.cern.ch.

[39] LHCb Experiment, Available online at: http://cern.ch/lhcb

[40] ALICE Experiment, Available online at: http://aliceinfo.cern.ch

[41] Large Hadron Collider - Wikipedia, online available at: http://en.wikipedia.org/wiki/Large_Hadron_Collider

[42] S. Bethke, M. Calvetti, H. F. Hoffmann, D. Jacobs, M. Kasemann, and D. Linglin, “Report of The Steering Group of the LHC Computing Review”, *CERN-LHCC/2001-2004*, February 2001.

[43] T. Anticic, F. Carena and et.al, “The ALICE Data-Acquisition System”, *Record of IEEE Nuclear Science Symposium Conference2005,* October, 2005.

[44] J. Troska, E. Corrin, Y. Kojevnikov, T. Rohlev and J. Varela, “Implementation of the Timing, Trigger and Control System of the CMS Experiment”, *Trans. of IEEE Nuclear*

Science, Vol. 53, June 2006, pp.834-837.

[45] R. Stoica, M. Frank, N. Neufeld, and A. C. Smith, “Data Handling and Transfer in the LHCb Experiment”, *Trans. of IEEE Nuclear Science*, vol. 55, February 2008, pp. 272-277.

[46] G. Lehmann, J. Bogaerts, M. Ciobotaru, E. Palencia Cortezon and et. al, “The DataFlow System of the ATLAS Trigger and DAQ”, *Proc. of Computing in High Energy and Nuclear Physics Conference 2003 (CHEP03)*, La Jolla, California, March, 2003.

[47] I. Foster, C. Kesselman, and S. Tuecke. “The Anatomy of the Grid: Enabling Scalable Virtual Organisation”, *International Journal of High Performance Computing Application*, 15(3):200-222, 2001.

[48] I. Foster, H. Kishimoto, et. al, “Open Grid Service Architecture version 1.5”, *Open Grid Forum OGSA working group*, GFD-I.080, Jul. 2006. Available online at: http://forge.gridforum.org/projects/ogsa-wg

[49] D. Booth, H. Hass, et. al, “Web Service Architecture”, *W3C*, Feb. 2004. Available online at: http://www.w3.org/TR/ws-arch/

[50] T. Banks, “Web Service Resource Framework version 1.2”, *OASIS WSRF TC*, May 2006. Available online at: http://docs.oasis-open.org/wsrf/wsrf-primer-1.2-primer-cd-02.pdf

[51] S. Graham, “Web Service Base Notification 1.3”, *OASIS WSN TC*, Oct. 2006. Available online at http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-os.pdf

[52] DMTF, “Web Service for Management Specification”, *Distributed Management Task Force*, DSP0226, Feb. 2008. Available online at: http://www.dmtf.org/standards/published_documents/DSP0226_1.0.0.pdf

[53] I. Foster, T. Maguire, and D. Snelling, “OGSA^{TM} WSRF Basic Profile 1.0”, *OGF OGSA working group*, May, 2006. Available online at: http://forge.gridforum.org/projects/ogsa-wg

[54] S. Graham and J. Treadwell, “Web Services Resource Properties version 1.2”, *OASIS WSRF TC*, Apr. 2006, Available online at: http://docs.oasis-open.org/wsrf/wsrf-ws_resource_properties-1.2-spec-os.pdf

[55] L. Srinivasan, and T. Banks, “Web Service Resource Lifetime version 1.2”, *OASIS WSRF TC*, Apr. 2006. Available online at: http://docs.oasis-open.org/wsrf/wsrf-ws_resource_lifetime-1.2-spec-os.pdf

[56] T. Maguire, D. Snelling, and T. Banks, “Web Services Service Group version 1.2”, *OASIS WSRF TC*, Apr. 2006. Available online at: http://docs.oasis-open.org/wsrf/wsrf-ws_service_group-1.2-spec-os.pdf

