Universidade de Aveiro Departamento de Electrónica, Telecomunicações e Informática
2009

David João Gonçalves Pacheco

Ciência 2.0: Partilha de dados científicos na Grid

Science 2.0: Sharing Scientific Data on the Grid

Universidade de Aveiro
2009

Departamento de Electrónica, Telecomunicações e Informática

David João Gonçalves Pacheco

Ciência 2.0: Partilha de dados científicos na Grid

Science 2.0: Sharing Scientific Data on the Grid

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática (M.I.E.C.T.), realizada sob a orientação científica do Professor Doutor José Maria Amaral Fernandes, Professor Auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro e do Mestre Ilídio Fernando de Castro Oliveira, Professor Assistente Convidado do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

Dedico este trabalho à minha família.

o júri

presidente

Prof. Doutor Joaquim Arnaldo Carvalho Martins
professor catedrático do Departamento de Electrónica Telecomunicações e Informática da Universidade de Aveiro

Profª. Doutora Inês de Castro Dutra
professora auxiliar do Departamento de Ciências de Computadores da Faculdade de Ciências da Universidade do Porto

Prof. Doutor José Maria Amaral Fernandes
professor auxiliar do Departamento de Electrónica Telecomunicações e Informática da Universidade de Aveiro

Mestre Ilídio Fernando de Castro Oliveira
professor assistente convidado do Departamento de Electrónica Telecomunicações e Informática da Universidade de Aveiro

agradecimentos

Mestre Ilídio Oliveira, Professor Doutor José Maria Fernandes, Professor Doutor João Paulo Cunha, Eng. Luís Alves, Eng. Micael Pedrosa, Sérgio Lima.

palavras-chave

Computação Grid, Repositórios de dados científicos, Imagem médica, e-Ciência

resumo

A computação assume-se cada vez mais como um recurso essencial em ciência, levando ao surgimento do termo e-Ciência para designar a utilização de tecnologias de computação avançada para suportar a realização de experiências científicas, a preservação e partilha do conhecimento.

Uma das áreas de aplicação do conceito de e-Ciência é o tratamento e análise de imagens médicas. Os processos que lidam com imagem médica, tanto ao nível clínico como de investigação, são exigentes em relação ao suporte computacional, devido aos algoritmos de processamento de imagem que requerem e à elevada capacidade de armazenamento relacionada com volume das imagens geradas.

As políticas públicas e os avanços tecnológicos recentes orientados para a e-Ciência, têm vindo a apoiar o desenvolvimento da computação em Grid, tanto a nível dos *middlewares* como da instalação de capacidade de produção, como um sistema de computação avançado que permite a partilha de recursos, instrumentos científicos e boas práticas em comunidades virtuais.

Este trabalho tem como objectivo desenvolver uma estratégia e um protótipo para o armazenamento de dados médicos na Grid, visando a sua utilização em investigação. Uma preocupação diferenciadora prende-se com o objectivo de colocar as potencialidades da Grid ao serviço de utilizadores não técnicos (e.g. médicos, investigadores), que acedem a serviços de processamento e de armazenamento e catalogação de dados de forma transparente, através de um portal Web.

O protótipo desenvolvido permite a investigadores na área das neurociências, sem conhecimentos específicos da tecnologia Grid, armazenar imagens e analisá-las em Grids de produção existentes, baseadas no *middleware* gLite.

keywords Grid Computing, Cientific Data Repositories, Medical Imaging, eScience

abstract Computing has become an essential tool in modern science, leading to the appearance of the term e-Science to designate the usage of advanced computing technologies to support the execution of scientific experiments, and the preservation and sharing of knowledge.

One of e-Science domain areas is the medical imaging analysis. The processes that deal with medical images, both at clinical and investigation level, are very demanding in terms of computational support, due to the analysis algorithms that involve large volumes of generated images, requiring high storage capabilities.

The recent public policies and technological advances are e-Science oriented, and have been supporting the development of Grid computing, both at the middleware level and at the installation of production capabilities in an advanced computing system, that allows the sharing of resources, scientific instrumentation and good practices among virtual communities.

The main objective of this work is to develop a strategy and a prototype to allow the storage of medical data on the Grid, targeting a research environment. The differentiating concern of this work is the ability to provide the non-experts users (e.g: doctors, researchers) access to the Grid services, like storage and processing, through a friendly Web interface.

The developed prototype allows researchers in the field of neuroscience, without any specific knowledge of Grid technology, to store images and analyse them in production Grid infrastructures, based on the gLite middleware.

Contents

List of Figures

List of Tables

List of Acronyms

AC	Attribute Certificate
ACL	Access Control List
AJAX	Asynchronous Javascript And XML
AMGA	ARDA Metadata Catalogue
APEL	Accounting Processor for Event Logs
APGridPMA	Asia Pacific Grid Policy Management Authority
API	Application Programming Interface
BDII	Berkeley Database Information Index
BIRN	Biomedical Informatics Research Network
BIRN-CC	BIRN Coordinating Center
C-OMEGA	Open Middleware Enabling Grid Applications
CA	Certification authority
caBIG	cancer Biomedical Informatics Grid
CASTOR	CERN Advanced STORage manager
CE	Computing Element
CERN	European Organization for Nuclear Research
CGSP	ChinaGrid Supporting Platform
CLI	Command Line Interface
CRUD	Create, Read, Update and Delete
CT	Computed Tomography
DAG	Directed Acyclic Graph
DAGMan	DAG Manager
DGAS	Distributed Grid Accounting System
DICOM	Digital Imaging and Communications in Medicine
DPM	Disk Pool Manager
DRS	Data Replication Service

EC	European Commission
EDG	European Data Grid Project
EDT	DataTAG–Data TransAtlantic Grid Project
EEG	Electroencephalogram
EELA	E-infrastructure shared between Europe and Latin America
EGA	Enterprise Grid Alliance
EGEE	Enabling Grids for E-sciencE
EuGridPMA	European Grid Policy Management Authority
fMRI	functional MRI
GAT	Grid Application Toolkit
GFAL	Grid File Access Library
GG	Grid Gate
GGF	Global Grid Forum
GIIS	Grid Index Information Server
GLUE	Grid Laboratory Uniform Environment
GMA	Grid Monitoring Architecture
GRAM	Grid Resource Allocation Manager
GRIDCC	Grid Enabled Remote Instrumentation with Distributed Control and Computation
GRIS	Grid Resource Information Server
GSIDCAP	GSI dCache Access Protocol
GT	Globus Toolkit
GUID	Grid Unique Identifier
HEP	High Energy Physics
HPC	High Performance Computing
IEETA	Instituto de Engenharia, Electrónica e Telemática de Aveiro
IGDM	IEETA Grid Data Middleware
IGF	IEETA Grid Framework
IGTF	International Trust Grid Federation
INFN	Italy's National Institute for Nuclear Physics
IS	Information System
iVDGL	International Virtual Data Grid Laboratory
J2EE	Java 2 Enterprise Edition

JDL	Job Description Language
JP	Job Provenance
JSDL	Job Submission Description Language
JSDL-WG	JSDL Working Group
JSF	Java Server Faces
LB	Logging and Bookkeeping
LCG	LHC Computing Grid
LDAP	Lightweight Directory Access Protocol
LFC	LCG File Catalogue
LFN	Logical File Name
LHC	Large Hadron Collider
LRMS	Local Resource Managing System
MDS	Monitoring and Discovery System
MPI	Message Passing Interface
MR	Magnetic Resonance
MRI	Magnetic Resonance Imaging
NS	Network Server
OASIS	Organization for the Advancement of Structured Information Standards
OGF	Open Grid Forum
OGSA	Open Grid Services Architecture
OGSA-DAI	OGSA–Data Access and Integration
OGSI	Open Grid Service interface
OMII-China	Open Middleware Infrastructure Institute for
OMII-Europe	Open Middleware Infrastructure Institute for Europe
OMII-UK	Open Middleware Infrastructure Institute for United Kingdom
OpenLDAP	Open source implementation of LDAP
PERMIS	PrivilEge and Role Management Infrastructure Standards validation
PET	Positron Emission Tomography
PFN	Physical File Name
PIC	Port d'Informatió Cientifica
PLS	Partial Least Square
PM	Package Manager

PPS	Pre-Production Service
PVM	Parallel Virtual Machine
R-GMA	Relational GMA
RA	Registration Authority
RB	Resource Broker
RFIO	Remote File Input/Output protocol
RFT	Reliable File Transfer
RLS	Replica Location Service
SARS	Severe Acute Respiratory Syndrome
SARSGrid	SARS Grid
SE	Storage Element
Share	Supporting and structuring Healthgrid Activities and Research in Europe
SIAS	Sistemas de Informação na Área da Saúde
SOA	Service Oriented Architecture
SPM	Statistical Parametric Mapping
SRB	Storage Resource Broker
SRM	Storage Resource Manager
SURL	Storage URL
TAGPMA	The Americas Grid Policy Management Authority
TGCP	TeraGrid Copy
TURL	Transport URL
UI	User Interface
UML	Unified Modeling Language
UNICORE	UNiform Interface to COmputing REsources
URI	Uniform Resource Identifier
UUID	Universally Unique Identifier
VDT	Virtual Data Toolkit
VFS	Virtual File System
VGF	Virtual Grid Folder
VO	Virtual Organization
VOMS	Virtual Organization Management System
WISDOM	Wide In Silico Docking On Malaria

WLCG	Worldwide LHC Computing Grid Project
WMS	Workload Manager Service
WN	Worker Node
WS	Web Service
WS-GRAM	Web Service–Grid Resource Allocation and Management
WSRF	WS-Resource Framework
XML	eXtensible Markup Language
XUL	XML User interface Language

Glossary

AccessGrid is an ensemble of resources including multimedia large-format displays, presentation and interactive environments, and interfaces to Grid middleware and to visualization environments. These resources are used to support group-to-group interactions across the Grid.

Condor project develops and evaluates policies and mechanisms to enable High Throughput Computing. The Condor is a specialized workload management system for computer-intensive jobs. It provides a job queuing mechanism, scheduling policy, priority scheme, resource monitoring and resource management.

D-Grid is the German National Grid Initiative (see http://www.d-grid.de/). This project is composed by two phases (D-Grid I (2005-2008) and D-Grid II (2007-2010)).

DataGrid was a project funded by European Union whose objective was to build the next generation computing infrastructure providing intensive computation and analysis of shared large-scale databases (see http://eu-datagrid.web.cern.ch/). The project time span was three years (2001-2004).

DataTag Project that aims the creation of a large-scale intercontinental grid testbed between the European Community and the USA, focusing mainly in the interoperability issues (see http://datatag.web.cern.ch/datatag/).

dCache is a system for storing and retrieving huge amounts of data, distributed among a large number of heterogeneous server nodes, under a single virtual filesystem tree with a variety of standard access methods.

e-Health is the term given to modern information and communication technologies that support healthcare services and practice.

EGEE Project aims the creation of a seamless Grid infrastructure for e-Science. (see http://www.eu-egee.org/). The EGEE project is composed by three phases (EGEE I (2004-2006), EGEE II (2006-2008) and EGEE III (2008-2010)), being currently at the third and last phase of the project.

EELA-2 Project aims at building a high capacity, production-quality, scalable Grid Facility, providing round-the-clock, worldwide access to distributed computing, storage and network resources needed by the wide spectrum of Applications from European - Latin American Scientific Collaborations.

Embrace is a project that aims to integrate the major databases and software tools in bioinformatics, using existing methods and emerging Grid service technologies (see http://www.embracegrid.info/).

gLite is the codename of the middleware that is currently being developed in the EGEE project (see http://glite.web.cern.ch/glite/).

GridFTP is a high-performance, secure, reliable data transfer protocol optimized for high bandwidth networks.

GridSphere is a standards based portlet framework for building web portals. It also provides a set of portlet web applications that work seamlessly with the GridSphere framework to provide a complete Grid portal development solution.

GSI is a Grid Security Infrastructure based in the public key cryptography. It allows, among other things, the use of certificates to authenticate users and services (see http://www.globus.org/security/).

HealthGrid initiative is a community composed by specialists from different areas that aim to create a fully operational European/International Grid to support health projects.

MyProxy is a credential management service. It manages X.509 Public Key Infrastructure service credentials and combines a online credential repository and a online certificate authority.

NAREGI is the Japanese National Research Grid Initiative.

Shibboleth is a standards-based, open source middleware software which provides Web Single SignOn (SSO) across or within organizational boundaries. It allows sites to make informed authorization decisions for individual access of protected online resources in a privacy-preserving manner (see http://shibboleth.internet2.edu/).

TeraGrid is a Grid infrastructure that connects the United States of America National Science Foundation Supercomputer Centers (see http://www.teragrid.org/).

1. Introduction

1.1. Motivation and context

A great concern among the medical community is the development of new and better methods for the improvement of our society's healthcare. In the last few years we have witnessed a great evolution in the areas of medical science namely in diagnosis methods, in particular in medical imaging techniques that can provide valuable information for the doctors and medical researchers. As a result of these evolutions, nowadays in all major medical institutions, the use of medical imaging based techniques is becoming mandatory.

This has also been translated in a great demand for the development of solutions to make efficient use of the technological resources to support and make available medical imaging in medical environments. One example is the size and number of medical images, increasing the need to store and share these images for diagnosis and research purposes. With the support of digital systems to store medical images, it is possible to share these images among several institutions to improve diagnosis, and allow investigations by the medical community among different institutions. However there are issues concerning privacy, security and technological solutions making the sharing of medical images a delicate matter [1]. Simultaneously, new solutions provide the potential of accessing the digital resources computing power, to run advanced image analysis algorithms, improving the clinical diagnosis and consequently affect the final decision.

To face the new needs of the scientific and medical community in particular, Grid computing is emerging as an interesting solution to store, share and process the huge volumes of medical images, generated by normal day-to-day routines in medical institutions, as it is already used in several scientific fields, like nuclear physics and biomedical [2,3].

Scientific disciplines like astronomy, physics, biology, medicine, etc, are feeling the need to improve their research methods with advanced computational resources [4]. Examples of projects that experience the compelling need for enhanced computational resources, both for the storage of large quantities of information, and/or run intensive algorithms are projects like the LHC (Large Hadron Collider) [5] located at CERN, that will generate huge amounts of data that needs to be stored and processed [6], or projects like BING [7], related with brain imaging research methods, being currently developed by group SIAS (Sistemas de Informação na Área da Saúde) at IEETA. SIAS research interests include medical imaging and the group has naturally included the advanced computing support provided by Grid in its research agenda.

Nevertheless, Grids still expose many technical idiosyncrasies, making it difficult for regular users to harness their true potential, as a resource to store large amounts of data and execute computationally intensive algorithms. We believe that end-users should be shielded from the underlying complexities by friendly domain applications [8].

1.2. Objectives

Storage of large amounts of data in Grid environments is still a very complex issue due to the heterogeneity and distributed nature of the Grid environment. One example is the lack of simple unified concepts such as shared file system or a common name-space, which adds complexity to any transparent and ubiquitous Grid based storage solution. Another example is the storage of metadata within a Grid ”entity”, that is, the data about the stored files that allow categorizing and querying the information on the Grid. It is required to develop custom solutions to find the stored files and describe their contents. There are already some Grid middleware technologies allowing the user to perform “low-level” operations on the Grid, like submitting jobs or storing data, and manage the files metadata, but those still lack in easy to use API's and user friendly interfaces, like Web Portals for specific tasks.

The main goal of this work is to develop a set of software components to facilitate the access to Grid-enabled infrastructures, allowing end-users to benefit from the Grid without being exposed to its underlying complexities. This will be achieved at two levels:

- for the application developer, we propose a services *Framework* (IGF that stands for IEETA Grid Framework), facilitating the “gridification” of applications;
- for the end-user, we propose a Web portal (MAGI, that stands for Medical Application Grid Interfacing portal) directed to non-expert users, like doctors and scientists.

Through MAGI, users will be able to perform their medical imaging analysis tasks, like storing, sharing and running analysis algorithms, without having to worry about the underlying complexity of the system that are managed by IGF. Ideally the user should not need to know that he is using a Grid environment.

The MAGI should offer a user friendly interface, and an attractive look. Ideally the portal should have a *drag-and-drop* look and feel, to give the users the idea that they are dragging the objects (e.g: files) to containers (e.g: folders), and provide a very intuitive interface.

To support MAGI, IGF will provide high-level programming abstractions and have good documentation to support it. Currently there aren't many *frameworks* available in the area of *Grid* storage, and the ones that exist lack in documentation and/or easy to use approaches.

Both MAGI and IGF will be deployed and validated in the context of the European project EGEE (Enabling Grids for E-sciencE) [9], which is the largest *Grid* infrastructure in the world.

1.3. Dissertation structure

This dissertation is divided into the following chapters, excluding this one:

• **Chapter 2 - Background Concepts and State of the Art,** presents the emergence of e-Science and Science 2.0 and their great importance in life sciences. The chapter also contains an introduction to the Grid itself, including the concept behind the technology, its architecture and the community. It discusses the main Grid middleware technologies and *frameworks* in the area of storage and *metadata,* as well as the main projects that existed or are currently undergoing in the area of data Grids. It also presents a brief introduction to the medical imaging research state of the art, and the relation of this scientific area with computing;

• **Chapter 3 – Supporting brain imaging research workflows,** discusses the specific requirements and use cases of a brain imaging research portal, and possible improvements that can be introduced with the usage of Grids to execute brain imaging research workflows;

• **Chapter 4 – Grid interfacing framework,** presents a new *framework* called IGF (IEETA Grid Framework) created to tackle the problems discussed earlier. This chapter presents the *framework's* architecture, as well as all the *design* choices and software methodologies used;

• **Chapter 5 – e-Science Portal for brain imaging research,** presents the integration of the previously discussed *framework* IGF, with the MAGI (Medical Applications Grid Interfacing) system Web portal to provide a high-level interface to the basic Grid services. This section presents the implemented architecture and the technology choices used to implement the portal, as well as practical results obtained with the usage of the portal;

• **Chapter 6 – Conclusions and Future Work,** presents the sum up of the most important aspects during the development of this work, discusses the results obtained and presents some of the lessons learned as well as possible future work within this area of research.

2. Background Concepts and State of the Art

2.1. The emergence of e-Science and the Grid

The term e-Science (or eScience) appeared in 1999, and it is used to denote the systematic development of research methods that exploit advanced computational resources. Such methods enable researchers the access to computational resources held on widely-dispersed computers as if they were on their own desktops. The resources can include data collections, very large-scale computing resources, scientific instruments and high performance visualization. Examples of the scientific areas include social simulations, particle physics, earth sciences and bioinformatics.

Another important concept that can be considered as an evolution of e-Science is Science 2.0. The term Science 2.0 comes from the application of Web 2.0 technologies to improve and facilitate scientific research tasks. The term Web 2.0 [10] is widely used to denote the technologies, applications, and business models that underlie success stories such as YouTube (http://www.youtube.com), Twitter (http://twitter.com), eBay (http://ebay.com), and Flickr (http://www.flickr.com). Web 2.0 websites are typically developed using technologies like AJAX [11], that gives the web pages an interaction model similar to desktop applications. Through Web 2.0 it is possible to provide powerful services (search, maps, product information, videos, etc) accessible through simple network protocols. They also provide ways for the clients to access a powerful programming platform (the ensemble of available services), a trend that proved to be revolutionary in terms of its impact on just about every aspect of the computer industry. By a similar approach, the term Science 2.0

can denote new approaches to research, *enabled by a quasi-ubiquitous Internet and Internet-based protocols for discovering and accessing services* [12]. Scientific communities like astronomy, have already demonstrated the potential of such approaches, via virtual observatories that provide online access to digital sky surveys and that have enabled both new discoveries and new approaches to education [13].

What is new and empowering with Science 2.0 is not only the fact the data is online, but the fact nowadays already exists enough uniformity in access protocols, and sufficient server-side computing power, to support access to services not only by people but by programs. One example of this kind of access of services by programs, is through the usage of web services using BPEL (Business Process Execution Language) [14]. BPEL is an executable language for specifying interactions with web services, allowing the execution of processes in a uniform way, that import or export information exclusively using web services.

Due to the advantages described above, we see an explosion in data access, as scientists write programs that process large quantities of data automatically. Increasingly, scientists can also publish useful programs as services. The services are published as service catalogs that list the available services and encourage the resulting rapid expansion in the scope and power of computational tools [12].

Although Science 2.0 provides great potentialities to the scientific community, it also raises many challenging methodological, sociological, and technical issues, that must be carefully analyzed before implementing a system that provides online scientific services to great scientific, large-scale projects like LHC [5], or other peta-byte scale projects [15].

2.1.1. The Grid concept and its role in modern science

In the latest years we have witnessed big advances in modern science and because of these advances, the methods and techniques used, have become more and more dependent on computers. In various fields of science, like physics, biology, mathematics or astronomy, the use of computationally heavy algorithms and/ or large datasets is unavoidable.

The term “The Grid” appeared in the mid 1990s to a proposed distributed computing infrastructure for advanced science and engineering . This infrastructure main goal is to try to solve the coordinated resource sharing and problem solving, in a dynamic, multi-institutional virtual organization. The Grid technology appeared to complement the existing solutions of distributed technologies, and not to compete with them [16].

With the emergence of Grid computing, many organizations related with these fields of science have turned to the Grid, to improve computational throughput, execution times of distributed algorithms and store and share data, using a secure environment provided by the Grid itself. Grid computing turns its focus on large-scale resource sharing, innovative applications, and, in some cases, high-performance orientation.

Although Grid computing provides many advantages in terms of computing, it inherits the complexities and threats of massive distributed systems. The use of grids involves many issues, like the protection of stored data, or the democratic use of available resources. Grid computing turns its focus on large-scale resource sharing, innovative applications, and, in some cases, high-performance orientation.

Grid can provide us with an incredible amount of resources. The access to these resources by users has to be made in a highly controlled environment. Assuming we have a single Grid were groups of users running experiments related with high energy physics, bioinformatics or earth sciences, each group will have different software needs i.e. some might need to run data intensive processes during a small period of time while others only need access to large distributed sets of data.