[57] D. Box, E. Christensen, et. al, “Web Service Addressing 1.0”, *W3C*, Aug. 2004. Available online at: http://www.w3.org/Submission/ws-addressing/

[58] L. Liu, and S. Meder, “Web Services Base Faults version 1.2”, *OASIS WSRF* TC, Apr. 2006. Available online at: http://docs.oasis-open.org/wsrf/wsrf-ws_base_fault-1.2-spec-os.pdf

[59] J. Alexander, D. Box, et. al, “Web Services Transfer”, *W3C*, Sept. 2006. Available online at: http://www.w3.org/Submission/WS-Transfer/

[60] J. Alexander, D. Box, et. al, “Web Services Enumeration”, *W3C*, Mar. 2006. Available online at: http://www.w3.org/Submission/WS-Enumeration/

[61] D. Box, L. Felipe, et. al, “Web Services Eventing”, *W3C*, Mar. 2006. Available online at; http://www.w3.org/Submission/WS-Eventing/

[62] M. Pereira, O. Tatebe, et. al, “Resource Namespace Service Specification”, *OGF RNS working group*, May, 2006. Available online at: http://forge.gridforum.org/sf/projects/ogsa-naming-wg

[63] A. Grimshaw, and D. Snelling, “Web Service Naming”, *OGF RNS working group*, Dec. 2006. Available online at: http://forge.gridforum.org/projects/ogas-naming-wg

[64] A. Nadalin, M. Goodner, et. al, “Web Service Secure Conversation”, *OASIS Web Services Secure Exchange TC*, Mar. 2007. Available online at: http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/ws-secureconversation-1.3-os.html

[65] A. Nadalin, M. Goodner, et. al, “Web Service Trust”, *OASIS Web Service Secure Exchange TC,* Mar. 2007. Available online at: http://docs.oasis-open.org/ws-sx/ws-trust/200512

[66] A. Nadalin, C. Kaler, et. al, “Web Service Security: SOAP Message Security version 1.1”, *OASIS Web Service Security TC*, Feb. 2006. Available online at: http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf

[67] T. Imamura, B. Dillaway, and E. Simon, “XML Encryption Syntax and Processing”, *W3C XML Encryption Working Group*, Dec. 2002. Available online at: http://www.w3.org/TR/xmlenc-core/

[68] M. Bartel, J. Boyer, et. al, “XML Signature Syntax and Processing”, *W3C*, Jun. 2008. Available online at: http://www.w3.org/TR/xmldsig-core/

[69] D. Snelling, D. Merril. And A. Savva, “OGSA Basic Security Profile 2.0”, *OGF OGSA working group,* Jul. 2008. Available online at: http://forge.gridforum.org/projects/ogsa-wg

[70] D. Merrill, “OGSA Basic Security Profile 2.0 –Secure Addressing”, *OGF OGSA working group,* Oct. 2007. Available online at: http://forge.gridforum.org/projects/ogsa-wg

[71] D. Merrill, “Secure Communication Profile 1.0”, *OGF OGSA working group,* Dec. 2007. Available online at: https://forge.gridforum.org/sf/go/artf6105

[72] I. Foster, A. Grimshaw, et. al, “OGSA Basic Execution Service Version 1.0”, *OGF OGSA Basic Execution Service working group*, Nov. 2008, Available online at: http://forge.gridforum.org/sf/projects/ogsa-bes-wg

[73] D. K. Fellows, and A. Papaspyrou, “OGSA Resource Selection Services Candidate Set Generator Specification”, *OGF OGSA Resource Selection Services working group*, Mar. 2009, Available online at: http://forge.gridforum.org/sf/projects/ogsa-rss-wg

[74] A. Anjomshoaa, F. Brisard, et. al, “Job Submission Description Language (JSDL) specification Version 1.0”, *OGF Job Submission Description Language working group*, Nov. 2005, Available online at: http://forge.gridforum.org/sf/projects/jsdl-wg/