The real and specific problem that underlies the Grid concept *is coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations* [16]. The sharing that we are concerned with is not primarily file exchange but rather direct access to computers, software, data, and other resources like scientific instruments or sensors, since

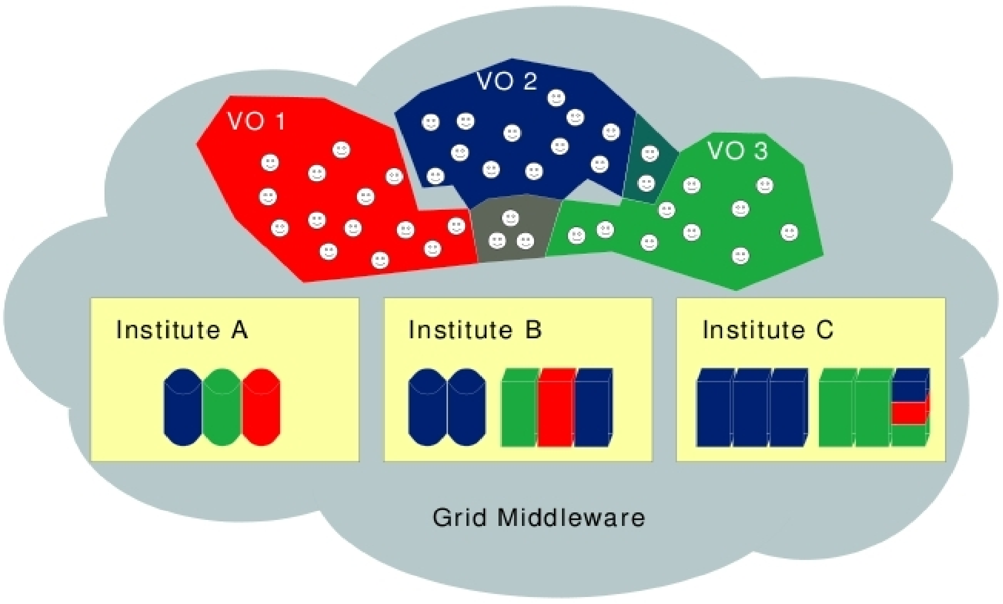


Figure 2.1: Virtual Organizations.

these resources are required by a range of collaborative problem-solving and resource brokering strategies emerging in industry, science, and engineering. This sharing must be, necessarily, highly controlled, with resource providers and consumers defining clearly what is shared, who is allowed to share, and the conditions under which sharing occurs. A set of individuals and/or institutions defined by such sharing rules form a Virtual Organization (VO) [16]. Good examples of VO's are: medical researchers working on the cure or investigation of a specific disease; members of an industrial consortium bidding on a new aircraft; a crisis management team and the databases and simulation systems that they use to plan a response to an emergency situation; and members of a large, international, multiyear high-energy physics collaboration. Each of these examples represents an approach to computing and problem solving based on collaboration in computation and data-rich environments (Figure 2.1).

As the examples show, VO's can vary in their purpose, scope, size, duration, structure, community, and sociology, but all of them share a common set of concerns and requirements:

- The need for highly flexible sharing relationships, ranging from client server to peer-to-peer;
- Sophisticated and precise levels of control over how shared resources are used, including fine-grained and multi-stakeholder access control, delegation, and application of local and global policies;
- Sharing various resources, ranging from programs, files, and data to computers, sensors, and networks;
- Diverse usage modes, ranging from single user to multi-user and from performance sensitive to cost sensitive and hence embracing issues of quality of service, scheduling, co-allocation, and accounting.

Regular distributed technologies do not address the concerns and requirements listed above, because they do not accommodate the range of resource types or do not provide the flexibility and control on sharing relationships needed to establish VO's. It is here that Grid technologies enter the scene. In the last few years, research and development efforts within the Grid community have produced protocols, services, and tools that address precisely the challenges that arise when we seek to build scalable VO's.

2.1.2. The Grid architecture

Since the appearance of Grid computing in the mid 1990's, the evolutions in its technology has been constant. In 1997 the GT (Globus Toolkit), an open source project was considered as the standard for Grid computing. With the start of a increased interest in Grids, new projects appeared and the Grid community started to search for standards to guarantee that the interoperability of the existing and future projects. In 2000 several existing Grid Forums merged to create the GGF (Global Grid Forum) that became a standards body. Finally in 2002 the OGSA (Open Grid Services Architecture) appeared as the true community standard for Grid infrastructures [17,18]. OGSA is a reference architecture in the area of Grid computing and defines a set of standards for Grid infrastructures. Some of the existing Grid infrastructures have architectures that implement or evolve OGSA, but always maintaining a Service Oriented Architecture (SOA) [19].

Most of all solutions follow a basic three tier layered architecture (Figure 2.2). From the architecture point of view, the Grid provides an abstraction layer to all the resources available to the user that wants to access the services (run an application or store data) (Figure 2.3). The group of services that compose this abstraction layer is called the Grid middleware. From a user perspective the middleware provides a set of API (Application Programming Interface) and CLI (Command Line Interface) that allow the access to several services.

At user level (top layer), it is possible to access scheduling and brokering services that allow the request and allocation of one or more resources. Other services like monitoring and diagnostic help prevent problems related with security, failure and other problems that might affect the infrastructure resources.

The services that compose the Grid middleware have very specific functions and interact among each other to provide the capabilities of the infrastructure to the user. Some services allow resource discovery and provide management mechanisms that are specific to the resource type. For instance, a computational resource should provide the mechanisms to start, monitor and retrieve results of an application that runs on the Grid, and should also (optionally) provide advance reservation capabilities.

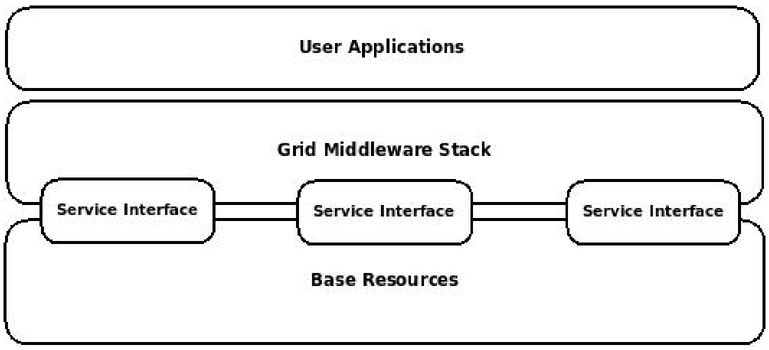


Figure 2.2: Grid Architecture.

Finally the bottom layer, consists of the resources that can vary from simple data to software or scientific instruments that can be owned by different organizations. This layer must be able to constrain the access to these resources to authorized users only. For instance, in the case of a storage resource they need to provide, for example, mechanisms for data movement and replication [17].

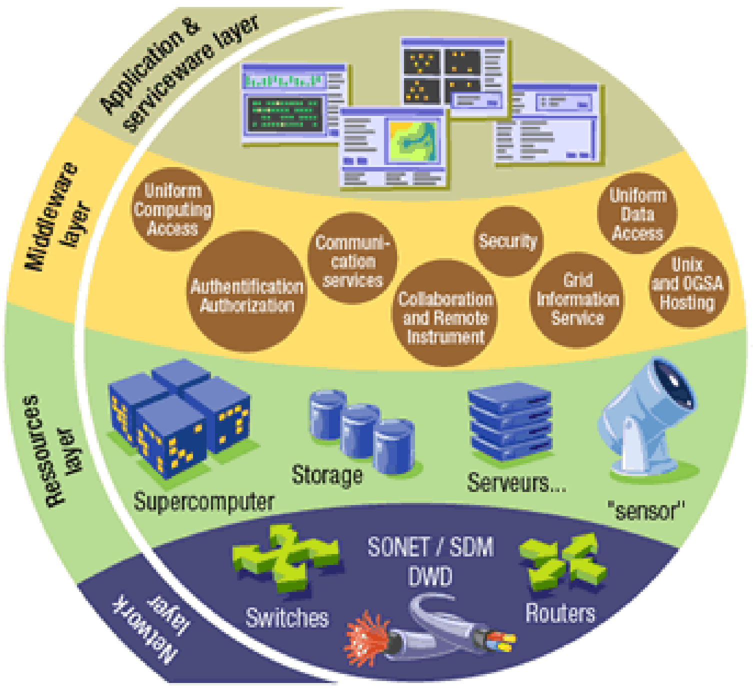


Figure 2.3: Detailed Grid architecture - In GridCafé website:
http://www.gridcafe.org/version1/gridatwork/architecture.html

2.1.3. Using the Grid

The access to these resources has to be necessarily highly controlled, and to enable this, the Grid VO concept (Virtual Organization) was created to identify a particular group of users that have access to a specific set of resources [20]. This kind of controlled environment, implies that each user must be authenticated. It is important that users authenticate only once when they are accessing multiple remote resources. For this to be possible there must exist methods that allow the users authentication to be delegated between services or resources [17]. For a user to be allowed to access a Grid, he/she must obtain a security certificate, from a proper certificate authority. Nowadays the majority of well known Grid middlewares use X.509 certificates [21] to authenticate the users, and also all the resources (machines, instrumentation, etc.). After obtaining the certificate, the user must be associated with one or more VO's to create the Grid proxy, and be able to access the available services.

Another important aspect of a distributed shared environment is the accounting. Besides giving access to their resources, organizations must be able to measure the amount of resources that were used by, for instance, a specific VO. This is quite difficult due to the heterogeneity of the resources presented in a Grid infrastructure.

The applications that run in these type of infrastructures have special requirements. We can divide them into five major classes: distributed supercomputing, high throughput, on-demand, data intensive and collaborative [4].

The aspects presented above are the technical and theoretical details, necessary for the usage of a Grid environment. In the real world, Grid nodes are scattered all around the world and are normally used by developers, administrators and end-users like scientists. The access to the major Grid infrastructures like EGEE or EELA isn't available to everyone, and only the duly authorized persons can access these kinds of infrastructures. In order to access these Grids, the new users must obtain the security certificate, generated by the proper CA (Certification Authority). The users must also be associated with at least one VO. The process to join one Grid infrastructure normally takes from a few days to one or two weeks.

2.2. Grid enabling technologies

In this section we present the major middleware technologies being used nowadays, with special emphasis on the data repositories technologies, as well as some known projects being developed on top of the gLite middleware to provide an easier Grid interface to both developers and end-users.

2.2.1. Grid middleware

The Grid infrastructure offers a transparent access to computing power in addition to an aggregation of distributed storage and heterogeneous resources. It is important that the various members of a VO can transparently use all the resources available (data, sensors, computing nodes, etc.) in collaborative terms. To achieve this interoperability, several implementations of standards-based architecture commonly named as Grid middleware are used. The Grid middleware manages and controls physical and logical resources that can be geographically separated. Examples of Grid middleware are gLite, Globus Toolkit (GT) and UNICORE (UNiform Interface to Computing REsources).

Grid projects rely on one or more grid middleware solutions from using only a specific middleware like GT to creating their own middleware. As we can see in Table 2.1 that presents the middleware stacks used by some of the largest Grid projects in the world, the diversity is high.

Projects	**Middleware Stacks**
EGEE	gLite middleware uses components from several Grid projects namely EGEE, EDG, EDT, INFN-GRID, GT and Condor
D-Grid	GT, UNICORE, gLite, dCache, SRB, OGSA-DAI, GridSphere, GAT, VOMS and Shibboleth
NAREGI	NAREGI middleware, GT 4.0.1, GSI and WS-GRAM
Open Science Grid	VDT is based in GT, Condor
UK e-Science	2001-2003 – GT, Condor, SRB 2003-Present – GT, OGSA-DAI, Web Services
TeraGrid	GT: GRAM, MDS, GridFTP, TGCP, RLS and MyProxy
ChinaGrid	CGSP (ChinaGrid Supporting Platform) based on GT

Table 2.1: Main middleware Stacks

Globus Toolkit

GT (Globus Toolkit) is an open source middleware that is being developed since the mid 1990s. It is presently being developed by a large community of organizations and individual users named *Globus Alliance*. The GT 4 provides a set of OGSA capabilities based on WSRF (Web Services Resource Framework). It is important to understand that GT services can be used to solve simple distributed problems but they are generally used in conjunction with other services for more complex situations [22].

Nowadays, GT services are being used in several Grid projects like EGEE, NAREGI or Tera-Grid (see Table 2.1). These services address Grid specific concerns like execution management, data access and transfer, replica management, monitoring and discovery, credential and instrument management. Most of the services are implemented using Java WS. Besides these services, there are also containers to host user-developed services written in Python, C or Java, providing them with a common interface and mechanisms that allow, for example, management and discovery functions. Client programs (written in Python, C or Java) can use GT4 and user-developed services through a set of client libraries [23].

To run a specific task the service used is the GRAM (Grid Resource Allocation Manager). This service offers a WS interface that permits initiating, managing and monitoring the execution of tasks. It also allows the user to select some parameters related with execution such as the number and types of resources used and the data that is needed for the execution or that is going to be retrieved after the execution ends. It is also used in other scenarios like service deployment and management where it controls the execution of services and resources consumption [23].

GT also provides data management services. These services can be used individually or in conjunction depending of the requirements. The GT data management services are:

- GT GridFTP implementation – Optimized for reliable and secure, data transfer on high bandwidth networks;
- RFT (Reliable File Transfer) service – Manages multiple GridFTP transfers;
- RLS (Replica Location Service) service – Decentralized service that manages the information and location of replicated files;
- DRS (Data Replication Service) service – Uses GridFTP and RLS to manage data replication.
- OGSA-DAI (OGSA–Data Access and Integration) tools – Provide data access and integration using relational and XML (eXtensible Markup Language) data.

The monitor and discovery in GT4 can be done through standardized mechanisms, based in the WSRF and WS-Notification implementations, built in every GT4 service. These provide the access to resources properties, based in XML. The information is collected and published by the aggregator service Index. Another service, named Trigger, collects information in an event-based. All of this information can be viewed through the WebMDS service [23].

GT4 security components are also highly based on standards. Despite the fact that it supports several security protocols (The protocols supported are: Message level with X.509 credentials - WS-Security-compliant implementation; or username/password (WS-I Base Security Profile compliant) and Transport-level security with X.509 credentials), by default it is used the Transport-Level based security with X.509 public key credentials because it is faster. Besides the GT4 components described above, other services are used in conjunction to support credential management (e.g. MyProxy, PERMIS (PrivilEge and Role Management Infrastructure Standards validation) and VOMS (Virtual Organization Management System)) [23].

UNICORE

UNICORE is another open-source Grid middleware technology that provides seamless, secure and intuitive access to distributed Grid resources such as supercomputers or cluster systems and information stored in databases. The development of UNICORE started in 1997 and was developed in two projects funded by the German ministry for education and research. In various European-funded projects UNICORE has evolved to a full-grown and well-tested Grid middleware technology, and is used in daily production at several supercomputer centers world-wide. Beyond this production usage, UNICORE serves as a solid basis in many European and international research projects [24].

UNICORE was fully written in Java and has an architecture based on web services (WSRF) to establish the communication between its services, under SSL (Secure Sockets Layer) secure connections. The architecture of UNICORE consists of three layers, namely user, server, and target system tier. The user tier is represented by the UNICORE Client, a Graphical User Interface (GUI) that exploits all services offered by the underlying server tier. Abstract Job Objects (AJO), the implementation of UNICORE's job model concept, are used to communicate with the server tier. An AJO contains platform and site independent descriptions of computational and data related tasks, resource information, and workflow specifications. The sending and receiving of AJOs and attached files within UNICORE is managed by the UNICORE Protocol Layer (UPL) that is placed on top of the Secure Socket Layer (SSL) protocol. The user of an UNICORE Grid does not need to know how these protocols are implemented, as the UNICORE Client assists the user in creating complex, interdependent jobs. For more experienced users a Command Line Interface (CLI) is also available. Both, the UNICORE Client and CLI, provide the functionalities to create and monitor jobs that can be executed on any UNICORE site (Usite) without requiring any modifications, including data management functions like import, export, or transfer of files from one target system to another. In addition, the

UNICORE plugin technology allows the creation of application-specific interfaces inside the UNICORE client.

gLite

The gLite middleware has been introduced by the EGEE project as a result of contributions from many other projects. The EGEE project, is the Europe's flagship Research Infrastructure Grid project funded by the EC (European Commission), that aims at the creation of a Grid infrastructure to support e-Science, and it is the world's largest Grid infrastructure of its kind. This project includes members from various scientific areas, like biomedicine, physics, computational chemistry and Life Sciences, and is supported by more than 240 institutions from 48 countries world-wide with more than 68.000 CPU's. Due to the large amount of people, data and resources involved in this project, one of its main goals is the development of a Grid middleware that can support this kind of large infrastructure with so many scientific fields [25].

The gLite middleware is based on a wide number of Grid projects like DataGrid, DataTag, Globus Toolkit, GriPhyN, iVDGL (International Virtual Data Grid Laboratory), EGEE and LCG (LHC Computing Grid). This middleware provides high level services that enable the scheduling and analysis of computational jobs, data storage, replication and retrieval, and information gathering about the infrastructure. All these service share a common security framework.

The gLite services are usually grouped as access, security, information and monitoring, job management and data services (Figure 2.4).

gLite Access and Security

The access to a Grid infrastructure using gLite is done through the UI (User Interface). The UI, is usually a dedicated machine, installed with the set of CLIs and APIs that provide access to services available in the Grid.

The security services provide the tools for authorization, authentication and auditing. These services are responsible to control the access to Grid resources, and provide information for analysis in case of security threats. Only authorized users can access the Grid resources, and only a defined set of the resources is available to the user, normally bounded by the VO.

Each user, Grid resource or service is identified by a digital X.509 certificate. These certificates are composed by a public and private key. These certificates are used to generate and sign a temporary certificate (called temporary Grid proxy), that in turn is used to authenticate the user, each time he/she wants to access Grid services or resources. During a Grid operation there are several services that might be used, and that request the user's authentication with the temporary proxy. To avoid the user to sign in every time the proxy is requested, there have been implemented methods that allow the proxy to be delegated between services. After the user submits its request he cannot cancel the proxy, so usually user proxies have a short life time, normally 12 hours.

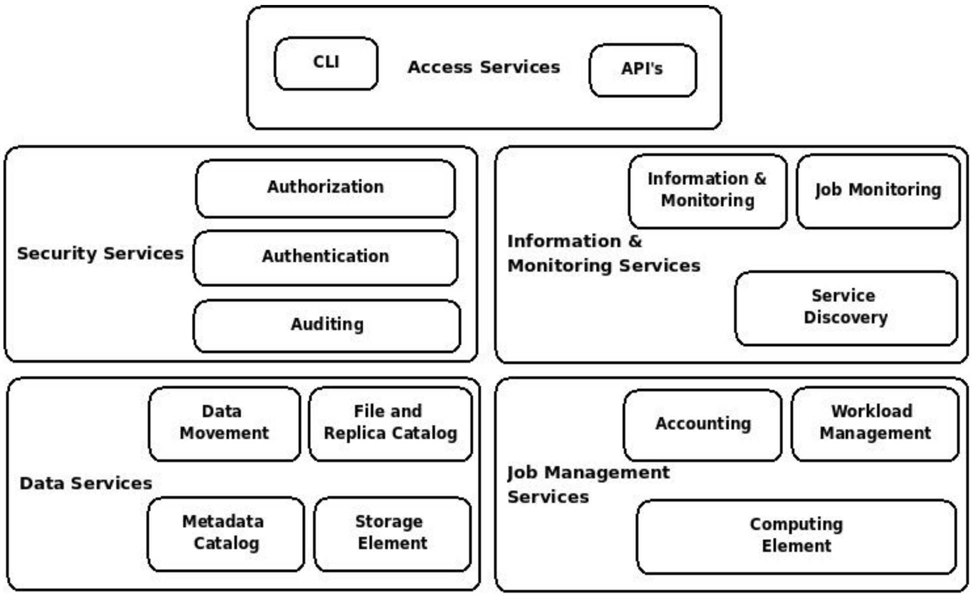


Figure 2.4: gLite Services

To manage the user's permissions inside each VO, gLite uses the VOMS service. This service allows a more controlled access to resources. When a user creates a proxy, the VOMS service is contacted and returns a signed AC (Attribute Certificate) that contains the information relative to that user. Then an extension is added to the generated proxy file where it is described the user's permissions inside each VO. The authentication in the resource can be done through two mechanisms. The first compares the subject name contained in the proxy with a local Grid-mapfile that maps users to local accounts. The second method relies on the VOMS and the LCAS/LCMAPS mechanism and allows a more detailed control of the user “rights”.

Due to the distributed nature of Grid infrastructures, the issuing of certificates besides being a highly secure and controlled process, must be widely available to users world wide. The International Grid Trust Federation (IGTF) is the international organization, composed by the EuGridPMA (European Grid Policy Management Authority), APGridPMA (Asia Pacific Grid Policy Management Authority) and TAGPMA (The Americas Grid Policy

Management Authority) that aims at the establishment and maintenance of a global trust relationship between its members. Inside each region we have several CA (Certification authority), one for each country, that are responsible for providing the X.509 certificates in the corresponding country. But before this can be done a procedure must be followed to guarantee, for example, the identity of a user. This process is usually done by a RA (Registration Authority) that are normally distributed throughout each country.

In gLite, the access to computer and data resources, is provided by two elements called CE (Computing Element) and SE (Storage Element) respectively. The CE (Computing Element) enables an abstraction of the computing resources available in a site. This enables the use of different computing resources, from batch queues of clusters to simple workstations, but providing a common interface for job management and information gathering. It includes the GG (Grid Gate), LRMS (Local Resource Managing System) and a collection of WN (Worker Node). The GG is responsible for accepting and dispatching jobs to the WN through the LRMS, acting as an interface between the CE and the rest of Grid services. The jobs are executed in the WNs of a site, that provide most of the CLI command and API's also available in the UI.

gLite Storage

The Storage Element (SE) provides the virtualization of a storage resource (ranging from simple disk servers to tape-based storage) much as the CE does for computational resources. There are currently three types of storage elements: DPM (Disk Pool Manager), dCache and CASTOR (CERN Advanced STORage manager). Every one of these SE's can be accessed through a common interface – the SRM (Storage Resource Manager).

The protocol used for file transferring is the gsiftp (GridFTP). For file I/O operations the protocols used are GSIDCAP (GSI dCache Access Protocol) and RFIO (Remote File Input/Output protocol) depending of the type of SE used. RFIO has an insecure and a secure version (gsirfio).

gLite Job Management

Job management is a fundamental part in Grid computing, because it allows job monitoring, accounting, scheduling and execution according to the availability of resources. gLite has several services that are responsible for job management like the already discussed CE, the WMS and accounting services.

To control and monitor the job submission to the CE, the gLite has a meta-scheduler named WMS (Workload Manager Service). Besides managing jobs execution, it also monitors job state through the LB (Logging and Bookkeeping) service. The WMS is composed by several components that are responsible for matchmaking resources and jobs, submit, cancel, monitor and keep a record of the state of each job (LB). The access to the WMS can be made through the WMProxy service that together with the LB provide a WS based interface with similar functionalities, enabling the user to run job execution related commands (e.g. job submission, cancellation or monitoring). Another important component of the WMS is the DAGMan (DAG Manager), a meta-scheduler responsible for managing groups of jobs with special dependencies among each other. The WMS is also responsible for the proxy renewal using the MyProxy service. The node where the WMS runs is called the RB (Resource Broker).

gLite JDL

Before submitting a job, a user has to create a file that describes among other things the type of job, rank, file to execute, input data, output data and requirements. These parameters are described using the JDL (Job Description Language) [26]. One of the parameters that can be defined is the Input and Output Sandbox that correspond to the location of the files that will be transferred to the WN where the job will be executed and the files that will be retrieved from the WN. The WMS is responsible for the transfer of these files that can be local or remote. Other parameters are related with the target resource to be chosen upon the job submission, like the Requirements and Rank. The first allows a user to define the requirements a resource must fulfill like the LRMS type or software installed. Rank allows the user to define rules so that the WMS is able to decide among the

```
[
      Type = "job";
      JobType = "normal";
      RetryCount = 3;
      ShallowRetryCount = 10;
      Requirements = other.GlueCEStateStatus == "Production";
      StdOutput = "std.out";
      StdError = "std.err";
      Rank = other.GlueCEStateFreeCPUs;
      Executable = "IGFInputData_davidp.sh";
      InputSandbox = {"/usr/GridApps/sumarize","/tmp/IGFData.sh"};
      OutputSandbox = {"std.out","std.err","v50_summarize.out"};
      DataRequirements = {
        [
        DataCatalogType = "DLI";
        InputData = {"guid:b0040940-f6b5-4656-9f32f58fc7dbb8496",
                     "guid:a2d36082-6ddf-4553-bade-a4b34c183619"};
        ]
      };
      DataAccessProtocol = {"gsiftp","rfio"};

]
```

Figure 2.5: JDL file example.

resources that satisfy the Requirements. An example could simply be the number of CPU's available for the submitted job [27].