[75] D. K. Fellows, A. Papasyrou, “OGSA Resource Selection Services Basic Execution Planning Service Specification”, *OGF OGSA Resource Selection Services working group*, Available online at: http://forge.gridforum.org/sf/projects/ogsa-rss-wg

[76] M. Antonioletti, M. Atkinson, et. al, “Web Services Data Access and Integration-The Core(WS-DAI) Specification Version 1.0”, *OGF Data Access and Integration Services Working Group*, Jun. 2006. Available online at: https://forge.gridforum.org/projects/dais-wg

[77] M. Antonioletti, B. Collins, et. al, “Web Service Data Access and Integration-The Relational Realisation(WS-DAIR) Specification Version 1.0”, *OGF Data Access and Integration Services Working Group*, Jun. 2006. Available online at: https://forge.gridforum.org/projects/dais-wg

[78] M. Antonioletti, S. Hastings, et. al, “Web Service Data Access and Integration-The XML Realisation(WS-DAIX) Specification Version 1.0”, *OGF Data Access and Integration Services Working Group*, Jun. 2006. Available online at: https://forge.gridforum.org/projects/dais-wg

[79] M. Antonioletti, C. B. Aranda, et. al, “Web Services Data Access and Integration-The RDF(S) Realization (WS-DAIRDFS) RDF(S) Querying Specification Version 0.9”, *OGF Data Access and Integration Services Working Group*, May. 2009. Available online at: https://forge.gridforum.org/projects/dais-wg

[80] M. E. Gutierrez, and A. G. Perez, “Web Services Data Access and Integration-The RDF(S) Realization (WS-DAI-RDF(S)) Ontology Specification”, *OGF Data Access and Integration Services Working Group*, Arp. 2009. Available online at:

https://forge.gridforum.org/projects/dais-wg

[81] G. Klyne, J. J. Carroll, and B. McBride, "Resource Description Framework (RDF): Concepts and Abstract Syntax", *W3C Semantic Web Activity*, Feb. 2004. Available online at: http://www.w3.org/RDF/

[82] I. Mandrichenko, W. Allcock, and T. Perelmutov, "GridFTP v2 Protocol Description", *OGF Grid File Transfer Protocol Working Grid Working Group*, May 2005. Available online at: http://forge.gridforum.org/projects/gridftp-wg/

[83] N. P. C. Hong, M. Drescher, et.al, "OGSA Byte Input/Output Specification V1.0", *OGF ByteIO Working Group*, Oct. 2005. Available online at: https://forge.gridforum.org/projects/byteio-wg/

[84] M. Antonioletti, M. Drescher, et, al., "OGSA Data Movement Interface Specification Version 1.0", *OGF Data Movement Interface Working Group*, Aug. 2008. Available online at: http://forge.gridforum.org/sf/projects/ogsa-dmi-wg

[85] R. Aydt, D. Gunter, et. al., "A Grid Monitoring Architecture", *OGF Grid Monitoring Architecture Working Group*, Jul. 2001. Available online at: http://www-didc.lbl.gov/GGF-PERF/GMA-WG/

[86] DMTF, "Common Information Model (CIM) Infrastructure", *Distributed Management Task Force Inc.*, May 2009. Available online at: http://www.dmtf.org/standards/cim/

[87] JSR 255, "Java^{TM} Management Extensions (JMX) Specification version 2.0", *Sun Microsystems*, Dec. 2007, Available online at: http://www.jcp.org/en/jsr/detail?id=255

[88] S. ANDREOZZI, S. Burke, et. al, "GLUE Specification version 2.0", *OGF GLUE Working Group*, Mar. 2009. Available online at: http://forge.gridforum.org/sf/projects/glue-wg

[89] R. Butler, and T. J. Genovese, "Certificate Policy Model", *OGF Grid Certificate Policy Working Group*, Jun. 2003. Available online at http://forge.gridforum.org/sf/projects/gcp-wg

[90] D. Chadwick, “Functional Components of Grid service Provider Authorisation Service Middleware”, *OGF Grid Authorization Working Group*, Apr. 2008.