The JDL also allows to define special types of jobs with specific attributes [26]. The different types available are:

- Job – Simple job that can be one of the following subtypes:

  – Normal – Simple batch job;

  – Interactive – Simple job with its standard streams forwarded to submitting client;

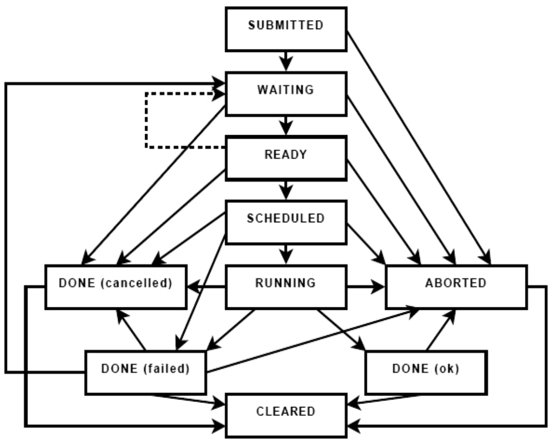
  – MPICH – Parallel application that uses MPICH-P4 implementation;

  – Partitionable – Job that can be divided into smaller independent jobs ;

  – Checkpointable – This type of job allows the execution of the program to be paused by defining pause flags in its code, and if necessary resume the job on those defined positions without losing the execution that was performed before the flags.

  – Parametric – Job that has parametric attributes that can be defined with several values. A job instance is created for each value of each attribute.
- DAG – Job with dependencies among each other, described by a DAG (Directed Acyclic Graph);
- Collection – Group of independent jobs.

A new approach to the job description is the JSDL (Job Submission Description Language) that uses an XML-based language to describe jobs. JSDL was proposed by the OGF were it is being developed by the JSDL-WG (JSDL Working Group). Currently the JSDL is already supported by the gLite WMS.



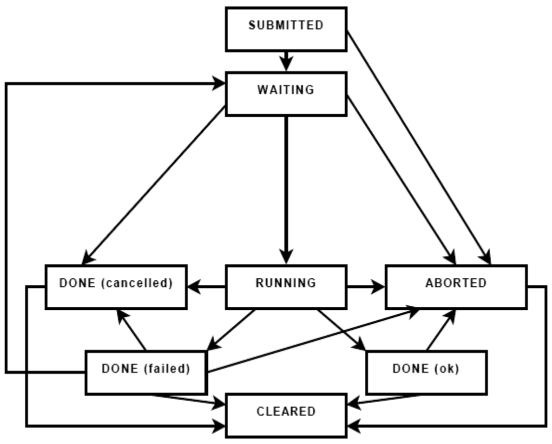


Figure 2.6: States of a normal (on the left) and DAG (on the right) job during execution.

A user can get information about the job state through the LB service (see Table 2.2). The transition states that a job can go through during its execution are shown in Figure 2.6.

State	Description
Submitted	User submits a job in the UI
Waiting	Job is accessed by the WMS and is waiting for resource allocation
Ready	A resource as been allocated for the job
Scheduled	LRMS has accepted the job and is in queue
Running	Job is sent to the WN as is being executed
Done	The job has finished its execution and its output is available
Cleared	The output has been transferred to the user and the job has been freed
Aborted	Job was aborted by the system
Canceled	User canceled the job
Unknown	Status cannot be determined
Purged	The job has been deleted from the LB server

Table 2.2: Job States

gLite Information System and Monitoring

All the information regarding Grid resources and their status is managed by the IS (Information System). gLite has two different IS where much of the data published conforms with the GLUE (*Grid Laboratory Uniform Environment*) schema, a common information model for resource discovery and monitoring [28]. The two information models being used are R-GMA (Relational GMA) and MDS (Monitoring and Discovery System). The first is used to publish accounting, monitoring and user related information while the MDS is used for resource discovery and status.

The MDS uses the OpenLDAP (Open source implementation of LDAP) information model to implement the GLUE schema. It does not allow secure access. MDS architecture is based on Information Providers installed in each site that gather static and dynamic information relative to that site that is them published by the GRIS (Grid Resource Information Server). The GRIS is an LDAP (Lightweight Directory Access Protocol) server that is normally installed locally in the resource. Then in each site a GIIS (Grid Index Information Server) collects the information published by the existing GRIS and republishes this information to a higher level GIIS. The information gathered by the GIIS is stored in a BDII (Berkeley Database Information Index).

The R-GMA is based on the GMA (Grid Monitoring Architecture) initially proposed by the GGF. R-GMA architecture is based in three main components:

- *Producers* are responsible for gathering information and informing the *Registry* about the information they are publishing and how it is accessed;
- *Consumers* contact the *Registry* to discover what *Producers* publish the target data and how it can be retrieved. Them they contact directly the *Producers* for the target data;
- *Registry* contains the information about the data and structure that each *Producer* has.

The information presented by the R-GMA is in the form of virtual database of virtual tables which structure is defined by the *Schema*. The R-GMA system is defined by the *Registry* and *Schema*.

gLite has two services, APEL (Accounting Processor for Event Logs) and DGAS (Distributed Grid Accounting System), that are responsible for accounting.

2.2.2. Robust storage on the Grid

Since the main focus of this work is on storage and sharing of medical images on the Grid, we decided to explore in more depth middleware components related with storage and metadata.

In order to make efficient use of the large amount of available disk space, spread across all the resources in a Grid infrastructure, the middleware technologies must have robust systems to deal with all the resources and the large amounts of data and metadata that need to be stored. In this chapter we will present some of the most important middleware "sub-systems" that deal with storage on the Grid.

glite DMS

The Data Management System (DMS) is an essential part of the Grid middleware for enabling users and applications to handle their data without having to worry about the complex details of the computing environment, and enabling them to look at the distributed data as it was a single pool of files. The main capabilities provided by the DMS are locating, accessing and moving files stored in the SEs.

From the functional point of view DMS offers two fundamental macro features: File Management and Metadata Management. The first one involves the typical file system functionalities (save file, copy file, read file, list files), placing functionalities (replicate file, transfer file) and security services (ACL for files, users roles). The second one involves database schema virtualization (metadata handling, intelligent search), file cataloging and file searching.

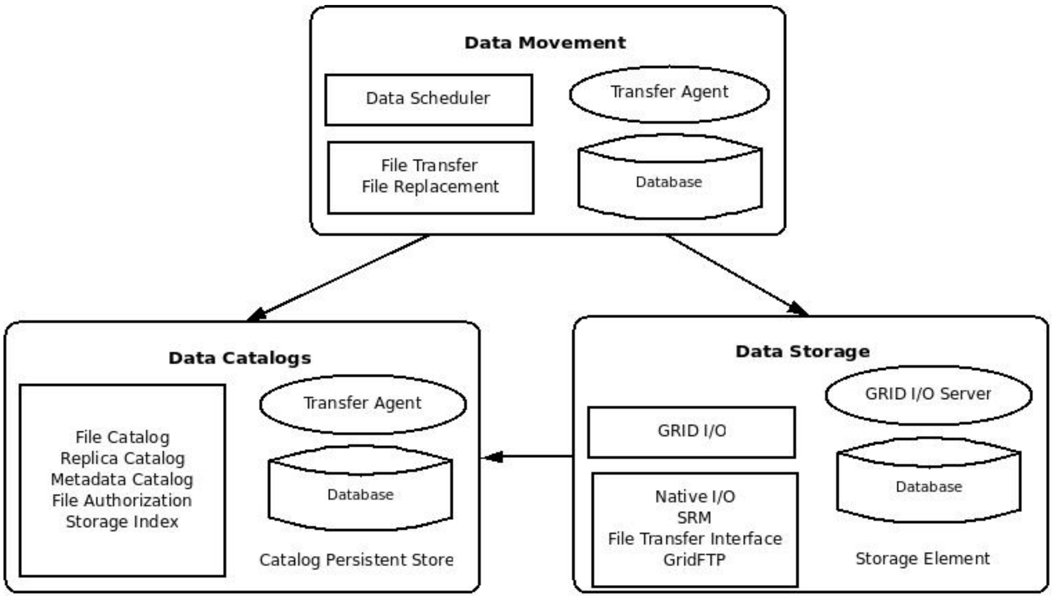


Figure 2.7: DMS Services architecture view.

From the users perspective, the DMS provides all the usual data operations: uploading/downloading files, creating file/directories, renaming file/directories, deleting file/directories, moving file, listing directories, creating symbolic links. Figure 2.7 shows the DMS subsystems and their interrelations. The Data Storage is the Grid service that takes care of data manipulation in order to make users and/or applications to access and manage their own files. The Data Movement service enables both the Grid services themselves and/or any clients (users and/or applications) to move files from/to a site, (a site is the smallest, complete and auto consistent hardware/software resource organization within a Grid infrastructure, then, think of Grid as a union of one or more sites). The Data Catalogs service has in charge to keep trace about files location into the distributed file system and store any kind of information about them.

In terms of file identification, Grid files can be identified by a GUID (Grid Unique Identifier), LFN (Logical File Name), SURL (Storage URL) or TURL (Transport URL).

The GUID is unique identifier, based on the UUID (Universally Unique Identifier) standard, that identifies every single file. Every file stored on the Grid (registered in the file catalog, and stored in a SE) has a corresponding GUID (e.g. guid:81adf875-ccd7-44fa-89b5-0fef8ae9cd34).

LFN provides files with a human readable name similar to the ones found in computers file systems (e.g. lfn:///grid/dteam/Data/ex.nii.gz).

SURL or PFN (Physical File Name) is used to identify files or replicas that are in a specific SE. The name has a prefix (srm or sfn) that identifies the SE with or without an SRM interface.

In the case of SEs that do not have an SRM-based interface, the SURL has a specific structure that identifies the host and the physical location of the file (e.g. sfn://xst.ieeta.pt/data/dteam /grid/dteam/Data/ex.nii.gz).

On the other side, in SRM-based SE's the SURL can or cannot include the files physical location. Normally, an SRM-based SE uses virtual file systems to map files (e.g. srm://xst.ieeta.pt/dpm /ieeta.pt/home/dteam/grid/dteam/Data/ex.nii.gz).

TURLs provide the necessary information for accessing files in SE like hostname, path, protocol and port. Due to the fact that TURLs are obtained dynamically, users should be aware that they may change over time. In an SE there can be more than a TURL for each file, because the SE can have multiple access protocols and the SE might have several copies of each file for load-balancing.

The data management can be made through the use of CLIs or APIs available in the WN and UI.

LFC (LCG File Catalogue) is the file catalogue adopted by gLite. This service provides several file related functions like mapping between the GUID, SURL and LFN (Figure 2.8), system (e.g. file size and checksum) and user metadata and replicas information. It also allows a user to attach an ACL to a file.

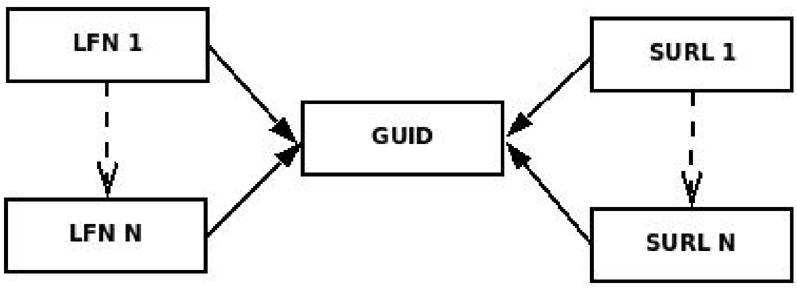


Figure 2.8: Relation between the LFN, SURL and GUID

Although LFC provides some basic metadata capabilities, there is a need for more complex metadata. To overcome this problem gLite chose as its official metadata catalogue, the AMGA (ARDA Metadata Catalogue) [29].

The Metadata support allows the user to associate descriptive attributes with files (metadata schema), publish these attributes on catalogs and finally make catalogs available to end users and client application. The user is able to define a schema as a set of attributes associated with an entry. An attribute is composed by a name, a data type and the respective value. An entry is an identifier for the instance of the metadata schema, meaning the values of its attributes. Finally, a metadata collection can be defined as a set of entries. For instance if you have a collection called *Images* you can define a schema containing the attributes *width* and *height* of type float. Each of the entries stored in the collection (typically one entry corresponds to one file stored on the Grid), must have the attributes *width* and *height* defined with the respective values. This way the files stored on the Grid can have custom metadata associated.

OGSA-DAI

The OGSA-DAI project began in 2002. OGSA-DAI main goal is to develop an effective solution to the data management challenge and in particular to data access and integration problems. This middleware solution allows resources, such as relational or XML databases to be accessed via the Grid [30].

OGSA-DAI provides a simple way of wrapping data resources, such as databases and files, with a web service interface which consume data centric workflows. These workflows encapsulate multiple client-services interaction into a single one, move computation close to the data and reduce the amount of data movement necessary to achieve a given end.

The out-of-the-box base functionality provided by OGSA-DAI enables powerful access and integration scenarios to be constructed. The OGSA-DAI architecture is modular, customizable and extensible allowing OGSA-DAI to meet the data access and integration requirements of other middleware projects.

2.2.3. Grid community and standards

Due to the large heterogeneity of resources, people and software involved in the large Grid infrastructures, there is the need for the definition of standards and the creation of communities for a better interoperability of all the resources involved in this kind of massive infrastructures.

In 1998 the United States, European and Asia-Pacific Grid Forums were established by the Grid community. In the year 2000 they merged to create the GGF that became a standards body. The main goal of this organization was to create new standards to guarantee the interoperability of the existing and future Grid projects [31]. A few years later, in 2002, the Open Grid Services Architecture (OGSA) finally appeared as the true community standard for Grid infrastructures [17,18]. In 2004, due to the growing potential and importance of Grid technology, several industry leaders decided to form the EGA (Enterprise Grid Alliance), a consortium created to accelerate the development of enterprise Grid solutions and deployment of Grid computing [32]. In 2006 the OGF (Open Grid Forum) was formed from the merge between the GGF and the EGA in an attempt to focus the development of standards.

In this context it is worth mentioning that one of the major accomplishments of the OGF was the OGSA. OGSA consists of a service-oriented architecture built on top of WS (Web Service) standards, and addresses some aspects that are relevant to Grid services such as service creation, life-time and several others. Initially it was supported by a core set of interfaces called OGSI (Open Grid Service interface) [33], that was later abandoned to be replaced by the WSRF (WS-Resource Framework) [34]. The WSRF is a set of WS specifications being developed by the OASIS (Organization for the Advancement of Structured Information Standards) [35] that describes how to implement OGSA capabilities using WS [19]. The GT 4 middleware [23] is an example of a software providing some OGSA capabilities based on WSRF.

2.2.4. Grid-enabled portals development

In this section we include a brief introduction to projects directly related to this work, that involve the development of technologies or frameworks for the construction of Grid-enabled web portals.

P-GRADE

The P-GRADE project [8] main goal, is to provide a high-level graphical environment to develop parallel applications transparently both for parallel systems and the Grid, so that the users do not need to learn programming methodologies for different parallel and distributed platforms. It supports the interactive execution of parallel programs as well as the creation of a Condor, Condor-G or Globus job to execute parallel programs in the Grid. With P-GRADE, the user can generate either PVM (Parallel Virtual Machine) or MPI code according to the underlying Grid where the parallel application should be executed. P-GRADE supports workflow definition and coordinated multi-job execution for the Grid.

Such workflow management can provide parallel execution at both inter-job and intra-job level. Automatic checkpoint mechanism for parallel programs supports the migration of parallel jobs inside the workflow providing a fault-tolerant workflow execution mechanism.

Vine toolkit

The Vine toolkit [36] is a modular, extensible Java library that offers developers an easy-to-use, high-level Application Programmer Interface (API) for Grid-enabling applications. Vine can be deployed for use in desktop, Java Web Start, Java Servlet 2.3 and Java Portlet 1.0 environments with ease. One of Vine's main features and advantages is the support for a wide array of middleware (like gLite, GT 4 or UNICORE) and third-party services, abstracting the user from the problems and complexities of typical Grid middleware technologies.

2.2.5. Enabling technologies for Grid repositories

In every Grid infrastructure, a huge amount of distributed storage resources is available to store the user's data. However, if you don't associate specific metadata with the stored data, finding a specific file can become very difficult. In this section we are going to present some of the most important technologies used nowadays, when dealing with Grid storage, and more specifically using the gLite middleware.

GSAF

Developed by the Consorzio COMETA, GSAF is an object oriented programming framework written in Java, built with the purpose of providing a uniform programming interface for Data Grid oriented applications, working as a wrapper for the Grid data management system. GSAF was built to work on top of the gLite middleware, used by the EGEE Grid.

The *Grid* architecture respects the principles of SOA (Service Oriented Architecture), therefore the Grid Data Management subsystem is built by several Data Services (File Catalog, Metadata catalog, Disk Pool Name Server, GridFTP, etc), that are independent from each other, and can work in a stand alone mode. This diversification of services produces a consequent API's fragmentation, makes life much harder for the software engineers that have to develop Grid storage applications. This fragmentation also requires that engineers and developers have a solid skill on Grid programming and a great technical knowledge about the services details. To counter this fact, GSAF main philosophy, is to mask the complexity and the fragmentation of the middleware, hiding all APIs details and troubles, and exposing a unified interface closer to the developer's mind rather than the Grid programming details.

During the development of the project, several problems were encountered, related with the middleware software (gLite), because this is still a research project and it is not closed neither stable yet. The lack of available low level API's and the existing bugs in the gLite was also a problem in the development of this project [37].

gLibrary: Digital Asset Management System for the Grid

The gLibrary project was developed by a team at the INFN (Italy's National Institute for Nuclear Physics), with the purpose to create an extensible, robust, secure and easy to-use system, to handle digital assets widespread across a distributed *Grid* infrastructure [38]. gLibrary was developed on top of the *Grid* services of EGEE gLite middleware [39].

The system offers an intuitive web interface to browse the available entries, and search for the stored assets. According to the proposed architecture, the assets are hierarchically organized by types. Upon the registration /upload process the assets are associated with a proper type. The asset can be registered with a type that inherits from another, automatically acquiring the attributes of the parent type. Each type has its own set of attributes, and can be defined according to the users needs. The type's attributes can also be used to create search filters, so that the user can find the files easier. Once the asset has been found, the user can retrieve the file replica from one of the *Grid* Storage Elements into his/her own desktop.

To implement the proposed architecture, the AMGA Metadata Catalog takes a central role, being used as the repository for all asset's attributes, types and categories hierarchies, available libraries and physical file locations. It also uses the AMGA catalog authorization system, to provide the security for the stored information, ensuring that only the authorized users can access it. User requests, like browsing and searching, are transformed into SQL-like queries, and sent to the AMGA server, that searches its back-end database for the corresponding data. Benchmark studies on performance, demonstrated that AMGA's overhead is very low [40], and this guarantees short response times by the gLibrary interface.

In conclusion, gLibrary is a system that can be used for many different purposes. Thanks to its modular architecture, different communities can easily adopt this tool to build their own digital libraries defining types and categories according to their needs [38].

2.3. Life sciences as an application domain

Since this work was proposed by the SIAS group at IEETA, with the purpose to explore the possibility to extend the scope of the BING project [7] to use major Grid infrastructures, we think it is important to make a brief introduction to the current medical imaging state of the art, as well as the real application of Grid infrastructures to support e-Health projects fulfilling the so called HealthGrid concept.

2.3.1. Medical imaging overview

Medical imaging has taken an essential role in healthcare [41-43]. It is mainly used nowadays, to support clinical diagnosis namely by helping the assessment and progression of a disease in response to a specific treatment. Several medical imaging techniques are wide spread like ultrasound, radiography, CT (Computed Tomography), MRI (Magnetic Resonance Imaging) and nuclear medicine images [44]. Despite the proved value of medical imaging in the clinical environment, to explore its full potential implies some expertise and depends highly on medical education and training of clinical experts. [45].

Different imaging modalities are used in different types of analysis. For instance some are more suited for soft tissues and others offer a better trade off between spatial and time resolution. In some cases data from different types of imaging are being combined to allow a better interpretation of the data. This approach called multimodal imaging is used for example with High resolution Electroencephalogram (HR-EEG or EEG), Magnetic Resonance Imaging (MRI and fMRI), Spectroscopy (MRS) or Single Photon/Positron Emitting Tomography (SPECT/PET). In case of the EEG provides a very good temporal resolution while the MRI provides the high level spacial resolution. This is specially important in brain studies [46].

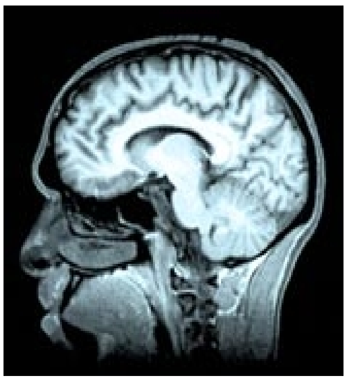


Figure 2.9: Image produced by a MRI brain scan.

A good example is the support to epilepsy surgery [47] where is extremely important for a surgeon to be able to identify critical areas of the brain (e.g. associated with movement, language and epilepsy) in order to remove epilepsy specific areas of the brain without interfering with other healthy brain functions. In this process a multimodal approach is the only solution relying in several non-invasive imaging techniques such magnetic resonance imaging (MRI) (Figure 2.9) to characterize the brain morphology and functional mapping supported in functional MRI that allows the establishment of relations between specific images and particular functions of the Human body [48].

2.3.2. HealthGrid: at the intersection of Grid and e-Health

Grid computing is being used in the scientific community as a key technology to solve large scientific challenges due to its ability to bring together distributed capabilities (large storage capability, computing resources, scientific instruments) and use them in an integrated virtual environment (e.g: [49]).

The emergence of Grids enabled the inter-disciplinary research in biomedical areas such as medical informatics, bioinformatics or system biology, creating new opportunities for research. An infrastructure that allows sharing heterogeneous and disperse medical relevant data, that can be used for processing and can be accessed by actors of healthcare in a secure manner, according to their authorization, is called a HealthGrid [50,51]. To support the development of HealthGrids it was created the HealthGrid initiative (http://community.healthgrid.org/).