[91] T. Moses, “eXtensible Accesss Control Markup Language (XACML) Version 2.0”, *OASIS eXtensible Access Control Markup Language (XACML) TC*, Feb. 2005.

[92] E. Maler, P. Mishra, and R. Philpott, “Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) Version 2.0”, *OASIS Security Assertion Markup Language TC”*, Feb. 2007. Available online at: http://saml.xml.org/saml-specifications

[93] L. Smarr, and C. E. Catlett, “Metacomputing”, *Communications of the ACM*, 35(6), June 1992. pp.44-52.

[94] I. Foster, “Globus Toolkit Version 4: Software for Service-Oriented Systems”. *Journal of Computer Science and Technology*, 21(4), 2006. pp513-520

[95] t. DeFanti, I. Foster, et. al, “Overview of the I-WAY: Wide Area Visual Supercomputing”. *International Journal of Supercomputing Applications*, 10(2), 1996.

[96] W. Allcock, J. Bresnahan, et. al, “The globus extensible input/output system (XIO): a protocol independent IO system for the grid”, *Proc. of 19th IEEE International Parallel and Distributed Processing Symposium*, Apr. 2005. pp.8

[97] K. Czajkowski, I. Foster, et. al. “A Resource Management Architecture for Metacomputing Systems”**,** *Proc. IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing*, pg. 62-82, 1998.

[98] Globus, “The Globus Resource Specification Language v1.0”, online available at: http://www.globus.org/toolkit/docs/2.4/gram/rsl_spec1.html

[99] M. Ripeanu, I. Foster, “A Decentralized, Adaptive, Replica Location Service”, *11th IEEE International Symposium on High Performance Distributed Computing (HPDC-11)*, Edinburgh, Scotland, July 24-16, 2002.

[100] B. Allcock, J. Bester, et. al, “Data Management and Transfer in High Performance Computational Grid Environments”, *Parallel Computing Journal***,** Vol. 28 (5), May 2002, pp. 749-771.

[101] H. Stockinger, A. Samar, et. al, “File and Object Replication in Data Grids”, *Journal of Cluster Computing***,** 5(3)305-314, 2002.

[102] M. Antonioletti, M.P. Atkinson, et. al., “OGSA-DAI Status Report and Future Directions”**,** *Procs. of the UK e-Science All Hands Meeting 2004***,** September 2004.

[103] X. Zhang and J. Schopf, **“**Performance Analysis of the Globus Toolkit Monitoring and Discovery Service, MDS2”**,** *Procs. of the International Workshop on Middleware Performance (MP 2004), part of the 23rd International Performance Computing and Communications Workshop (IPCCC)*, April 2004.

[104] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke, “A Security Architecture for Computational Grids”, *Proc. 5th ACM Conference on Computer and Communications Security Conference*, pp. 83-92, 1998.

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

[106] Ian Foster, Carl Kesselman, Laura Pearlman, et.al. “The Community Authorization Service: Status and Future”, *Procs. of Computing in High Energy Physics 03 (CHEP '03)*, 2003.

[107] The Virtual Data Toolkit web site http://vdt.cs.wisc.edu

[108] B. Seqal, L. Robertson, F. Gaqliardi, and F. Carminati, “Grid Computing: the European Data Grid Project”, *Conf. of IEEE Nuclear Science Symposium*, Lyon, France.

[109] I. Bird, L. Robertson, and J. Shiers, “Deploying the LHC computing grid-the LCG service challenges”, *appears in Local to Global Data Interoperability-Challenges and*

Technologies, Jun. 2005. pp.160-165.

[110] gLite Middleware. http://cern.ch/glite.