The application of Grid computing technologies in life sciences can be generically referred as HealthGrids [52]. The HealthGrid vision stands for the use of Grid infrastructures for Medical Research, Healthcare and Life Sciences, which implies the availability of Grid services and the definition and adoption of international standards and interoperability mechanisms [50]. The HealthGrid concept defines a Grid that is able to manage, relate and process information from multiple levels (molecules, organ, tissue, individual, populations), as explored in the HealthGrid white paper [52].

In life sciences the Grid technology is specially important in addressing biological data complexity and in allowing the interoperability between the large number of databases that provide specific representations of biological data like Embrace [53]. In molecular biology, Grid also plays an important part in comparative data analysis, mandatory in most of molecular biology data analysis workflows. Also important, is the constant need for molecular biologists to access databases to retrieve information related with their research. Because of this it is important to make existing databases, accessible to biologists and provide the resources to analyse them.

Another area of research closely related with healthcare is medical research in imagiology. Since imagiology plays a key role in diagnosis, therapy planning and treatment follow-ups, the amount of data produced in hospitals is increasing every year [54]. Moreover in some countries this data must be accessible to patients and so hospitals are forced to keep archives for 7 to 20 years. In Europe it is estimated that the volume of data produced in hospitals is comparable with the one expected for CERN (European Organization for Nuclear Research) LHC (Large Hadron Collider) which is in the order of Peta Bytes per year [55,1].The amount of medical data available, if accessible, could improve epidemiology and pathology studies.

Grid technology appears as the ideal candidate to create medical federated storage and provide the resources necessary to analyse this data [52], because it provides the basic information platform for a reliable and dependable solution over distributed data sources (e.g. [56,57]), maintaining simultaneously the existing domains (either organizational, geographical), and access to processing intensive methods.

Nevertheless the use of Grid technologies in the creation of medical federated storage has special requirements. There are some major issues that need to be addressed before the HealthGrid vision can be fulfilled [58]:

- **Grid middleware**: The current middleware solutions, like gLite or UNICORE, are all still a “work in progress”, and therefore can be problematic to run healthcare applications on top of these platforms;
- **Deployment**: There exist several limitations to deploy grid nodes in healthcare facilities like hospitals. These limitations are the security requirements, the lack of friendly interfaces, the interoperability between Grid services and existing data management solutions already adopted and the difficulty in installing the grid nodes;
- **Standards**: Before the data can be shared there must be defined international standards and mechanisms that allow for example the anonymization and pseudonymization of the information;
- **Management**: The concept of VO is not flexible enough in managing healthgrids. It should be possible, for example, to define VO of VOs.

Regarding the specific work of this dissertation, it can be related with the HealthGrid concept, because it shares common goals with it. Our work is focused on the development of a Grid enabled web portal that will provide services to medical researchers, contributing to the improvement of medical research methods. This work, was also developed having in mind the possible integration with the GERES-med [49] and BING [7] projects being currently developed by the SIAS group. The goal is to provide a Grid "plug-in" for these projects, allowing the respective users to access the Grid seamlessly. Both projects propose the creation of medical repositories, with support for medical applications, ranging from simple to more complex workflows. The developed work could be a great asset, not only for the BING and GERES-med projects themselves, but also to other health specific Grids.

2.3.3. Selected HealthGrid repositories projects

Although there are special requirements in the implementation of healthgrids (see section 2.3.2.), specially the ones that deal with medical data, there exist several projects that are developing solutions to some of these problems. Examples are caBIG (cancer Biomedical Informatics Grid), BIRN (Biomedical Informatics Research Network), the SARSGrid (SARS Grid), and the Medical Data Manager project.

caBIG [3] is an initiative that wants to link researchers, physicians and patients in the cancer community to foster the research in the medical research area. This way they are able to share research results, information and foster the collaboration between researchers.

The BIRN [56], is an infrastructure that supports neuroimaging research. Its is divided in three testbeds that research specific areas in neuroimaging:

- Function BIRN: Develop tools and methods that solve problems associated with multi- site functional MRI;
- Morphometry BIRN: Analyze data from a large group of subjects, suffering from memory dysfunction or depression, provenient from different neuroimaging sites and study structural differences;
- Mouse BIRN: Use mice to study neurodegeneratives diseases. Analyze multi-scale structural, functional, genomic and gene expression data acquired from mice's brain.

The information technology infrastructure that enables the research in BIRN is managed and supported by the BIRN-CC (BIRN Coordinating Center).

The SARSGrid [59] is an AccessGrid based collaborative platform that was used, in 2003, to share and discuss SARS (Severe Acute Respiratory Syndrome) patient's medical data between healthcare professionals to diagnose, treat and monitor home and hospital quarantined SARS patients in Taiwan.

The Medical Data Manager project [60] was developed with the purpose to create a secure medical data management system that takes advantage of the advanced gLite data management services, developed in the context of the EGEE project [9], to fulfill the demanding needs of the medical community. The main goals are, to ensure medical data protection through strict data access control, anonymization and encryption, and to provide a grid Storage Resource Manager (SRM) interface to standard medical DICOM servers thereby enabling transparent access to medical data without interfering with medical practice. The multi-level access control provides the flexibility needed for implementing complex medical use-cases. Data anonymization prevents the exposure of most sensitive data to unauthorized users, and data encryption guarantees data protection even when they are stored at remote sites.

3. Supporting brain imaging research workflows

Brain Imaging (BI) is essential in the neuroscience research, which can be considered to be in the frontier between neurology, engineering and physics. Multimodal medical imaging techniques, such as Magnetic Resonance Imaging (MRI and fMRI) and Spectroscopy (MRS), Single Photon/Positron Emitting Tomography (SPECT/PET) among others, are emerging medical research tools that can provide valuable information for characterizing healthy and abnormal brain function namely in brain diseases such as epilepsy [61,62] or neurodegenerative diseases [63,64]. The problem is that using these multimodal medical imaging techniques, also generates large amounts of data that need to be processed, analyzed and stored. From the researcher perspective, the sharing of the generated data can also be an extra requirement, so it can be processed and later accessed by the medical and research community members.

The multimodal analysis consists on combining information from several sources within a common referential, establishing spatial and temporal reasoning, that contextualized by a clinical and/or clinical protocol, can guide a diagnosis or a scientific conclusion. Good examples are voxel based morphometry where structural comparison enables the detection of changes in brain morphology along the time [65] or the fMRI based analysis of simple paradigms, like finger tapping that can be used to establish functional pathways associated to healthy condition [66,67] or associated to specific pathologies [68].

Another fact that increases the complexity of the BI scenario is that, typically computationally intensive algorithms are used to process the generated data. Usually this is according to the analysis of workflows that are both data and/or research objective dependent. Therefore two main issues arise. The first is the computational complexity of the methods used and the second is the usage of the methods that are conditioned by existing data/processing dependencies.

Regarding the intensive computational complexity, the range of processing methods used in these steps is wide and may present simple algorithms (e.g. thresholding the values of a given image) to more elaborated filtering [69], registration solutions or more complex and research oriented methods [70,71]. For that reason, it is only possible to determine if this issue is relevant or not taking into consideration the specificities of the application taking into consideration the size of datasets (from KB to GB), or the usage (applied once or to thousands of images in a given protocol) among several other factors.

The data/processing dependencies are typically associated with both processing and analysis protocols and related workflows. In neuroimaging, for instance, it is common to use a workflow based approach to the analysis supported in a scripting and/or programming support (including pre-processing stages) as it is clearly illustrated by the widely used SPM (Spatial Parametric Mapping) [72] and FSL (FMRIB Software Library) [73], that combines native applications and TCL based scripting packages. These packages, given the computational complexity of the methods, already consider some optimized methods which resource to Grid environments, namely to Sun Grid engine [74].

3.1. The case for Grid enabled brain research

Grid infrastructures appear as a good candidate to support a brain imaging scenario, and are already being successfully used in medical image processing to handle the demanding requirements of large images storage and communication, and to enable complex analysis workflows [52,1]. In order to make the use of the Grid environment, as a useful platform, there are two main aspects that must be considered before implementing a Grid enabled brain imaging research portal. These aspects are the storage and computing services that must be provided by the system.

A Grid based solution should have special requirements in relation to data security, due to the delicate nature of the medical images. The data should be stored in a secured environment, and only accessible to the brain researchers community members. It should also ensure the availability of processing applications, to run simple or complex analysis workflows. Providing a basic brain imaging research oriented semantic, and also be able to accommodate other possible research scenarios, is one of the main goals of this project. Such solution, should be accessible through a friendly user interface, like a web portal following the Science 2.0 fundamentals.

The Portuguese Brain Imaging Network (BING) [7] is an example of work being developed to provide such an environment to support the brain imaging research community, using the Grid as the computational environment to support storage and processing. The main goals of BING are:

- Develop an IT infrastructure to support collaborative use and sharing of neuroimaging data collections, analysis and modeling software and visualization tools.
- Provide a "neuroscientist-friendly" web portal to enable secure access to the available tools.
- Encourage scientific collaborations among participants from different research institutions and different areas of science that typically work independently, providing a virtual environment that promotes pluri-disciplinary studies on neuroimaging issues.
- Establish standards for multi-vendor biomedical data exchange between participants and enable equipment and procedures quality control.

There also some international efforts in terms of brain imaging Grid research. The NeuroLOG project is an example of work being done to enable neuroscientists to use the Grid potentialities, for brain imaging research, adopting the HealthGrid vision. This project aims to design a middleware, targeting a focused application area and adopting a user-centric perspective, to meet the neuroscientists demands [75]. Other example is the neuGRID project. The neuGRID project aims to create a user-friendly Grid-based research e-Infrastructure enabling the European neuroscience community to carry out research in the area of degenerative brain diseases. In neuGRID, the collection/archiving of large amounts of imaging data will be paired with computationally intensive data analyses [76]. And finally, another example is the NeuroGrid project. The Neurogrid aims at using the experience of other UK e-Science projects to assemble a Grid infrastructure, to conduct collaborative neuroscience research, and apply this to three exemplar areas: stroke, dementia and psychosis [77].

3.2. Requirements for Grid-enabled virtual labs

In the previous sections, some requirements were clearly identified. In terms of storage, the brain researcher should be able to store the images and associated metadata with the images, so they can be queried and accessed later by the researcher, and they should be stored in a secured environment, due to privacy issues associated with medical images. It must be considered the community or VO concept as a basis for data access policy. The data should only be available to that community (or to other communities, but only if allowed). The Grids fulfill this requirement through the usage of VO's defining virtual boundaries throughout its resources.

The security in the usage of medical images is also paramount, due to anonymity issues that must be considered, so a solution must be used to provide the anonymization or pseudo-anonymization of the stored data. Another important requirement to consider, is the necessity for the brain image researcher to organize his data in studies. This must be considered in the use-cases and the domain model of the system, as a way to organize the stored information.

Regarding the computing requirements, the user should be able to run computational analysis using the stored data as input. The Grid, through its middleware, provides the necessary tools to run custom applications that can range from simple application to more advanced scenarios like workflows. For instance, the Multi voxel fMRI analysis technique [71], used to calculate the association measure between two fMRI series, involves a sequence of processing steps (workflow). Normally the processing sub-steps are inter-dependent, and the resulting files from one step are the input to the following step(s). The main Grid middlewares, like gLite [39], GT Toolkit [23] or UNICORE [24], provide the necessary tools to run workflows, and thus transforming the Grid into an attractive platform to develop this kind of research platform. Another important issue to consider is the fact that these medical analysis algorithms are very heavy in terms of computation. Preferably, the system should have a great amount of computational resources, so the tasks can be divided, and therefore greatly diminishing the execution time of the application. Once again, the Grid appears as the ideal candidate.

Given this context, our objective was to explore the use of a Grid based solution for a brain imaging scenario, that could support the BING project integration with the Grid. The following chapters will present the work developed to support the brain imaging scenario. Chapter 4., presents the IGF Grid framework which is a semantic independent framework developed to support the MAGI web portal presented, in chapter 5.

4. Grid interfacing framework[1]

This chapter will present in detail the architecture and implementation of the IGF (IEETA Grid Framework), providing wrapping services to the Grid to support the e-Science portal (Figure 4.1). In this chapter all the diagrams presented will use the UML notation [78].

4.1. Why an interfacing layer

Our main motivation was to provide a Grid framework that provides an easy access to the Grid services by non-expert users but, to achieve such result, we needed to create an effective Grid enabled platform to make a bridge between our specific services and the Grid - the IGF (IEETA Grid Framework).

The IGF was developed with the purpose to create a Grid access layer independent from any of the semantic aspects of the portal, so it could be easily extendable, and possibly support other kind of Grid applications. Another reason to develop this framework, was to create a good alternative to the existing frameworks that work on top of gLite, that lack documentation and/or provide a more low-level approach and sometimes are of difficult usage.

The main objectives of IGF are:

- Provide basic services for storage and sharing of data on the Grid;
- Create a more high-level approach to the Grid, than the existing frameworks, and possibly use some of these frameworks to implement the low-level services;
- Provide good documentation.

1 The contents of this chapter were partly published in the Proceedings of Ibergrid 2009

- Provide a comprehensive framework solution, that supported a vast number of services (Computing, Storage, Proxy, Security).

4.2. IEETA Grid Framework (IGF) architecture

The IGF was developed to provide a developer-friendly Grid access layer, so it can be used in other Grid applications. It provides a clear Java [79] object-oriented API [80], with high-level abstractions.

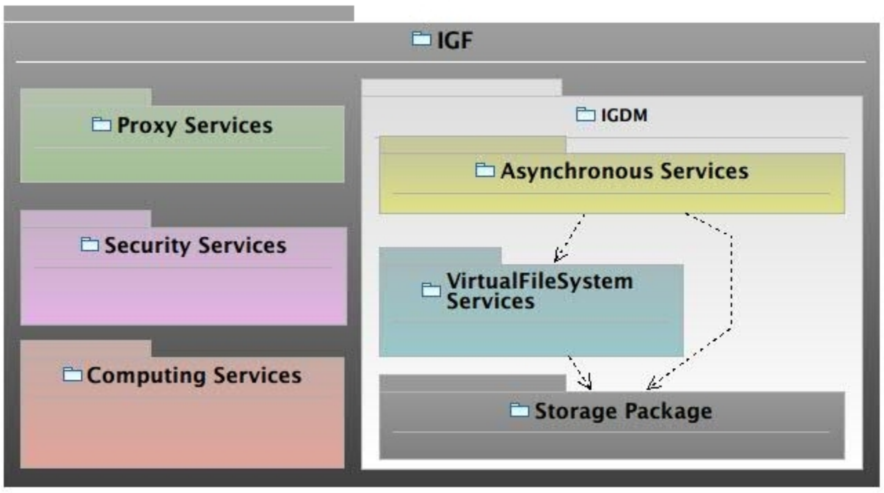


Figure 4.1: IEETA Grid Framework architecture – UML package diagram.

The main focus of IGF is the storage and sharing of information on the Grid*,* and the proposed architecture reflects that. The package IGDM (IEETA Grid Data Middleware) contains all the necessary services for the data Grid operations like copying, retrieving or searching files, etc. A Computing Services package is also available to provide the basic services to submit, and monitor jobs, a Proxy Services package to start and destroy user proxies, and a Security Package to provide a higher security layer for the data and computing operations.

Regarding design options, the API services are accessed through a main Factory class [81], that is responsible to construct the objects that implement the main services of the framework. For each of the interfaces there is a corresponding context class that is responsible for holding all the necessary information for the services to work (e.g. paths, environment variables, or other configuration options necessary for the Grid environment). These context classes were created to organize and centralize the necessary configuration options for the Grid environment. Every package has its own context class that is necessary for the respective service class to be instantiated. Through the API it is possible to select the type of service that is going to be instantiated. For instance, in the case of the IGDM Virtual File System Services, the developer can choose between the asynchronous service, or the normal (synchronous) service.

Another important feature of the framework, is the fact that every public IGF method (except for a few exceptions), from every class returns an object that implements the same interface, containing a return status, an output message and an error message. This feature improves the usability, because the user can easily access the all the information about the result of the method. It also improves the debugging because the possible error messages that appear in the results from the methods, of the packages top interfaces, are "filtered" throughout the layers of the package, appearing to the user the information that really matters.

4.3. Middleware services implementation

4.3.1. IGDM: Storage Package

The Storage Services contains all the operations responsible for dealing with both the File Catalog and Storage Elements (SE). All the operations that involve file transfers to and from SEs, and direct access to the File Catalog, including:

- Copying a file to a specific SE and registering the file in the File Catalog;
- Deleting a file from the catalog and from the SE;
- Retrieving a file to the User Interface (UI) ;
- Creating file replicas;
- Listing available Storage Elements (SE's).

To provide the Storage and File Catalog access, an internal module was implemented to interface the storage Grid services; the current implementation is based on delegating on the gLite system command line interface (CLI), instead of service interfaces. While this implementation can easily be replaced by a more convenient one, we decided to use this approach because it proved to be more stable and less error prone than using the existing scarcely documented Grid storage programming frameworks. The package was built having in mind the possibility to switch the CLI module to another kind of access to the storage and file catalog operations. The only service from this package that was implemented using another framework, was the GridFTP operations, on top of GSAF [37].

The IGDM package was also built having in mind, the possibility to provide the user several alternatives to execute the storage related operations. For instance, the user can call a File Catalog service that automatically transfers the file and registers it in the catalog, or alternatively, the user can “manually” transfer the file using GridFTP, and afterwards, register the file in the catalog using one of the available File Catalog service.

As depicted in Figure 4.2, the Storage package contains one main class that implements the *IGDMStorageServices* interface with the main methods from the package responsible for the basic storage operations: copy a file to a SE, retrieve a file, replicate a file, delete a file, create an LFC directory, etc. The diagram also depicts the usage of interfaces in some of the core services, to provide the possibility to extend or switch these modules.

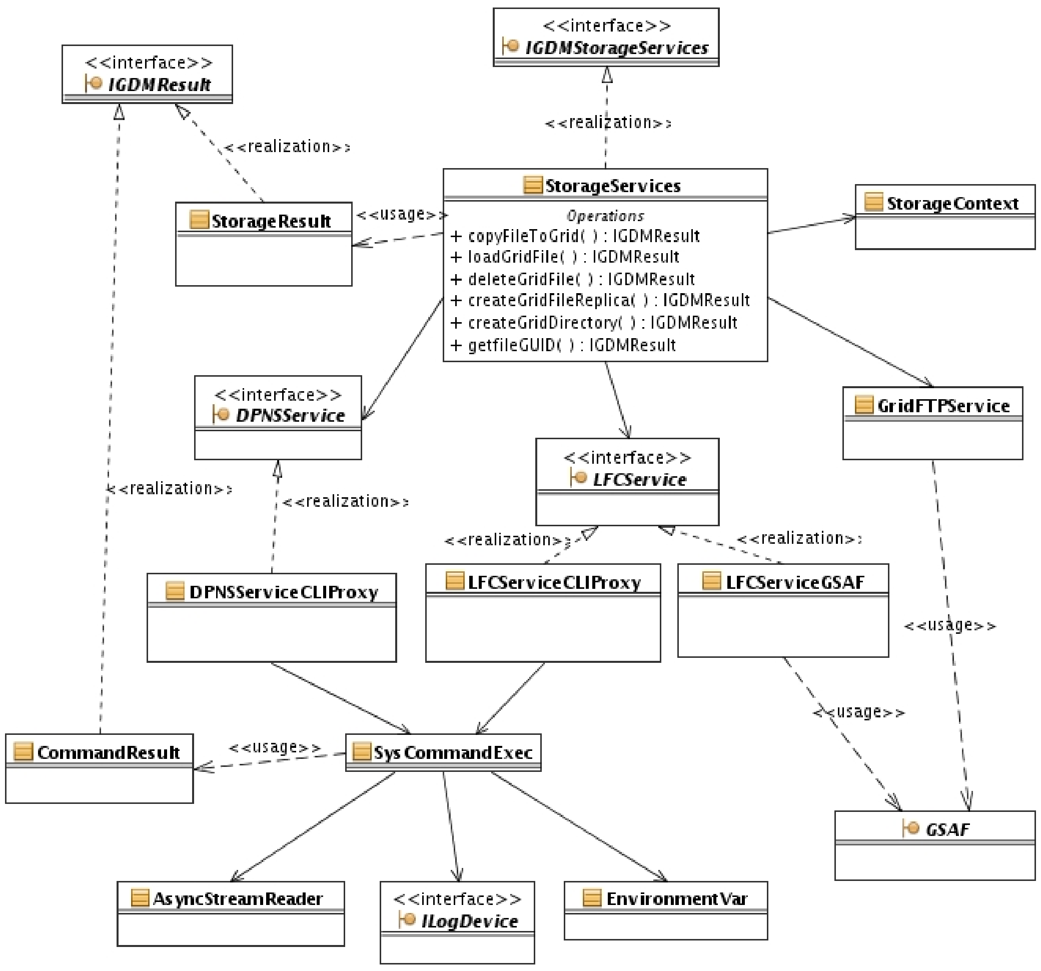


Figure 4.2: Storage package class diagram.

4.3.2. IGDM: Virtual File System Package

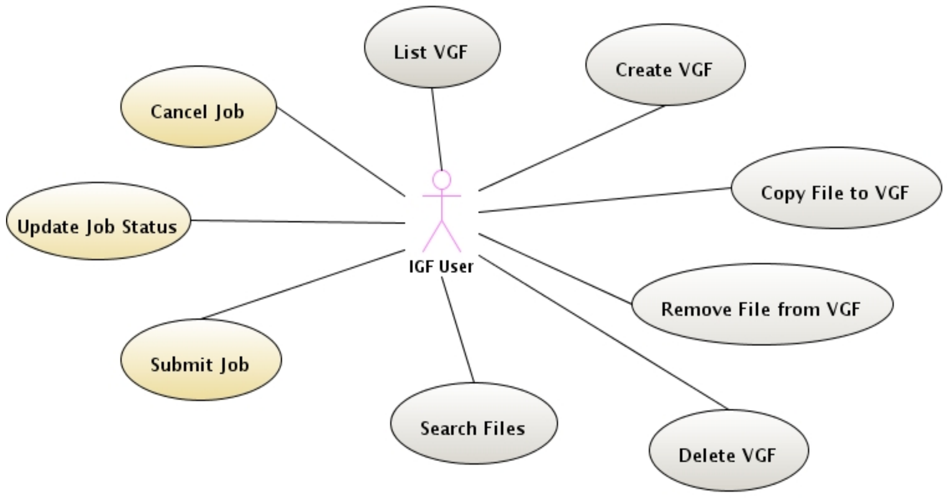


Figure 4.3: Main use-cases provided by the Virtual File System and Computing packages of the IGF Framework - (VGF stands for Virtual Grid Folder).

This package was created with the intent to provide a more high-level approach to the Grid storage, providing an abstraction closer to a regular file system, with files and folders. The services from this package depend on the Storage package services to work.

The Virtual File System (VFS) package main services are (Figure 4.3):

- Create a Virtual Grid Folder;
- Copy a file to a Virtual Grid Folder (with the possibility of creating replicas);
- List a Virtual Grid Folder;
- Search for file inside a Virtual Folder;
- Delete a file from a Virtual Folder;
- Delete a Virtual Folder;

The main purpose of this package is to provide a more intuitive way, for the users of this framework, to store and share their data on the Grid. To create this easy approach, we introduced a concept called Virtual Grid Folder (VGF), that basically is intended to represent a folder on the Grid (Figure 4.4). This virtual folder is composed by a real folder in the File Catalog, and a Collection in the AMGA catalog. The VFS core services rely on the AMGA metadata catalog [29].

The Virtual Grid Folder uses the AMGA catalog to maintain the association between the file catalog and the related metadata. This association is ensured by the unique file GUID that is included as part of the entry name in the AMGA catalog, so that the retrieval of stored files can be easily done. This can be overridden by the developer if he wants to use the file GUID, or if he wants to manually set the entry name in the AMGA catalog. Associated to the Virtual Grid Folder there is also the Tag concept that consists on an AMGA Schema that every Virtual Grid Folder must have. Every files stored in that folder must have the same schema as the folder.

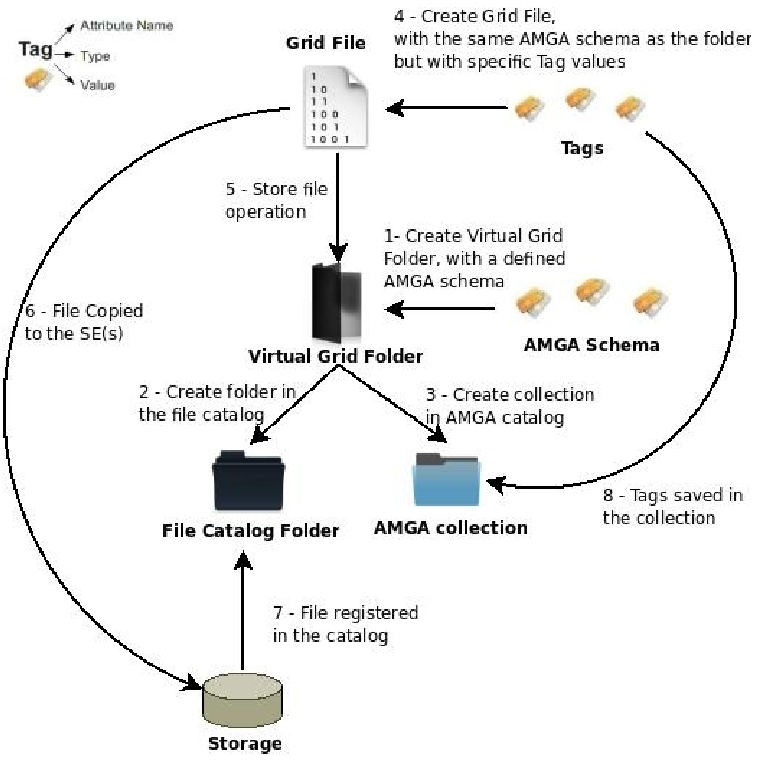

Tag
Attribute Name
Type
Value
Grid File
4 - Create Grid File, with the same AMGA schema as the folder but with specific Tag values
Tags
5 - Store file operation
1- Create Virtual Grid Folder, with a defined AMGA schema
AMGA Schema
6 - File Copied to the SE(s)
Virtual Grid Folder
2 - Create folder in the file catalog
3 - Create collection in AMGA catalog
File Catalog Folder
AMGA collection
8 - Tags saved in the collection
7 - File registered in the catalog
Storage

Figure 4.4: Relationships between Virtual Grid Folder (VGF), Grid Files and Tags, and workflow example of creating a VGF, followed by the copy of a file to that VGF.

As depicted in Figure 4.4, when a new Virtual Folder is created, the user must define a set of Tags to associate with that folder (AMGA schema), and the files that are stored in that same folder, must also have that same set of Tags, with the specific values for the files. For instance, we have a Virtual Folder called 'Images', and the defined schema for this folder is composed by two tags, 'size' and 'date'. If one wants to store a file inside this folder, he/she must also define the tags 'size' and 'date' with the corresponding values for that file. This way, one can search the files by using Tags (in the example, 'date' or 'size'), because the framework provides a method for searching the files by tags. Internally it queries the AMGA catalog, to select the respective entries that match the search. Also note, that you can create Virtual Folders inside other Virtual Folders, and that child Virtual Folder's don't need to have the same schema as the parent Folders.

All the main operations of the Virtual File System package have a transaction control that ensures the operations consistency. For instance if the user calls the method to copy a file to a certain virtual folder, and the operation responsible for copying the file and registering it in the catalog is successful, but the storing of the tags in the AMGA catalog fails, the first operation will be reversed, and the file will be deleted from the SE and from the file catalog.

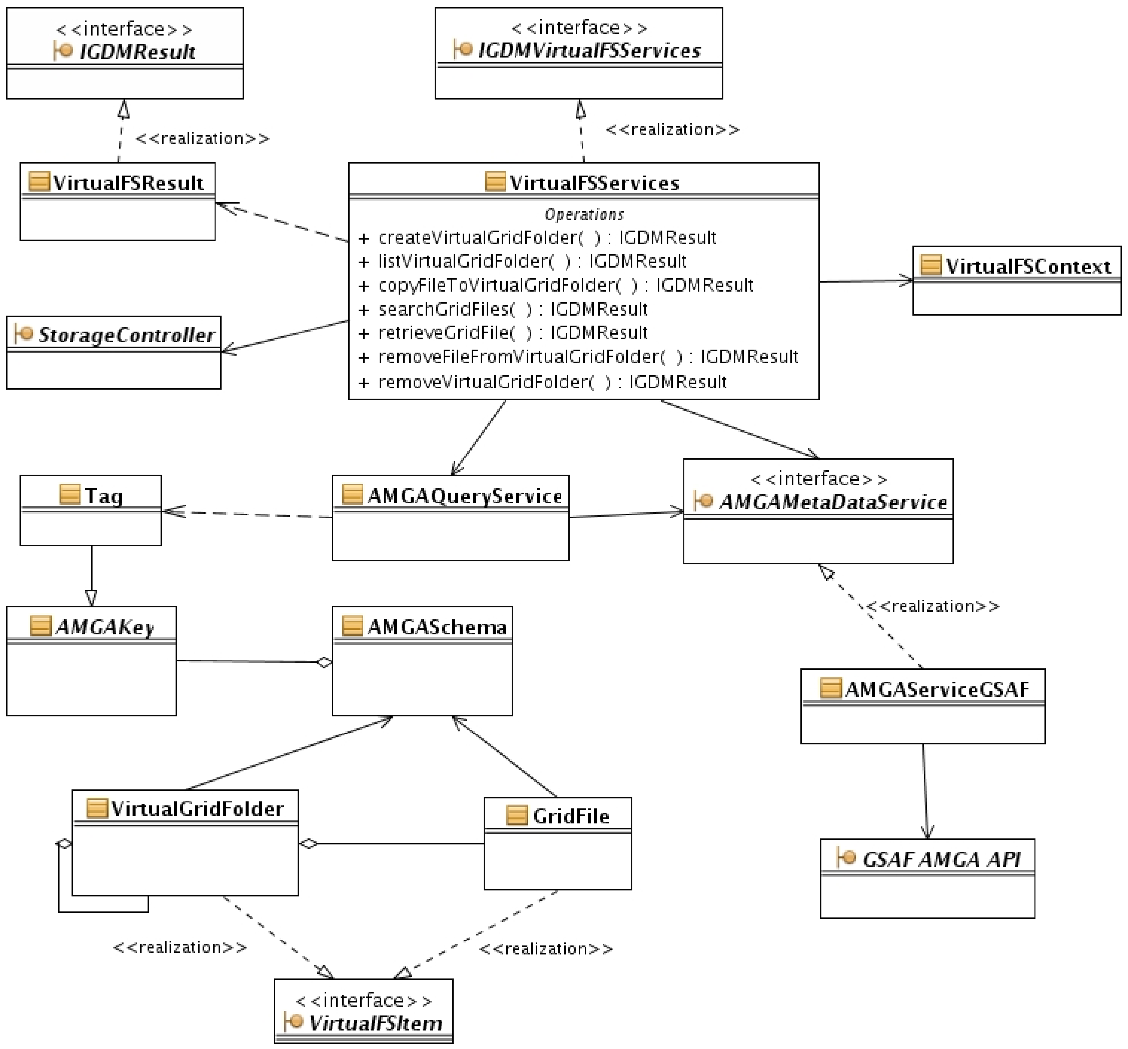


Figure 4.5: Virtual File System package class diagram.

This package also allows the user to access the lower level services, to perform more specific operations, like dealing directly with the AMGA catalog. The AMGA catalog access is performed using the GSAF framework AMGA API.

Similar to the Storage package, this package also contains one main class that implements the IGDMVirtualFSServices interface, that contains the main methods from the package: create Virtual Grid Folder, copy file to Virtual Grid Folder, retrieve file, search files, etc. (Figure 4.5).

4.3.3. IGDM: Asynchronous Package

The Asynchronous Services package was created with the purpose of providing an asynchronous execution mode for services of the Virtual File System and the Storage. As some of the Grid storage operations like storing, deleting or retrieving a file, can suffer random delays due to various factors, an asynchronous service provider would facilitate in

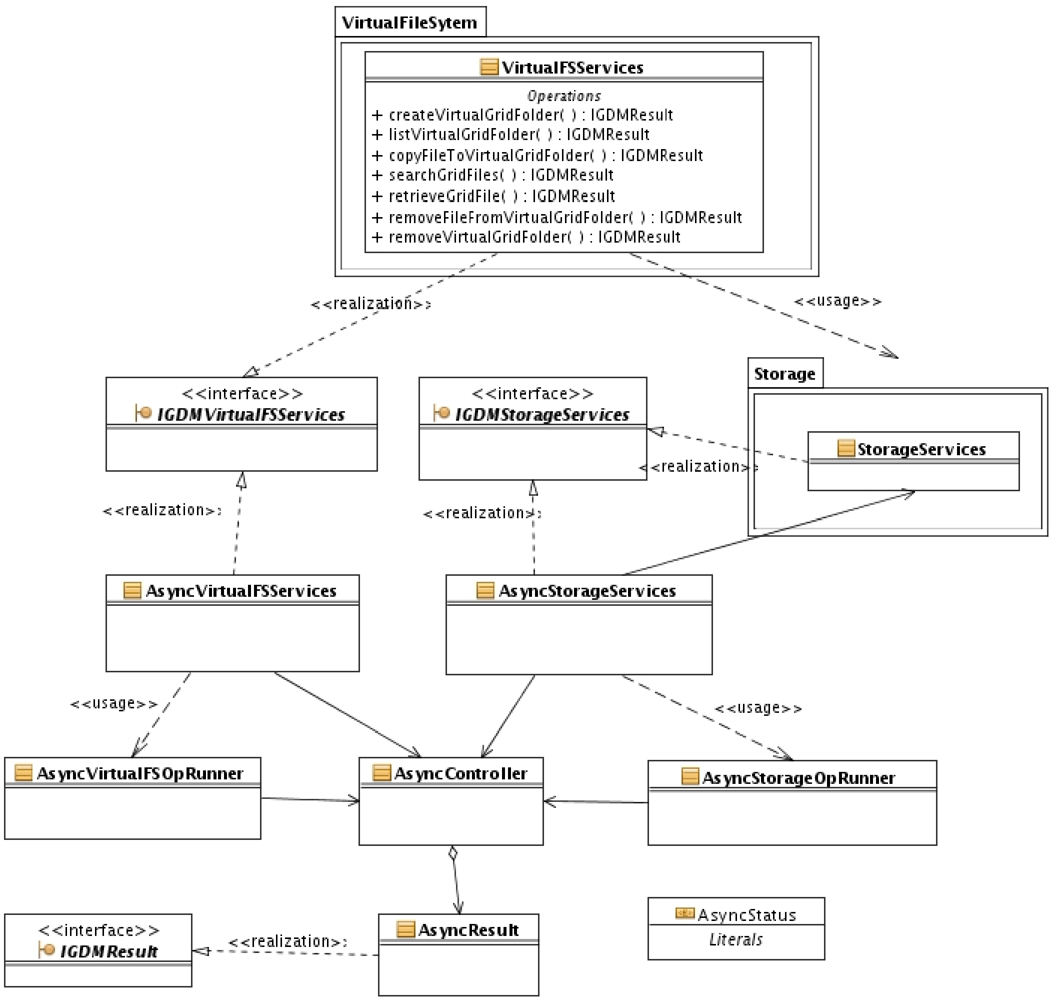


Figure 4.6: Asynchronous package class diagram

the construction of interactive interfaces that do not block waiting for results of running operations. This package ensures that individual calls to storage or virtual file system are executed on individual threads. When the corresponding thread finishes, it reports the result of the operation to a controller class that observes the running/executed operations.

In order to maintain a consistent and easy to use approach, this package was designed with the objective of providing asynchronous implementations to the IGDMStorageServices, and IGDMVirtualFSServices interfaces. The same methods presented in the previous sections from the Storage and Virtual FS packages are provided in this package in asynchronous mode (Figure 4.6).

As depicted in Figure 4.6, the classes AsyncVirtualFSOpRunner and AsyncStorageOpRunner, are the thread classes responsible for running the methods asynchronously, that report the results to an instance of the class AsyncController. This class contains a list of AsyncResult objects that hold the entire information corresponding to the execution of the thread with the operation. The classes AsyncVirtualFSServices and AsyncStorageServices contain a reference to an AsyncController object that contains the

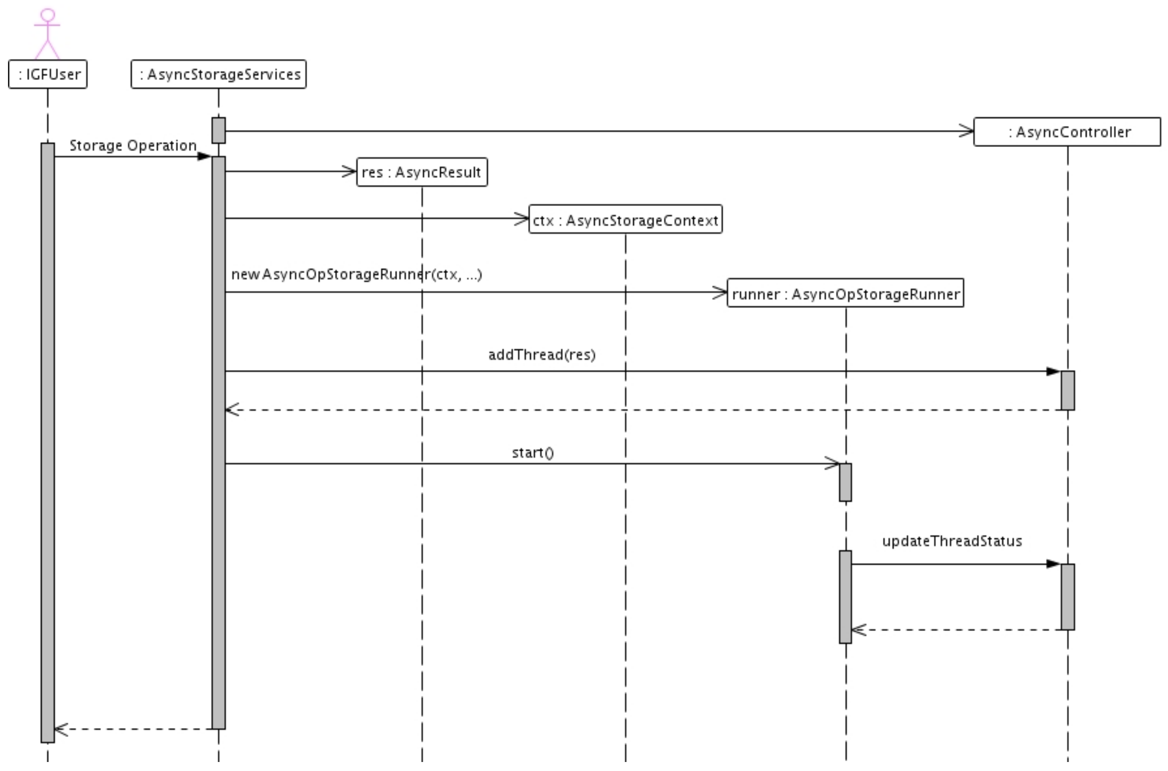


Figure 4.7: Asynchronous storage operation sequence diagram.

list of executed operations. To better understand the interactions between the classes of the package, a sequence diagram is presented, with a generic asynchronous storage operation (Figure 4.7). In case of a Virtual File System operation the interactions between are very similar, so the diagram also applies to that case.

4.3.4. Computing Package

This package provides basic computing services, and enables to develop applications that could integrate both storage and computing services provided by the same framework. The current release still doesn't support complex computing scenarios, like the usage of DAG jobs, but we plan to develop this feature in the future.

Like in the IGDM packages, this package also contains a main class (ComputingServices) that implements the IGFComputingServices interface, with the main services provided. The complexity of the package is centered in the class JobDescription, responsible for generating the job JDL, and if necessary, auxiliary scripts, based on the attributes defined in the ComputingContext object.

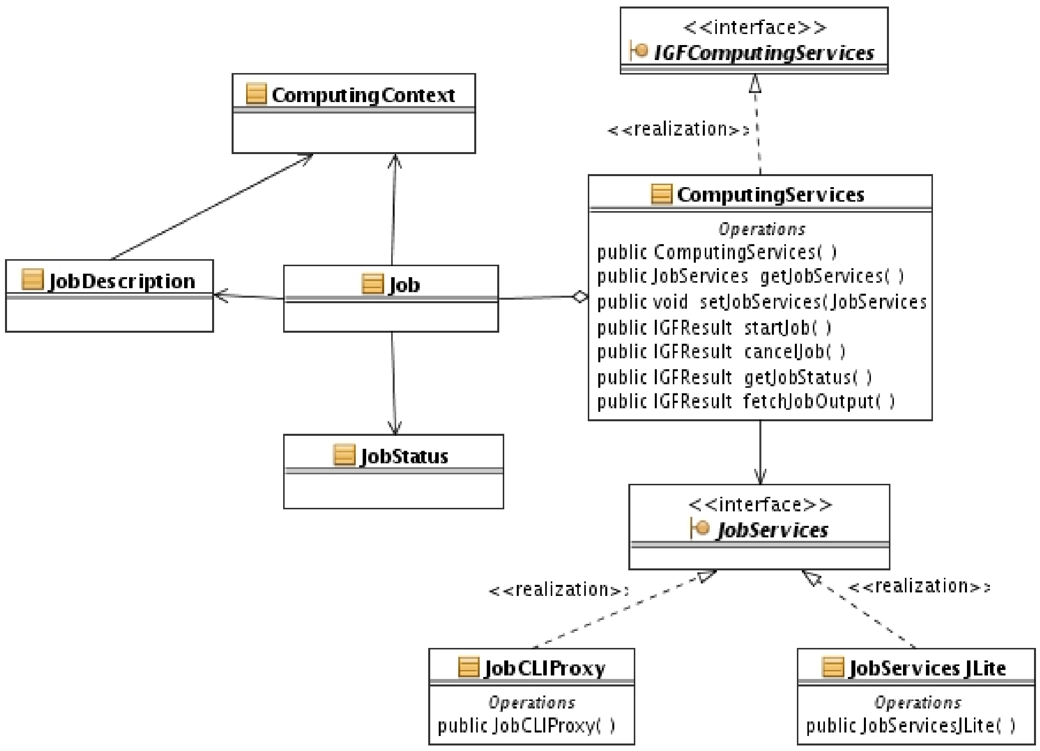

<<interface>>
IGFComputingServices
<<realization>>
ComputingContext
ComputingServices
Operations
public ComputingServices( )
public JobServices getJobServices( )
public void setJobServices(JobServices
public IGFResult startJob( )
public IGFResult cancelJob( )
public IGFResult getJobStatus( )
public IGFResult fetchJobOutput( )
JobDescription
Job
JobStatus
<<interface>>
JobServices
<<realization>>
<<realization>>
JobCLIProxy
Operations
public JobCLIProxy( )
JobServicesJLite
Operations
public JobServicesJLite( )

Figure 4.8: Computing package class diagram.

As depicted in Figure 4.8 the core services of the package have two alternate implementations provided by the classes JobCLIProxy and JobServicesJLite. The first one provides the calls to the *glite-wms* CLI commands, while the second uses one new experimental API called JLite [82]. When the user accesses the main factory object to create the Computing services object, he can choose between these two services.

4.3.5. Proxy Package

The Proxy package provides basic services related with Grid proxies. At this moment this is the "smallest" package and only has two operations available: create a VOMS proxy and destroying that proxy. In the future, this package could be extended to support more advanced proxy services like the usage of MyProxy servers [83].

4.3.6. Security Package

The Security package was created to provide a higher-level and alternative security solution to the ones already provided by the Grid environment itself. This package is still under development and its design is still an incomplete work. The basic idea is to create a Unix style user system. The Virtual Grid Folders and Grid Files, have a user and a group (like a VO inside the application domain) associated with it, each of them having the respective read, write and execute permissions. With these permissions, the developer can control the access to the Grid files and folders within the developed application itself.

4.4. Using the IGF API

As it was referred in the previous sections, one of the main objectives of this framework is to provide the users with an easy and well documented API to develop Grid enabled applications (see section 4.1.). To accomplish that goal we designed a "lightweight", "developer-friendly" API supported with documentation. The IGF was written in Java language, and uses some well known software design patterns.

In order to provide an *easy-to-use* approach, a main factory class was created to centralize the access to the services. To instantiate a service class, the user just has to access the factory object, select the desired service and pass the corresponding context object with the necessary Grid environment configurations. After that, a reference to the new object that provides the selected services is returned, and the user can start using the desired functions.

Another feature provided by the framework, is the ability to easily debug the applications.

To accomplish that, every method of the framework returns an object that implements a common interface. That interface provides a return status, an output string, and the most important in terms of debugging, an error message, in case any error occurs in the execution. Some of the methods also need to return other objects (e.g: Collections), and in that case, these objects are contained within the return object (that implements the common return interface (IGFResult or IGDMResult), and to access them the user just has to use the available method to retrieve that object from the return object. For a better understanding of these specific cases and operations, the respective *javadoc* documentation was created.

4.4.1. Virtual File System API

We will now present a few examples on how to use the API, more specifically the Virtual File System package, which is the most important package of the whole framework. The other packages follow the same methodology for the usage of the respective services. Note that in the examples, a variable called *CTX* will appear. Every time this variable appears, it represents the respective context object for the package, with the necessary attributes defined in it. Figure 4.9 depicts the instantiation and initialization of the *CTX* variable that will be used throughout the other examples.

```
VirtualFSContext CTX = new VirtualFSContext();

CTX.setCA_CERT_LOCATION("/etc/grid-security/certificates/");
CTX.setUSER_PROXY_LOCATION("/tmp/x509up_u502");
CTX.setLCG_CATALOG_TYPE("lfc");
CTX.setLCG_GFAL_INFOSYS("lcg-bdii.cern.ch:2170");
CTX.setLCG_GFAL_VO("dteam");
CTX.setLFC_HOST("prod-lfc-shared-central.cern.ch");
CTX.setLFC_HOME("/grid/dteam/ieeta.pt");
CTX.setLCG_RFIO_TYPE("dpm");

CTX.setAMGA_HOME("/grid/dteam/ieeta.pt");
CTX.setAMGA_HOST("localhost");
CTX.setAMGA_PORT(8822);
CTX.setAMGA_PASS("******");
CTX.setAMGA_USER("********");
```

Figure 4.9: Java-like pseudo-code example of the IGF AP - CTX (VirtualFSContext) object instantiation and initialization.