[111] Portable Batch System, http://www.pbsgridworks.com/Product.aspx?id=11

[112] D. Thain, T. Tannenbaum, and M. Livny, "Condor and the Grid", *Grid Computing: Making The Global Infrastructure a Reality*, John Wiley, 2003. ISBN: 0-470-85319-0

[113] M. Q. Xu, "Effective metacomputing using LSF Multicluster", *Proc. of First IEEE/ACM International Symposium on Cluster Computing and the Grid*, Australia, May 2001. pp.100-105.

[114] C. Aiftimiei, P. Andreetto, et. al., "Design and Implementation of the gLite CREAM Job Management Service", *INFN Technical Note*, May, 2009. Available online at: http://www.lnf.infn.it/sis/preprint/detail.php?id=5147

[115] Condor-C, http://www.cs.wisc.edu/condor/

[116] Batch Local ASCII Helper, http://egee-jra1-wm.mi.infn.it/egee-jra1-wm/blah_porting_notes.txt

[117] CERN Advanced STORage manager, http://castor.web.cern.ch/castor/

[118] M. Ernst, P. Fuhrmann, and T. Mkrtchyan, "Managed data storage and data access services for data Grids", *Conf. on Computing in High Energy and Nuclear Physics (CHEP)*, Mumbai, India, Oct, 2004.

[119] G. A. Steward, D. Cameron, G. A. Cowan, and G. McCance, "Storage and data management in EGEE", *Procs. of the fifth Australian symposium on ACSW frontiers*, Darlinghurst, Australia, Australia, 2007. pp.69-77.

[120] G. Avellino, S. Beco, B. Cantalupo, et. al., "The Data-Grid Workload Management System: Challenges and Results", *Journal of Grid Computing*, 2(4):353-367, 2004.

[121] F. Pacini, "Job Description Language Attributes Specification", Available online at: https://edms.cern.ch/document/555796/1

[122] B. Coghlan, A. W. Cooke, A. Datta, et. al., "R-GMA: A Grid Information and Monitoring System" *Conf. on UK e-Science all hands*, Sheffield, 2-4 September 2002.

[123] R. Alfieri, R. Cecchini, V. Ciaschini, et. al, "VOMS: an Authorisation System for Virtual Organisations", *Procs. of Computing in High Energy Physics (CHEP)*, India, 2004.

[124] M. Romberg, "The UNICORE architecture: seamless access to distributed resources", *Procs. of 8th International Symposium on High Performance Distributed Computing*, Redondo, USA, 1999. pp.287-293.

[125] R. Ratering, A. Lukichev, M. Riedel, et. al, "GridBeans: Support e-Science and Grid Applications", *Procs. of 2nd IEEE internal conference on e-Science and Grid computing*, Dec. 2006. pp.45.

[126] J. Novotny, M. Russell, and O. Wehrens, "GridSphere: an advanced portal framework", *Procs. of 30th Euromicro Conference*, Aug. 2004. pp.412-419.

[127] V. Venturi et al. "Using SAML-based VOMS for Authorization within Web Services-based UNICORE Grids", *Proc.UNICORE Summit at Euro-Par 2007, Rennes, France*, 2007.

[128] M. Ellert, M. Grnager, A. Konstantinov, et. al, "Advanced Resource Connector middleware for lightweight computational Grids", *Future Generation Computer Systems*, 23:219-240, 2007.

[129] H. Nakada, S. Matsuoka, K. Seymour, et. al, "A GridRPC Model and API for Advanced and Middleware Applications", *OGF Grid Remote Procedure Call Working Group*, Available online at: https://forge.gridforum.org/projects/gridrpc-wg/

[130] MPICH-G2, http://www.globus.org/grid_software/computation/mpich-g2.php

[131] Directed Acyclic Graph Manager, http://www.cs.wisc.edu/condor/dagman/

[132] M. A. Pettipher, A. Khan, T. W. Robinson, and X. Chen, "Review of Accounting and Usage Monitoring (final Report)", *JISC Final Report*, Jul. 2007.