Figure 4.10 presents an example (using Java-like pseudo-code) of using the API, for a simple operation of copying a file to a VGF (Virtual Grid Folder), assuming that the virtual folder was created previously in the Virtual File System. In the example we can see the instantiation of a *VirtualFSServices* object, that implements the *IGDMVirtualFSServices* interface (variable *vfs*) through the main factory object. After that the user executes the operation, and the return results are stored in the variable *res*, that implements an

IGDMResult interface.

```
// Creates the folder information
VirtualGridFolder vgf = new VirtualGridFolder();
vgf.setVirtualGridFolderPath("MAGIRoot/UserX");

//Grid File information
GridFile gfile = new GridFile();

gfile.setName("TestFileName.img");


// Sets the location of the temp file in the UI
gfile.setUi_location("/tmp/TestFileName.img");


AMGASchema schema = new AMGASchema();
schema.add(new Tag("Subject", "varchar", "John Doe"));
schema.add(new Tag("Modality","varchar","fMRI"));
schema.add(new Tag("Date", "timestamp", now()));


gfile.setSchema(schema);

IGDMVirtualFSServices vfs =
       IGFServicesFactory.factoryVitualFSServices
       (IGFService.ASYNC_VIRTUALFS_SERVICES, CTX );


IGDMResult res = vfs.copyFileToVirtualGridFolder(gfile,vgf,
                                  StoragElementsList);
```

Figure 4.10: Java-like pseudo-code example of the IGF API – Copy file *gfile* to Virtual Grid Folder *vgf* .

The approach to the other available functions, is pretty similar to the one described above. Figure 4.11 depicts a Java pseudo-code excerpt, containing the necessary code to remove a file from the Virtual Grid folder. Note that after the remove file operation is performed, the

```
VirtualGridFolder vgf = new VirtualGridFolder();

vgf.setVirtualGridFolderPath("MAGIRoot/UserX");

GridFile gfile = new GridFile();
gfile.setGuid(FILE_GUID); //
gfile.setEntryName("guid");
gfile.setName("testFile.dcm");

IGDMVirtualFSServices vfs =
       IGFServicesFactory.factoryVitualFSServices
       (IGFService.VIRTUALFS_SERVICES, CTX );

IGDMResult res = vfs.removeFileFromVirtualGridFolder(gfile,
                                                      vgf);
if(res.getReturnStatus() != 0) {
       System.out.println(res.getReturnError);
}
```

Figure 4.11: Java-like pseudo-code example of the IGF API – Remove file *testFile.dcm* from VGF *MAGIRoot/UserX*.

result is stored in an object that implements the *IGDMResult* interface, and the verification for the success of the operation is performed. If the result is different from zero, it means that an error occurred, and that error can be accessed by the *getReturnError* method. Also note that the factory operation is different from the copy file operation (Figure 4.10). In this example a different services class is instantiated. This time the services selected are *VIRTUALFS_SERVICES* that provide synchronous operations, in contrast to the previous example that uses asynchronous operations.

Figure 4.12 presents the necessary code to list the contents of a VGF. Note that the definition of the AMGA schema for the folder is present. If the AMGA schema is not defined, the operation will only return the name of the files and folders within the target VGF without the tags. If the user wants to obtain the tags associated with the files, the AMGA schema must be defined, containing the attributes that the user wants to select. Also note that to obtain the results the user has two alternatives. In the first one (uncommented line at the end of the code in Figure 4.12), the VGF object passed to the method will be "loaded" with its contents, that can later be accessed by calling the *getFileList* or *getFolderList* methods of the VGF object. The other alternative (see commented lines at the bottom of Figure 4.12), consists of using the *IGDMResult* interface to obtain the results, like in the previous presented operations, but this time using a cast to convert to a specific results object (in this example the *VirtualFSResult*), and then access the methods to obtain the specific results.

Figure 4.13 presents another operation example, this time for the search of files stored in a

```
IGDMVirtualFSServices
       vfs=IGFServicesFactory.factoryVitualFSServices
       (IGFService.VIRTUALFS_SERVICES, CTX);

VirtualGridFolder vgf = new VirtualGridFolder();

vgf.setVirtualGridFolderPath("MAGIRoot");
List<AMGAKey> keylist = new ArrayList<AMGAKey>();
keylist.add(new Tag("Description", "varchar", ""));
keylist.add(new Tag("Subject", "varchar", ""));
keylist.add(new Tag("Modality", "varchar", ""));
keylist.add(new Tag("Equipment", "varchar", ""));
keylist.add(new Tag("Date", "timestamp", ""));
AMGASchema schema = new AMGASchema(keylist);
vgf.setSchema(schema);

vfs.listVirtualGridFolder(vgf);

List files = vgf.getFileList();

// IGDMResult res = vfs.listVirtualGridFolder(vgf);
// Collection files = ((VirtualFSResult)res).
//                           getResultFolder().
//                           getFileList();
```

Figure 4.12: Java-like pseudo-code example of the IGF API – List contents of Virtual Grid Folder *MAGIRoot*.

VGF. The operation is very similar to the previous one, that listed the contents of a VGF, in terms of code. The user also has to define the AMGA schema for the folder. This schema is used to define the tags that the user wants to have in the returning files from the operation. The major difference between the previous operation (List contents), and the search operation, lays in the definition of a list of Tag objects that will be used in the search. The operation will try to find all the files within the selected VGF, that match the tags passed in the list. Note that the result of the search operation can be obtained using a cast to the *VirtualFSResult* object (instead of using the normal *IGDMResult* interface) to store the results, and then use the *getCol* method to obtain the collection of files found.

```
IGDMVirtualFSServices
       vfs=IGFServicesFactory.factoryVitualFSServices
       (IGFService.VIRTUALFS_SERVICES, CTX);

VirtualGridFolder vgf = new VirtualGridFolder();

vgf.setVirtualGridFolderPath("MAGIRoot");
List<AMGAKey> keylist = new ArrayList<AMGAKey>();
keylist.add(new Tag("Description", "varchar", ""));
keylist.add(new Tag("Subject", "varchar", ""));
keylist.add(new Tag("Modality", "varchar", ""));
keylist.add(new Tag("Equipment", "varchar", ""));
keylist.add(new Tag("Date", "timestamp", ""));
AMGASchema schema = new AMGASchema(keylist);
vgf.setSchema(schema);


List<Tag> tags = new ArrayList<Tag>();

tags.add(new Tag("Description", "varchar", "XPTO123″));

tags.add(new Tag("Subject", "varchar", ‶S32JPinto″));

tags.add(new Tag("Modality", "varchar",″fMRI″));


VirtualFSResult res = (VirtualFSResult) vfs
                            .searchGridFiles(vgf, tags);

List results = res.getCol();
```

Figure 4.13: Java-like pseudo-code example of the IGF API – Search files stored in the Virtual Grid Folder *MAGIRoot.*

Finally we also present an example (see Figure 4.14) of how to create a VGF with IGF. The user must define the folder path that will be used both in the file catalog path and in the AMGA catalog collection, and the respective AMGA schema that the folder will support.

```
IGDMVirtualFSServices
       vfs=IGFServicesFactory.factoryVitualFSServices
       (IGFService.VIRTUALFS_SERVICES, CTX);

// Creates the folder information
VirtualGridFolder vgf = new VirtualGridFolder();

List<AMGAKey> fkeys = new ArrayList<AMGAKey>();
fkeys.add(new Tag("Description", "varchar", ""));
fkeys.add(new Tag("Subject", "varchar", ""));
fkeys.add(new Tag("Modality", "varchar", ""));
fkeys.add(new Tag("Equipment", "varchar", ""));
fkeys.add(new Tag("Date", "timestamp", ""));

AMGASchema fschema = new AMGASchema(fkeys);

vgf.setVirtualGridFolderPath("MAGIRoot/UserX");
vgf.setSchema(fschema);


IGDMResult res = vfs.createVirtualGridFolder(vgf);

System.out.println(res.getReturnString());
```

Figure 4.14: Java-like pseudo-code example of the IGF API – Creation of the Virtual Grid Folder *MAGIRoot/UserX*.

4.4.2. Computing API

In this section we'll present a brief example on how to use the Computing package of the IGF API. The overall strategy for using the Computing package follows the same principle as the Virtual File System package presented before. There is a *ComputingContext* class, responsible for holding the parameters necessary to configure the environment and generate the job JDL file. Figure 4.15 shows the instantiation and initialization of the *CTX* variable that will be used in the next computing examples. Figure 4.16 shows the necessary code to create a new Job object and submit that job to the Grid. Note that there are more parameters that can be configured in the *ComputingContext* object, but in order to simplify the example we only show the most important ones. For instance the user can define the specific CE where the job will run, or specify other kind of special requirements. Figure 4.17 shows the code to update the job status of the job launched in the previous example. Finally in Figure 4.18, we present the example of how to retrieve the job output files from the previously submitted job, and store these files in a specific directory in the UI machine.

```
        ComputingContext CTX = new ComputingContext();

        CTX.setUID(this.sharedinfo.getLoggedUser().getUsername());

        CTX.setJDL_STORE_FOLDER("/tmp");
        CTX.setJDL_FILENAME("MAGIJob");
        CTX.setWMS_PROXY_SERVER("https://wms01.lip.pt:7443/
                             glite_wms_wmproxy_server");
        CTX.setRANK("other.GlueCEStateFreeCPUs");

        CTX.setEXECUTABLE(/home/User/GridApps/association);
        CTX.setSTD_OUTPUT("std.out");
        CTX.setSTD_ERROR("std.err");
        CTX.setARGS(" input_file V2 2 45 output.txt"); //String with
the application arguments

        List insandbox = new ArraList();
        insandbox.add("/tmp/data/input_file.hdr");
        CTX.setINPUT_SANDBOX_LIST(insandbox);

        //Data stored in SE's
        Collection<GridFile> inputData = new ArrayList<GridFile>();
        GridFile inputtest = new GridFile();
        inputtest.setName("input_file.img");
        inputtest.setGuid(FILE_GUID);
        inputData.add(inputtest);
        CTX.setINPUT_DATA(inputData);

        List outsandbox = new ArraList();
        outsandbox.add("std.out"); outsandbox.add("std.err");
        outsandbox.add("output.txt");
        CTX.setOUPUT_SANDBOX_LIST(outsandbox);
```

Figure 4.15: Java-like pseudo-code example of the IGF API – *CTX* (*ComputingContext*) object instantiation and initialization.

```
        IGFComputingServices cs =
        IGFServicesFactory.factoryComputingServices(
                            IGFService.COMPUTING_SERVICES_CLI, CTX);

        Job job = new Job(CTX.getJDL_STORE_FOLDER() + "/" +
               CTX.getJDL_FILENAME() + UID +".jdl", CTX);

        ComputingResult res = cs.submitJob(job);

        if(res.getReturnStatus() != 0) {
               //Process error
               System.out.println(res.getReturnError();
        }
```

Figure 4.16: Java-like pseudo-code example of the IGF API – Submit a job, configured with the *CTX* context variable.

```
        // ... Code from job submission here...

        ComputingResults ures = cs.updateJobStatus(job.getId());
        int jobStatus = job.getJobStatus().getStatus();

        if(ures.getReturnStatus() != 0) {
               //Process error
               System.out.println(ures.getReturnError();
        }

        // Convert the jobstatus from int to String
        if(jobStatus == 2)
               String STATUS = "RUNNING";
               // Just an example not real status codes
```

Figure 4.17: Java-like pseudo-code example of the IGF API – Update the job status of a specific job, with an ID defined by the middleware upon the job submission.

```
        // ... Code from job submission here...

        int jobStatus;
        //checks if the job has finished
        do {
               jobStatus = cs.updateJobStatus(job.getId());
               Sleep(POOLING_TIME);

               if(jobStatus == JobStatus.SUCCESS ||
                      jobStatus ==  JobStatus.CLEAR ||
                      this.jobStatus == JobStatus.ERROR )
               break;
        } while(true);

        String outputPath = "/tmp/" + job.getShortJobId();

        if(job.getJobStatus().getStatus() == JobStatus.SUCCESS) {

               res = cs.fetchJobOutput(job.getId(),outputPath);

               . . .
        }
```

Figure 4.18: Java-like pseudo-code example of the IGF API – Fetch the job output from a specific job, with an ID defined by the middleware upon the job submission.

In this section, we presented the IGF API together with a small tutorial exemplifying its use. In the next chapter it will be presented the MAGI web portal, which makes use of the IGF API to implement the basic Grid services and therefore demonstrate the real potential of this framework.

5. e-Science Portal for brain imaging research[2]

In this chapter we will present an e-Science brain imaging research portal called MAGI, which stands for Medical Application Grid Interface together with details on its implementation. It will also be discussed the integration of the MAGI with the IGF framework, described in the previous chapter, to enable the access to the Grid environment.

5.1. The MAGI Portal as scientific community enabler

In order to provide the non-experts users (e.g: scientists, medical researchers) the great storage capabilities and computational power of the Grid, we propose a web portal, that enables these users to access Grid resources. The portal will act as a scientific community enabler by, disseminating good practices, making expensive resources available to the scientific community, and by facilitating the sharing of data and experimental results. The construction of this portal is meant to be a prototype approach for the BING [7] and GERES-med [49] projects integration with the Grid environment. The developed web portal is called MAGI, that stands for Medical Applications Grid Interfacing portal.

The definition of the MAGI system was centered in the following objectives:

- support domain-oriented semantics, allowing researchers to express their experiments with concepts they are familiar with (never needing to handle computational infrastructure issues);
- provide intuitive, friendly user interfaces for the end-users;

2 The contents of this chapter were partly published in the Proceedings of Ibergrid 2009

- seamlessly integrate basic Grid services (currently over gLite), covering both storage and job execution and security.

Our present focus is on modeling common medical imaging research requirements and support them in the web portal. These scenarios have been already identified in previous works, like the BING and GERES-med projects [49,7]. The system is still in a prototype phase, but it was developed having in mind the need for extensibility, for instance, to support more advanced functionalities or complex processing workflows, like in MOTEUR [84], or more sophisticated query engines (e.g. content based querying).

5.2. Brain imaging research portal use-cases

Based on our group experience in the context of previous projects [49,7] we were able to identify basic use cases to develop a community research portal (Figure 5.1), to support medical researchers, specially with respect to biosignal processing and medical imaging modalities, with special focus on brain imaging.

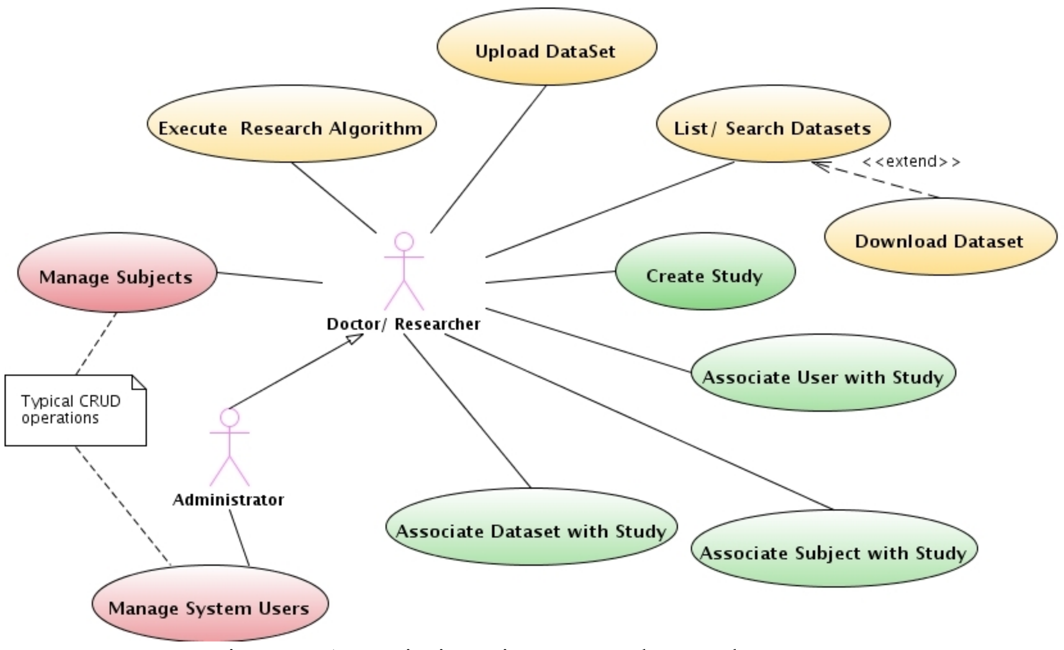


Figure 5.1: Brain imaging research portal use-cases.

The main use cases, can be divided into three main groups, as we can see by the colors in Figure 5.1 and are described in detail in Table 5.1.

Group	Use-Case	Details
Data related Use-Cases	**Upload dataset**	The user uploads a file to the storage environment.
	Search/List datasets	The user selects a filter or defines a search parameter to obtain a list of Datasets, and possibly retrieve the correspondent files.
	Execute research algorithms	The user selects dataset(s) as input for an available research oriented algorithm, that is available in the interface.
User and Subject Management Use-Cases	**Manage Subjects**	Typical (CRUD) create, read, update and delete operations, on the user subjects.
	Manage Users	Typical (CRUD) create, read, update and delete operations, on the system users. (Can only be performed by system administrators)
Studies Use-Cases	**Create Study**	The user creates a new study, and “owns” it. Only that user can add more users to that study.
	Associate User/Subject/Dataset/ with studies	The user selects items (Users, Subjects or Datasets) to associate with studies. The study entity possesses a group of users, subjects and datasets associated with it.

Table 5.1: MAGI Portal use-cases details.

5.3. Domain concepts model

After defining the basic use cases for a brain imaging research portal (see section 3.2.), we identified the most important concepts to map them in the MAGI domain model (see Figure 5.2). In MAGI, a researcher (User) belonging to a specific Organization may be responsible for several Subjects that he can gather in one or more research studies (Study). Most of his research activity is centered in analyzing Datasets of a specific Modality, corresponding to a Subject, obtained using a specific Equipment. We decided to add two more classes (Mail and Task) to the domain model that can be considered “strangers” to the medical research context. Both of these classes were created to support the interaction between the user and the execution of Jobs on the Grid via web portal. The class Mail, is used to map the concept mail (or message), and it is used for interconnection between the users of the system, or for sending automatic messages from the system to the users. For instance, upon a task completion on the Grid, the user receives a message with the final result of the execution. This prevents the user to be logged on while waiting for a task to complete. The Task class is used to map the task concept (or Grid job), and it was created for monitoring and statistical reasons. The user has a list of his tasks, and can see information, like the current status of the task, submission date or a running time, etc.

This schema with simple modifications is able to accommodate a wide set of application scenarios, and allows different research fields that make use of medical imaging techniques to use this application and harness the Grid benefits. It fits our brain imaging research goals, but with further refinements it can suit specific medical analysis applications.

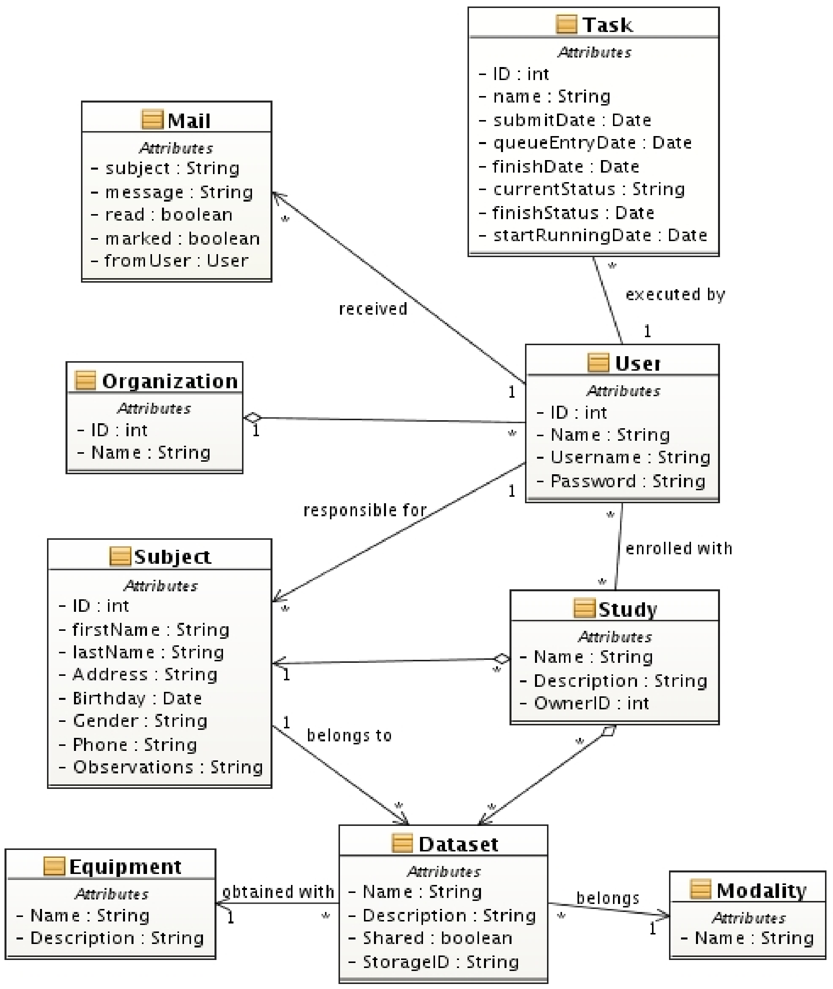


Figure 5.2: Domain model class diagram.

5.4. MAGI System architecture

MAGI has a classic three tier layered architecture composed by a top package (User Interface layer), a middle package (Domain Logic layer) and a lower package, which is the Grid interfacing layer called IGF, described previously (see chapter 4.).

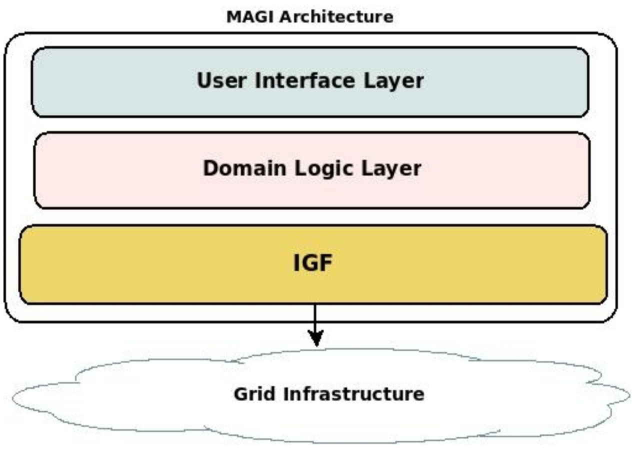


Figure 5.3: MAGI system architecture.