[133] P. Garfjall, “Accounting in Grid Environments, an architectural proposal and a prototype implementation”, *Master Thesis*, 27 May 2004, Umea University, Sweden, available at: http://www.cs.umu.se/~peterg/thesis/thesis.pdf

[134] Webster, “Definition of Accounting”, Available online at: http://www.merriam-webster.com/dictionary/accounting

[135] J. Coles, “The evolving grid deployment and operations model with EGEE, LCG and Gridpp”, *Proceedings of 1st International Conf. on e-Science and Grid computing*, Dec. 2005, pp8

[136] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, et. al., “Web Services Agreement Specification”, *Open Grid Forum GRAAP WG*, GFD.107, March 2007, available at: http://www.ogf.org/documents/GFD.107.pdf

[137] L. McGinnis, R. Mach, R. Lepro-Mez, and S. Jackson, “Usage Record-Format Recommendation version 1.0”, *Open Grid Forum Usage Record Working Group*,GFD.58, September 2006. Available online at: https://forge.gridfourm.org/projects/ur-wg/

[138] J. Ainsworth, S. Newhouse, and J. MacLaren, “Resource Usage Service Based on WS-I Basic Profile 1.0 (draft)”, *Open Grid Forum Resource Usage Service Working Group*, August, 2006.

[139] K. Weeks, “The National Grid Service User Accounting System”, *Proc. of UK e-Science All Hands Meeting 2007*, September, 2007.

[140] J. D. Ainsworth, J. MacLaren, J. M. Brooke, *Implementing a Secure, Service Oriented Accounting System for Computational Economics*, CCGrid, 2005.

[141] R. Byrom, R. Cordenonsi, and L. Cornwall, “APEL: An implementation of Grid accounting using R-GMA”, *UK e-Science All Hands Conference*, Nottingham, September 2005.

[142] S. Fisher, “Building the e-Science Grid in UK: Grid Information Service”, *Proc. Of UK e-Science All Hands Meeting 2003*, Nottingham, September 2003.

[143] R. Byrom and D. Kant, “LCG Accounting Schema”, *EGEE Support and Management Activity (SA1) document*, Oct. 19th, 2004. Available online at: http://www.egee.cesga.es/EGEE-SA1-SWE/accounting/guides/apel-schema.pdf.

[144] R. M. Piro, A. Guarise, and A. Werbrouck, “An economy-based accounting infrastructure for the datagrid”, *Proceedings of Fourth International Workshop on Grid Computing (IEEE, 2004)*, pp 202-204

[145] R. M. Piro, M. Pace, A. Ghiselli, A. Guarise, E. Luppi, G. Patania, L. Tomassetti, and A. Werbrouck, “Tracing Resource Usage over Heterogeneous Grid Platforms: A Prototype RUS interface for DGAS”, *Proceedings of International Conf. on e-Science and Grid Computing*, Dec. 2007, pp93-101

[146] P. Gardfjall, E. Elmroth, L. Johnsson, and O. Mulmo, “Scalable Gridwide capacity allocation with SweGrid Accounting System” *Concurrency and Computation: Practice and Experience*, John Wiley and Sons Ltd, June 2008.

[147] P. Canal, S. Borra, M. Melani, “GRATIA, a resource accounting system for OSG”, *Proc. of Computing in High Energy and Nuclear Science 2006 (CHEP06)*, Mumbai, Inida, February 2006.

[148] W. Frings, M. Riedel, A. Streit, D. Mallmann, D. Snelling and V. Li, “LLview: User-Level Monitoring in Computational Grids and e-Science Infrastructure”, *German eScience 2007 conference*, May 2007.

[149] “ARCO: N1 Grid Engine 6 Accounting and Reporting Console (white paper)”, *Sun Microsystems. Inc.* May 2005.

[150] W. Gentzsch, “Sun Grid Engine: towards creating a compute power grid”, *Proc. of 1st IEEE/ACM International Symposium on Cluster Computing and the Grid 2001 (CCGrid2001)*, May 2001.

[151] Gold Allocation Manager, http://www.clusterresources.com/pages/products/glod-acllocation-manager.php

[152] HPCx, http://www.hpcx.ac.uk/

[153] HECToR, http://www.epcc.ed.ac.uk/msc/hpc-systems/hector/

[154] J. Shiers, “Memorandum of Understanding for Collaboration in the Deployment and Exploitation of the Worldwide LHC Computing Grid”, Jan. 13th, 2009. Available online at “http://lcg.web.cern.ch/lcg/mou.htm”.

[155] X. Chen and A. Khan, “Aggregative accounting service enabling economic modelling for commercial grid”, *Conf. on Grid technology for financial modelling and simulation*, Feb. 3-4, 2006, Palermo, Italy.

[156] X. Chen and A. Khan, “Development and Performance of Resource Usage Service in WLCG”, *Conf. on IEEE Nuclear Science Symposium*, Oct. 2006. pp.603-606.

[157] X. Chen and A. Khan, “Development of Multi-Grid Resource Usage Service in LCG”, *Conf. of International Symposium on Grid Computing (ISGC) 2007*, Mar. 26-29, 2007, Taiwan.

[158] X. Chen, R. M. Piro, P. Canal, et. al, “Aggregate Usage Representation Version 1.0”, *OGF Usage Record working group*, Dec. 2006. Available online at: https://forge.gridfourm.org/projects/ur-wg/

[159] I. Bird and D. Kant, “EGEE-II Operational Accounting Portal”, *EGEE Management Service Activity (MSA) document*, July 19th, 2007, online available at: “https://edms.cern.ch/document/726137/4”

[160] P. Rey, J. Lopez, C. Fernadez, D. Kant and J. Gordon, “The Accounting Infrastructure in EGEE”, *Proc. of 1st Iberian Grid Infrastructure Conference*, May 14-16th, 2007, Spain.

[161] XML:DB API, http://xmldb-org.sourceforge.net/xapi/

[162] Hibernate, https://www.hibernate.org/

[163] Web Service Interoperability, http://www.ws-i.org/

[164] Apache Axis Project, online available at: "http://ws.apache.org/axis/".

[165] Grails, http://www.grails.org/

[166] Groovy, http://groovy.codehaus.org/

[167] ISO 8601, "Date Elements and Interchange formats – Information Interchange Representation of Dates and Times", *International Standard Organization*, Dec. 3rd, 2004.

[168] Spring Framework, http://www.springsource.org/

[169] Acegi, http://www.grails.org/AcegiSecurity+Plugin

[170] J. Clark and S. DeRose, "XML Path Language Version 1.0", Nov. 1999. Available online at: http://www.w3.org/TR/xpath

[171] X. Chen, "OGSA Resource Usage Service IDL WS-I Rendering", *OGF Resource Usage Service working group*, Dec. 2007. Available online at: https://forge.ggf.org/sf/sfmain/do/go/artf6090?nav=1&selectedTab=attachments

[172] X. Chen and A. Khan, "GRUS: An Extensive Solution to Resource Usage Service", *Conf. on IEEE Nuclear Science Symposium*, Dresden, Germany, Oct. 2008.