The top layers contain the application specific semantic, while the bottom layer (IGF) is the semantic independent framework that allows the users to access the Grid's core services, like storage and computing.

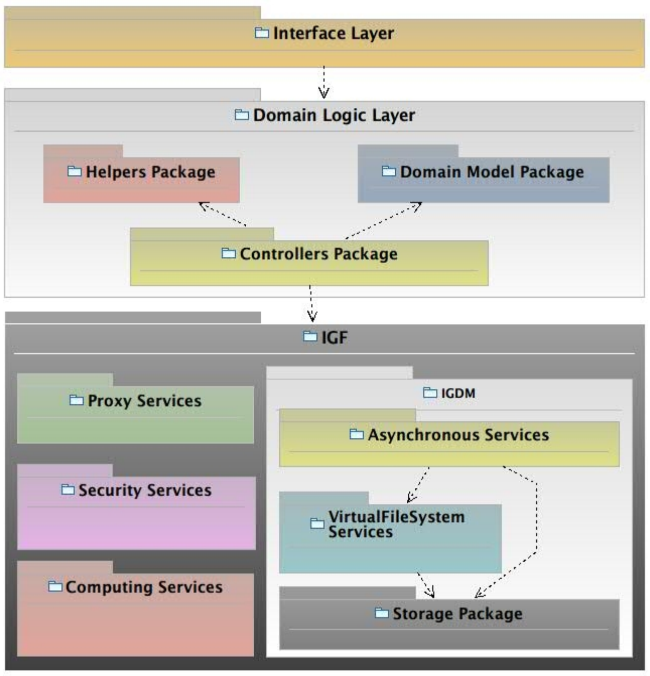


Figure 5.4: Detailed architecture of the MAGI, containing all the packages and IGF.

The domain logic layer was created to make the bridge between the use-cases available in the web portal and the Grid interfacing layer. This package was subdivided into 3 packages: Controllers package, Model Package and Helpers package (Figure 5.4).

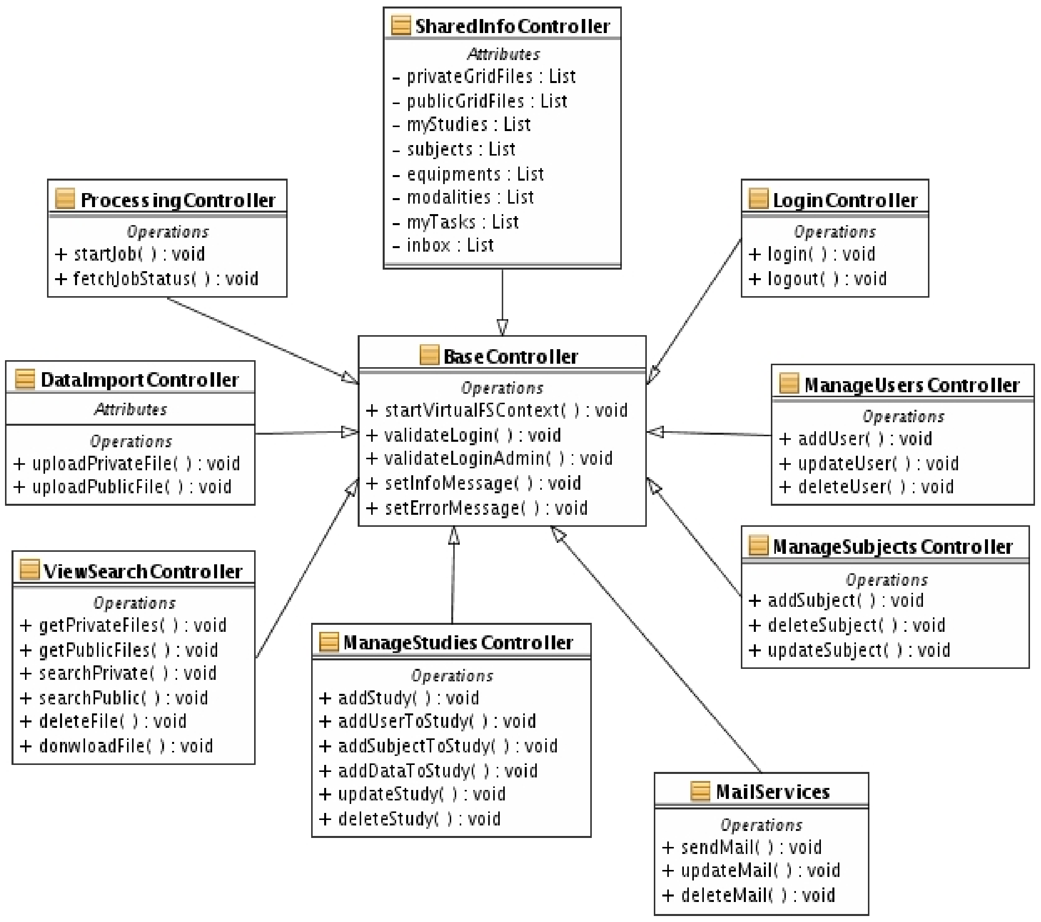


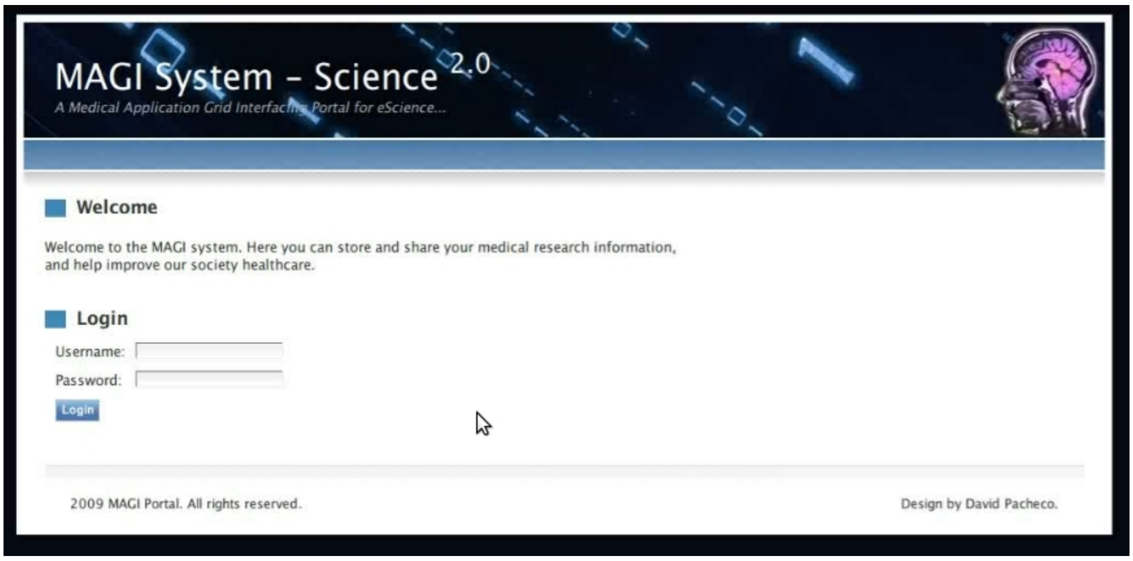
Figure 5.5: Controllers package class diagram, showing the most important methods and attributes.

The Controllers package contains all the main classes responsible for the interaction between the interface and both the system's database and the Grid interfacing layer. As depicted in Figure 5.5 , the package is composed by a parent class called BaseController, that holds the common methods for all the other controller classes. There is also another class in this model that plays a central role in the operations of the portal, which is the SharedInfoController class. This class centralizes all the important information necessary to display in the web interface, essentially the lists of objects (Subjects, Studies, Files, etc.), and also provides the necessary methods to fetch and update these lists. All the other controller classes were created and organized based on the defined use-cases that had to be implemented.

The Model package contains the domain model classes (section 5.3.) and finally the Helpers package contains a few classes created to help in the execution of some of the operations that involve the Controllers and Model packages (from the Domain logic layer) and the Grid access layer.

5.5. Portal services implementation

The MAGI *Web* portal was designed to be user friendly and to have an attractive look. The portal supports visual rich interactions, such as intuitive *drag-and-drop* functionalities that give the users the idea that they are dragging the information (e.g subjects, datasets, objects, files) between semantic rich containers (e.g: folders, studies, subject groups), hiding the task details, like moving, copying files between specific file system/Grid location, while maintaining the coherence of the semantic data model. We also tried to maintain a coherent, similar look and interactions methods throughout the pages, so the human interfacing interaction could be more intuitive.


MAGI System - Science 2.0
A Medical Application Grid Interfacing Portal for eScience...
Welcome

Welcome to the MAGI system. Here you can store and share your medical research information, and help improve our society healthcare.

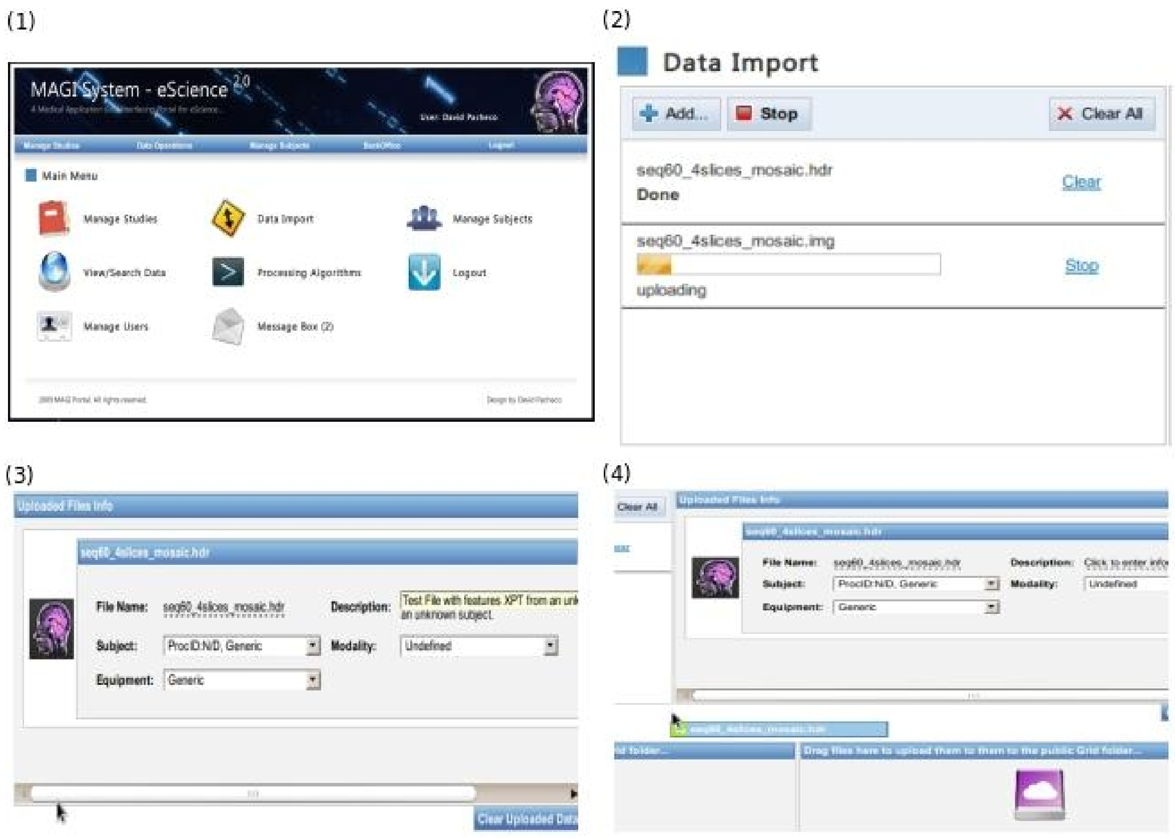
Login
Username:
Password:
Login
2009 MAGI Portal. All rights reserved.
Design by David Pacheco.

Figure 5.6: MAGI login page.

The Web portal was developed using open technologies: JSF (Java Server Faces) [85], and the Richfaces framework [86] to provide some of the functionalities that require the use of AJAX. JSF is a Java-based Web application framework intended to simplify development of user interfaces for Java Enterprise applications (J2EE). Unlike the request-driven MVC

(Model-View-Controller) [87] web frameworks, JSF uses a component-based approach. The state of the UI components is saved when the client requests a new page and restored when the request is returned. We used JavaServer Pages (JSP) for JSP display technology, but there are applications that use other technologies, such as XUL (XML User Interface Language). The other technology used, Richfaces, is a rich component library for JSF and an advanced framework for easily integrating AJAX capabilities into applications. RichFaces takes advantage of the benefits of the JSF framework including lifecycle, validation, and conversion facilities, along with the management of static and dynamic resources.

For the back-end supporting database, the technologies chosen were PorstgreSQL [88] as database management system, and Hibernate Annotations [89] to execute the mappings between the domain model classes and the relational database tables.



seq60_4slices_mosaic.hdr
Done

seq60_4slices_mosaic.img
uploading

seq60_4slices_mosaic.hdr
File Name: seq60_4slices_mosaic.hdr
Description: Test File with features XPT from an unknown subject.
Subject: ProcID:N/D, Generic
Modality: Undefined
Equipment: Generic

Figure 5.7: Web portal pages, demonstrating the data import use-case.

In Figure 5.7, the flow of events when uploading a data file is illustrated: (1) the user chooses to upload a file; (2) then he/she picks a local file to transfer, (3) enters descriptive file metadata and (4) “drags” the file icon into to the *Grid* container (thus triggering seamlessly underlying Grid operations).

5.5.1. Security design

Regarding security and authorization, the MAGI system grants the users access to the Grid using a delegation approach. This means that the users are authenticated when they log-in to the portal, and they can only use the Grid services through the portal. Another important security issue is the one related with Grid certificates and user proxies. For now, we use a single X.509 certificate to the whole system. Since the user's actions on the Grid are limited by the portal's use cases, we don't see the usage of a single certificate as a downfall. In the future, if proper isolation of user data or tasks is advised, one should consider the usage of proxy repositories to provide an alternative solution [83].

In a shared research environment, especially in those involving human subjects, the privacy and anonymity of data is always an issue. Our system was created with a research oriented semantic, and the target datasets are only for research purposes and not medical care scenarios; still, the system needs to adopt solutions to provide anonymization. To address this problem, we store pseudo-anonymized data, so it is not possible to associate a dataset with a subject, using only the information stored on the Grid. To create the pseudo-anonymization, we store the subject's Id (a surrogate key, free from domain semantics) instead of the name, as one of the file Tags, to preserve the subject's privacy, and only using the MAGI's private database it is possible to associate the data with the subject. To harden the security of the system, we also plan to develop and make use of the Security package from the IGF framework (Figure 4.1) or consider more advanced security scenarios (like in [90]).

5.5.2. MAGI Integration with the IGF Framework

The domain model presented in section 5.3. was used to create a back-end database to support the MAGI portal. This database has a scope different from the Grid storage. While the database is used to store the tables correspondent to the domain model, the Grid storage is used to store the files and the respective associated metadata in the AMGA catalog.

The concept Dataset is used as a generalization for medical images. In order to make use of the IGF framework and store the datasets on the Grid, we had to define an AMGA schema to associate with the files. The schema used for the medical datasets is composed by the file name, a description, the patient Id, the modality, the medical equipment and a time-stamp. Also note, that the concept Dataset used in the domain model has an attribute 'shared' (Figure 5.2), that is used to define whether the dataset will be stored in the pool of shared datasets (accessible to everyone within a given community), or in the user's private

Virtual Grid folder, and only accessible to that user, responsible for uploading the file. The user can therefore decide if the dataset will be public or not.

To make the bridging between the Web portal and the Grid environment, the services from the Domain Logic layer of the MAGI architecture (Figure 5.3), access the Grid using the IGF API calls, similar to the one presented in Figure 4.10. The user interacts with the Web interface, performing the necessary tasks (filling forms, uploading files, etc.), and the respective Domain Logic services use the information from the user, to create the necessary variables (e.g: creating an AMGA schema with a set of Tags – Figure 4.4), or call the Grid services from the IGF API, to perform the necessary tasks. After the execution of the Grid operations, the Domain Logic services can use the information returned from the IGF calls, to show the necessary information in the Web interface.

5.6. Experimental results

5.6.1. Storage level

Since the main focus of this work is on storage, we decided to run performance tests on both the IGF framework and the MAGI portal storage capabilities, in order to conclude if

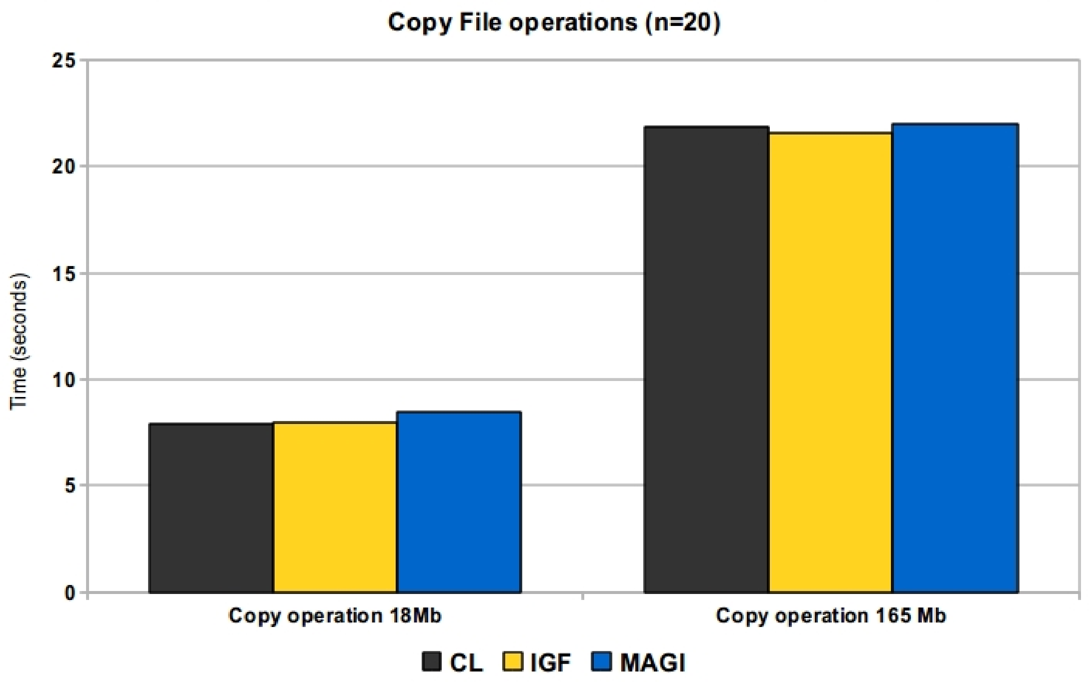


Figure 5.8: Chart representing the file transfer times, comparing the usage of the command line (CL) the IGF and the MAGI system.

the developed software introduced any delay, when comparing with the command line interface operations. The storage tests, consisted on the transfer of two files (real brain imaging research files), one with 18 Mega-bytes and another one with 165 Mega-bytes, from the UI node to a storage element (file copy), and the inverse operation (download file).

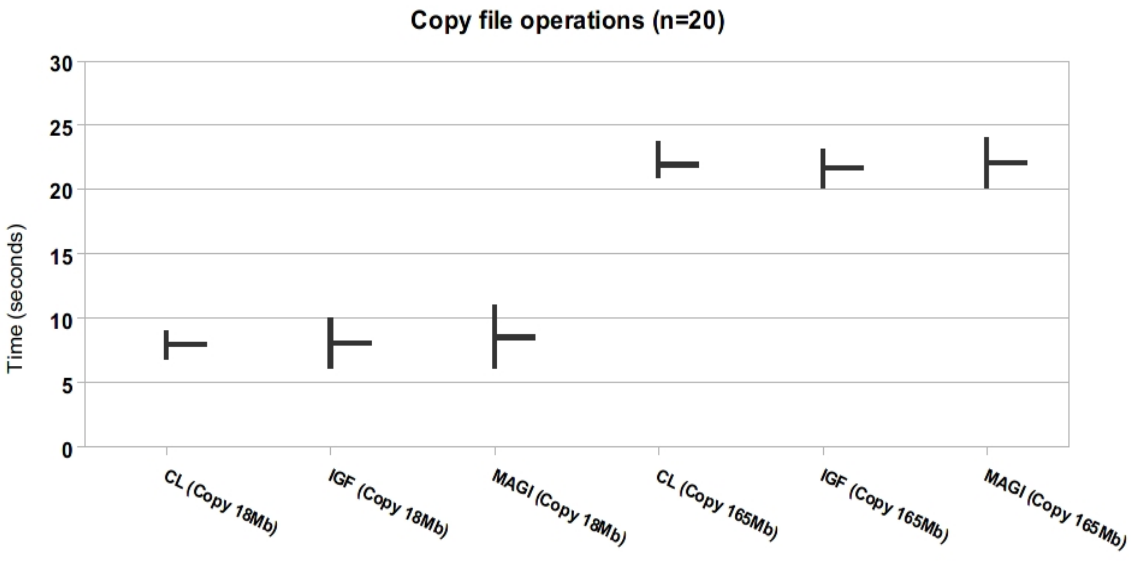


Figure 5.9: Chart comparing the maximum, minimum and average times of each of the file copy tests.

Both the UI node and the storage element were on the same local network. Note that the file copy operation consisted not only in copying the file from the UI to the SE, but also in registering the file in the LFC file catalog (remote server located in CERN). In the specific case of the MAGI portal, the files were also registered in the AMGA metadata catalog. Although this last operation differs from the storage operations performed with the command line and the IGF, it was deliberately done in order to conclude if the delay introduced by the metadata registration was considerable or not.

In terms of results, the delay introduced both by the IGF and the MAGI portal is in average less than 0.5 seconds. We can see the difference between the three systems in Figure 5.8, for the file copy operation and in Figure 5.10, for the file download operation. The charts in Figure 5.9 and Figure 5.11, show the the comparison between the maximum, minimum and average times for each case, proving that there aren't any discrepant values in the results and that the average values are accurate.