[173] G. Netzer, "OGSA Resource Usage Service-Core IDL Specification Draft Version 1.0", *OGF Resource Usage Service working group*, Sept. 2007. Available online at: https://forge.gridforum.org/sf/go/artf6015

[174] J. Alexander, D. Box, L. F. Cabrera, et. al., "Web Service Enumeration", *W3C*, Mar. 2006. Available online at: http://www.w3.org/Submission/WS-Enumeration/

[175] D. Chamberlin, M. Dyck, D. Florescu, et. al., "XQuery Update facility 1.0", *W3C XML Activity*, Jun. 2009. Available online at: http://www.w3.org/TR/xquery-update-10/

[176] S. Boag, D. Chamberlin, M. F. Fernandez, et. al., “XQuery 1.0: An XML Query Language”, *W3C XML Query*, Jan. 2007. Available online at: http://www.w3.org/TR/xquery/

[177] Apache Ant, http://ant.apache.org/

[178] Wiseman, “Java Implementation of Web Service Management”. Available online at: https://wiseman.dev.java.net/

[179] Java Web Services Developer Pack, http://java.sun.com/webservices/downloads/previous/index.jsp

[180] Hibernate Query Language, http://docs.jboss.org/hibernate/core/3.3/reference/en/html/queryhql.html

[181] JSR 175, “A Metadata Facility for the Java^{TM} Programming Language”, *Sun Inc.* Available online at: http://jcp.org/en/jsr/detail?id=175

[182] Simple API for XPath, http://sourceforge.net/projects/saxpath/

[183] Jaxen, http://jaxen.org/

[184] Simple API for XML, http://www.saxproject.org/

[185] Document Object Model, http://www.w3.org/DOM/

[186] XML Object Model, http://ws.apache.org/commons/axiom/

[187] K. Ballinger, D. Ehnebuske, M. Gudgin, et. al., “WS-I Basic Profile Version 1.0”, *WS-I Interoperability Organisation*, Apr. 2004. Available online at: http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html

[188] S. Crouch, D. Fellows, X. “Experiences of Using Usage Record (UR) Version 1.0”, *OGF Usage Record Working Group*, Oct. 2009, Available online at: http://forge.gridforum.org/projects/ur-wg

[189] Korpela, E., D. Werthimer, D. Anderson, J. Cobb, and M. Lebofsky. "SETI@home - Massively distributed computing for SETI", *Computing in Science and Engineering*, 3(1), p. 79, 2001.

[190] B. Jacob, R. Lanyon-Hogg, D. K. Nadgir, and A. F. Yassin, “A Practical Guide to the IBM Autonomic Computing Toolkit”, *IBM Redbook Series*, Apr. 2004, Available online at: http://www.redbooks.ibm.com/redbooks/pdfs/sg246635.pdf.

[191] A. Andrieux, K. Czajkowski, et. al, “Web Services Agreement Specification (WS-Agreement)”, *OGF Grid Resource Allocation Agreement Protocol Working Group*, March 14, 2007, Available online at: http://forge.gridforum.org/sf/projects/graap-wg.

[192] Portable Batch System (PBS), http://www.pbsworks.com/

[193] S. Tuecke, K. Czajkowski, I. Foster, et. al, “Open Grid Service Infrastructure (OGSI) Version 1.0”, *Open Grid Forum Open Grid Service Infrastructure Working Group*, June 27, 2003. Available online at: http://www.globus.org/toolkit/draft-ggf-ogsi-gridservice-33_2003-06-27.pdf.

[194] V. Bullard, B. Murray, and K. Wilson, “An Introduction to Web Service Distributed Management (WSDM)”, *OASIS Web Service Distributed Management (WSDM) TC*, Feb. 24, 2006. Available online at: http://www.oasis-open.org/committees/download.php/16998/wsdm-1.0-intro-primer-cd-01.doc.

[195] D. Chamberlin, M. Dyck, D. Florescu, J. Melton, J. Robie, and J. Simeon, “Xquery Update Facility 1.0”, *World Wide Web Candidate Recommendation*, June 2009. Avaiable online at: http://www.w3.org/TR/xquery-update-10/.

[196] X. Chen, G. Wills, L. Gilbert, and D. Bacigalupo, “TeciRes: A Technical Review of Using Cloud for Research”, *JISC Documents and MultiMedia Repository*, June, 2010. Online available at: http://tecires.ecs.soton.ac.uk/publications.