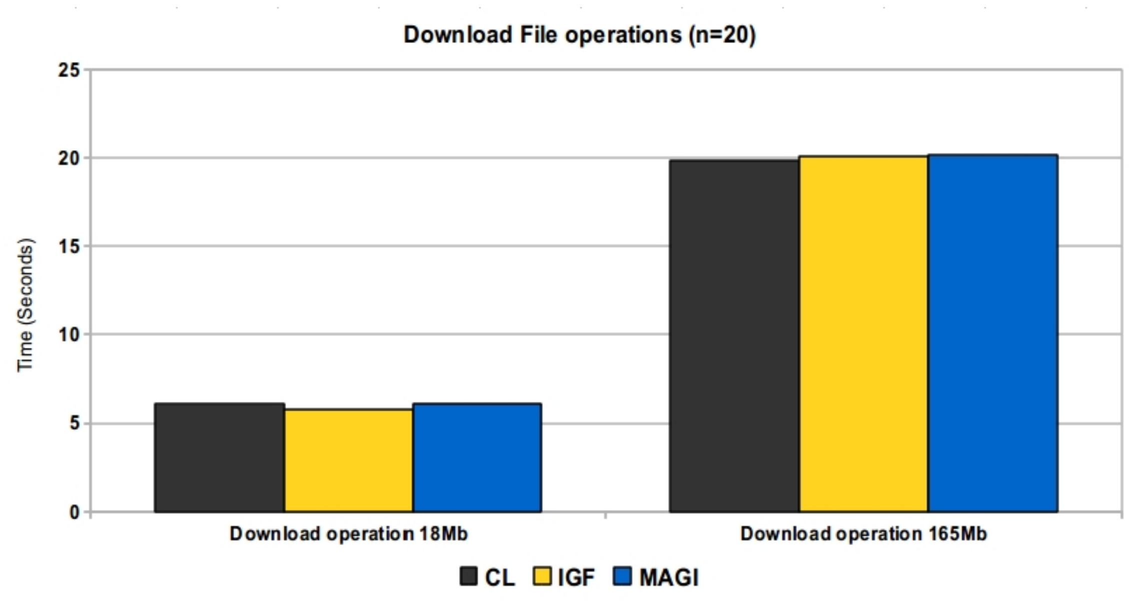


Figure 5.10: Chart representing the file download times, comparing the usage of the command line (CL) the IGF and the MAGI system.

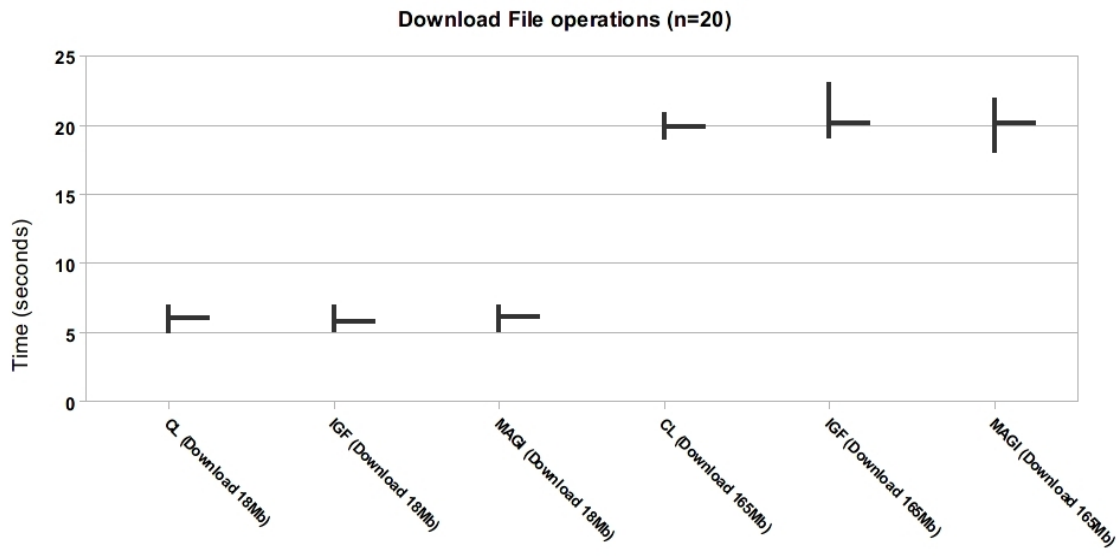


Figure 5.11: Chart comparing the maximum, minimum and average times of each of the file download tests.

5.6.2. Jobs execution

To illustrate the use of MAGI within the wider Grid environment, we tested two applications both locally and at the PIC (Port d'Informatió Cientifica) node at Barcelona. We were able to successfully integrate and run with MAGI, two brain imaging research applications, and run them on the Grid. One of the applications is called *association* and basically takes one fMRI analyse file as input, and generates the association maps between one input position (x,y,z), and other voxels in the given volume time series. The other application is called *summarize*, and generates a sum up of a multi volume analyse fMRI file.

To test the applications we stored the input test files both in our local storage at IEETA and replicated to the storage element in PIC (Port d'Informatió Cientifica) at Barcelona, Spain. When the jobs are created, the gLite middleware automatically selects the best CE to send the jobs, according to the proximity with the SE's, where the data input files defined in the job JDL file are stored. With this scenario we were able to obtain fairly good results from running both the *association* and *summarize* applications. Each of the applications took in average about eight minutes to perform the complete cycle (the job is submitted, scheduled in the selected CE, runs in a WN, and finally the results are transferred from the WN to the UI), from which of these eight minutes, six were spent during the running phase in the WN. Note that these average execution times were obtained with “perfect” conditions, which means, with the minimal waiting time in the queue (queue empty) and without any errors along the execution cycle. All the tests were performed with the *dteam* VO, in the EGEE infrastructure.

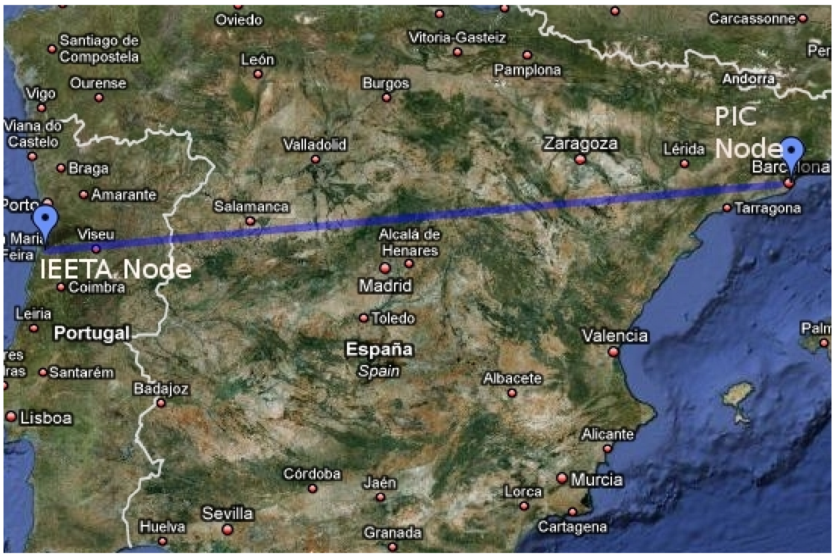


Figure 5.12: Map of the Grid nodes used during the tests.

6. Conclusions and Future Work

The presented MAGI web portal supported by the IGF framework, fulfills the proposed objectives of enabling non Grid experts to benefit from the Grid's processing power and scalable data management. The MAGI system similarly to other Grid oriented Web portal projects [8,56,75], provides a way for the non-experts users to access the Grid, in a transparent and easy way. However, not only it provides the same easy access to the Grid, but was also built with a semantic model that can accommodate a large variety of medical related research projects, with an attractive look and with intuitive and easy-to use functionalities, like drag-and-drop, to execute the main use cases.

In terms of implementation options, we decided to create the IGF framework, to isolate the MAGI system from the complexities of the Grid, shielding it from possible changes in the Grid middleware. The IGF also provides another alternative development framework to the Grid community. As demonstrated in the integration with the MAGI web portal, the IGF framework was flexible and easy to use, as proposed in its definition. Through IGF we created an isolation between the upper software layers of the MAGI and the Grid environment, being possible to access this environment, with just a few lines of code more, and even connect to different middleware stacks, in the future.

In IGF implementation, we decided to use the AMGA catalog to implement the sharing and description of stored Grid files, like in the gLibrary project [38]. The use of the AMGA catalog to support the sharing of information on the Grid as proven to be a good choice, since early tests using our framework have demonstrated that the response times for querying the catalog are low; this confirms other previous benchmark studies that have demonstrated that AMGA's overhead is very low, providing short response times [40].

In contrast to existing API's in the area of Grid storage, like GFAL [91] or SEE-GRID File Management Java API [92], that provide a more fine-grained and low-level programming interface (still lacking in documentation), IGF was created with the aim to provide a more high-level framework, like the Vine toolkit [36] (partially integrating other existing solutions, like GSAF [37]) that, while supporting our own efforts, could provide to other interested parties, an easy to use API with good documentation. We also plan to make our framework available, to public use as the system matures.

The IGF framework architecture was conceived having in mind a complete framework solution for the Grid, providing services for storage, computing and security. While the storage package of IGF is in a more advanced state of development, providing various services for storing and sharing files on the Grid, both the computing and security packages are still a work in progress. At this stage the computing package only supports basic job services, like creating and launching simple jobs, updating status, output retrieval and cancellation of jobs. As future work it would be very interesting to develop the support for more complex jobs, like DAGJobs (see gLite section in chapter 2.2.1.), since they enable the use of workflows, which is a mandatory feature in terms of medical imaging analysis. The security package is also a mandatory feature in terms of Grid environments, because providing security for scientific data, and especially medical research data is paramount. Further developing of this package would be an important aspect for the acceptance of the solution in production environments.. For the scope of this work, we decided to implement the security on the domain logic layer of the MAGI portal, providing pseudo-anonymization for the medical datasets. In the future, the security feature could be provided by the IGF Security package.

Regarding technological choices, the usage of object-relational mapping solutions based on Hibernate Annotations (using a PostgreSQL back-end database), proved to be a good choice for a quick and effective development of the data access application layer. The Hibernate Annotations allow a very organized and easy way to bind the domain model classes with the respective tables in the relational database. In terms of web interface development choices, the decision to use JavaServer Faces, combined with Richfaces, has proved to be a good choice for web development, since it allowed quick, intuitive and easy development of an advanced web based portal. This quick and easy development was also possible because of the good documentation and code examples that Richfaces provides. However the usage of this kind of web frameworks can also bring some disadvantages to the development process. If the developer wants to implement some functionality that requires a more specific approach that is not supported by the available framework controls, the development of new custom controls or the customization of the available ones can be a painful task.

In order to verify if the developed work achieved the proposed objectives, we performed tests using both the IGF and the MAGI portal, and compared the results with the usage of the CLI, which can be considered the “old way” of using Grids. The tests were run in a real production Grid, the EGEE Grid, using the *dteam* VO. The results obtained can be considered very positive. The users were able to use a web 2.0 portal to access the Grid, store, share, run scientific applications and retrieve the results, without having to know any technical aspects of the Grid itself. The results obtained also encourage to further develop both the MAGI portal, and especially the IGF framework, extending its services to support the execution of workflows and improved storage and security features.

References

[1] J. Montagnat, F. Bellet, H. Benoit-Cattin, V. Breton, L. Brunie, H. Duque, Y. Legré, I.E. Magnin, L. Maigne, S. Miguet, J.-. Pierson, L. Seitz, and T. Tweed, "Medical Images Simulation, Storage, and Processing on the European DataGrid Testbed," *Journal of Grid Computing*, vol. 2, Dec. 2004, pp. 387-400.

[2] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, K. Blackburn, A. Lazzarini, A. Arbree, R. Cavanaugh, and S. Koranda, "Mapping Abstract Complex Workflows onto Grid Environments," *Journal of Grid Computing*, vol. 1, Mar. 2003, pp. 25-39.

[3] K.K. Kakazu, L.W.K. Cheung, and W. Lynne, "The Cancer Biomedical Informatics Grid (caBIG): pioneering an expansive network of information and tools for collaborative cancer research," *Hawaii Medical Journal*, vol. 63, Sep. 2004, pp. 273-5.

[4] I. Foster and C. Kesselman, "Computational Grids," *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kauffman, 1999, pp. pages 15–51.

[5] "LHC_Homepage" , http://lhc.web.cern.ch/lhc/ [accessed April 29, 2009].

[6] H.B. Newman, M.H. Ellisman, and J.A. Orcutt, "Data-intensive e-science frontier research," *Commun. ACM*, vol. 46, 2003, pp. 68-77.

[7] J.P.S. Cunha, I. Oliveira, J.M. Fernandes, A. Campilho, M. Castelo-Branco, N. Sousa, and A. Sousa Pereira, "BING: The Portuguese Brain Imaging Network Grid," *IberGrid*, 2007.

[8] P. Kacsuk, G. Dózsa, J. Kovács, R. Lovas, N. Podhorszki, Z. Balaton, and G. Gombás, "P-GRADE: A Grid Programming Environment," *Journal of Grid Computing*, vol. 1, Jun. 2003, pp. 171-197.

[9] "EGEE: Enabling Grids for E-sciencE phase I and II, FP6 European IST project, contract number INFSO-RI-508833" , http://www.eu-egee.org/ [accessed February 26, 2009].

[10] T. Oreilly, "What Is Web 2.0: Design Patterns and Business Models for the Next Generation of Software," *SSRN eLibrary*.

[11] L. Paulson, "Building rich web applications with Ajax," *Computer*, vol. 38, 2005, pp. 14-17.

[12] I. Foster, "Service-Oriented Science," *Science*, vol. 308, May. 2005, pp. 814-817.

[13] "International Virtual Observatory Alliance" , http://www.ivoa.net/ [accessed April 29, 2009].

[14] X. Fu, T. Bultan, and J. Su, "Analysis of interacting BPEL web services," *Proceedings of the 13th international conference on World Wide Web*, New York, NY, USA: ACM, 2004, pp. 621-630.

[15] A.S. Szalay, J. Gray, and J. vandenBerg, "Petabyte Scale Data Mining: Dream or Reality?," Aug. 2002.

[16] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *International Journal of High Performance Computing Applications*, vol. 15, Aug. 2001, pp. 200-222.

[17] I. Foster and C. Kesselman, "Concepts and Architecture," *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kauffman, 2005, pp. pages 37–63.

[18] I. Foster, C. Kesselman, and S. Tuecke, "The Open Grid Services Architecture," *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kauffman, 2005.

[19] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, "The physiology of the grid: An open grid services architecture for distributed systems integration, Open Grid Service Infrastructure WG, Global Grid Forum," 2002.

[20] R. Alfieri, R. Cecchini, V. Ciaschini, Á. Frohner, A. Gianoli, K. Lőrentey, and F. Spataro, "An Authorization System for Virtual Organizations," *in Proceedings of the 1st European Across Grids Conference, Santiago de Compostela*, 2003, pp. 13--14.

[21] R. Housley, W. Ford, W. Polk, and D. Solo, "Internet X.509 Public Key Infrastructure Certificate and CRL Profile," 1999.

[22] "The Globus Alliance" , http://www.globus.org/ [accessed April 14, 2009].

[23] I. Foster, "Globus Toolkit Version 4: Software for Service-Oriented Systems," *Journal of Computer Science and Technology*, vol. 21, Jul. 2006, pp. 513-520.

[24] D. Erwin and D. Snelling, "UNICORE: A Grid Computing Environment," *Euro-Par*

2001 Parallel Processing, 2001, pp. 825-834.

[25] E. Laure, C. Gr, S. Fisher, A. Frohner, P. Kunszt, A. Krenek, O. Mulmo, F. Pacini, F. Prelz, J. White, M. Barroso, P. Buncic, R. Byrom, L. Cornwall, M. Craig, A. Di Meglio, A. Djaoui, F. Giacomini, J. Hahkala, F. Hemmer, S. Hicks, A. Edlund, A. Maraschini, R. Middleton, M. Sgaravatto, M. Steenbakkers, J. Walk, and A. Wilson, "Programming the Grid with gLite," *Computational Methods in Science and Technology*, vol. 12, 2006, p. 2006.

[26] "EGEE-Project. Job description language (jdl) attributes specification." , Available at https://edms.cern.ch/document/590869/1/ [accessed April 12, 2009].

[27] F. Pacini, "EGEE User's Guide - WMS Service." , https://edms.cern.ch/document/572489/ [accessed April 12, 2009].

[28] S. Andreozzi, S. Burke, L. Field, S. Fisher, B. Konya, M. Mambelli, J. Shopf, M. Viljoen, and A. Wilson, "GLUE Schema Specification-Version 1.2," Technical report. 2005.

[29] "AMGA: The gLite Grid Metadata Catalogue" , http://amga.web.cern.ch/amga/ [accessed February 21, 2009].

[30] "OGSA-DAI" , http://www.ogsadai.org.uk/ [accessed April 15, 2009].

[31] C. Catlett, "Standards for grid computing: Global grid forum," *Journal of Grid Computing*, 2003, pp. 1(1):3–7.

[32] "Accelerating the adoption of grid solutions in the enterprise, available at http://www.ogf.org," Dec. 2004.

[33] H.G.O.K. Czajkowski, I.F. Anl, J.F. Ibm, C.K. Usc/isi, D. Snelling, F. Labs, and P.V. Nasa, "GWD-R (draft-ggf-ogsi-gridservice-23) Editors: Open Grid Services Infrastructure (OGSI) S. Tuecke, ANL."

[34] I. Foster, K. Czajkowski, D. Ferguson, J. Frey, S. Graham, T. Maguire, D. Snelling, and S. Tuecke, "Modeling and Managing State in Distributed Systems: The Role of OGSI and WSRF," *Proceedings of the IEEE*, vol. 93, 2005, pp. 604-612.

[35] "OASIS: Advancing open standards for the global information society" , http://www.oasis-open.org/home/index.php [accessed April 5, 2009].

[36] M. Russell, P. Dziubecki, P. Grabowski, M. Krysiński, T. Kuczyński, D. Szjenfeld, D. Tarnawczyk, G. Wolniewicz, and J. Nabrzyski, "The Vine Toolkit: A Java Framework for Developing Grid Applications," *Parallel Processing and Applied Mathematics*, 2008, pp. 331-340.

[37] S. Scifo, "GSAF Grid Storage Access Framework," *Enabling Technologies: Infrastructure for Collaborative Enterprises*, 2007, pp. 296-297.

[38] A. Calanducci, C. Cherubino, L. Ciuffo, M. Fargetta, and D. Scardaci, “A Digital Library Management System for Grid,” *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2007. WETICE 2007. 16th IEEE International Workshops*, 2007, pp. 269-272.

[39] “gLite, (Lightweight Middleware for Grid Computing)” , http://glite.web.cern.ch/glite/ [accessed February 21, 2009].

[40] N. Santos and B. Koblitz, “Distributed Metadata with the AMGA Metadata Catalog,” Apr. 2006.

[41] C. Costa, A. Silva, and J. Oliveira, “Current Perspectives on PACS and a Cardiology Case Study,” *Advanced Computational Intelligence Paradigms in Healthcare-2*, 2007, pp. 79-108.

[42] The Editors, “Looking Back on the Millennium in Medicine,” *N Engl J Med*, vol. 342, Jan. 2000, pp. 42-49.

[43] R. Acharya, R. Wasserman, J. Stevens, and C. Hinojosa, “Biomedical imaging modalities: a tutorial,” *Computerized Medical Imaging and Graphics: The Official Journal of the Computerized Medical Imaging Society*, vol. 19, pp. 3-25.

[44] J.D. Bronzino, *The Biomedical Engineering Handbook*, 2000.

[45] R.A. Geenes and James F. Brinkley, *Medical Informatics: Computer Applications in Health Care and Biomedicine*, Springer, 2001.

[46] R. Andrade, “Multi-voxel fMRI Analysis Using an High Throughput Grid Framework,” Universidade de Aveiro, 2007.

[47] J. Engel, “Update on surgical treatment of the epilepsies: Summary of The Second International Palm Desert Conference on the Surgical Treatment of the Epilepsies (1992),” *Neurology*, vol. 43, Aug. 1993, p. 1612.

[48] F. Rosenow and H. Luders, “Presurgical evaluation of epilepsy,” *Brain*, vol. 124, Sep. 2001, pp. 1683-1700.

[49] I.C. Oliveira, J.M. Fernandes, L. Alves, A. Sousa Pereira, and J.P.S. Cunha, “GERES-med : An Architecture for Grid-Enabled scientific REpositorieS for medical applications,” *Proceedings of IberGrid*, 2008.

[50] V. Breton, I. Blanquer, V. Hernandez, Y. Legré, and T. Solomonidés, “Proposing a roadmap for HealthGrids,” 2006.

[51] I. Andoulsi, I. Blanquer, V. Breton, A. Dobrev, C. van Doosselaere, V. Hernandez, J. Herveg, N. Jacq, Y. Legré, M. Olive, H. Rahmouni, T. Solomonides, K. Stroetmann, V. Stroetmann, and P. Wilson, “The SHARE road map: Healthgrids for biomedical research and healthcare,” *Studies in Health Technology and Informatics*, vol. 138,

2008, pp. 238-78.

[52] V. Breton, K. Dean, T. Solomonides, I. Blanquer, V. Hernandez, E. Medico, N. Maglaveras, S. Benkner, G. Lonsdale, S. Lloyd, K. Hassan, R. McClatchey, S. Miguet, J. Montagnat, X. Pennec, W. De Neve, C. De Wagter, G. Heeren, L. Maigne, K. Nozaki, M. Taillet, H. Bilofsky, R. Ziegler, M. Hoffman, C. Jones, M. Cannataro, P. Veltri, G. Aloisio, S. Fiore, M. Mirto, I. Chouvarda, V. Koutkias, A. Malousi, V. Lopez, I. Oliveira, J.P. Sanchez, F. Martin-Sanchez, G. De Moor, B. Claerhout, and J.A.M. Herveg, "The Healthgrid White Paper," *Studies in Health Technology and Informatics*, vol. 112, 2005, pp. 249-321.

[53] "Embrace Network of Excellence" , Available at http://www.embracegrid.info/page.php?page=home [accessed April 13, 2009].

[54] J.K. Iglehart, "The New Era of Medical Imaging -- Progress and Pitfalls," Jun. 2006.

[55] J. Montagnat, D. Jouvenot, C. Pera, Á. Frohner, P. Kunszt, B. Koblitz, N. Santos, and C. Loomis, "Bridging clinical information systems and grid middleware: a Medical Data Manager," *Challenges and Opportunities of HealthGrids: Proceedings of Healthgrid 2006*, 2006.

[56] "BIRN - Biomedical Informatics Research Network" , http://www.nbirn.net/ [accessed March 4, 2009].

[57] J. Saltz, S. Oster, S. Hastings, S. Langella, T. Kurc, W. Sanchez, M. Kher, A. Manisundaram, K. Shanbhag, and P. Covitz, "caGrid: design and implementation of the core architecture of the cancer biomedical informatics grid," *Bioinformatics*, vol. 22, Aug. 2006, pp. 1910-1916.

[58] V. Breton, I. Blanquer, V.H. Hernandez, N. Jacq, Y. Legré, M. Olive, and T. Solomonides, "Roadmap for a European Healthgrid," *HealthGrid*, 2007.

[59] S. Hung, T. Hung, and J. Juang, "SARS Grid--An AG-Based Disease Management and Collaborative Platform," *Challenges and Opportunities of HealthGrids: Proceedings of Healthgrid 2006*, 2006.

[60] J. Montagnat, Á. Frohner, D. Jouvenot, C. Pera, P. Kunszt, B. Koblitz, N. Santos, C. Loomis, R. Texier, D. Lingrand, P. Guio, R. Brito Da Rocha, A. Sobreira de Almeida, and Z. Farkas, "A Secure Grid Medical Data Manager Interfaced to the gLite Middleware," *Journal of Grid Computing*, vol. 6, Mar. 2008, pp. 45-59.

[61] J. Duncan, "Imaging and epilepsy," *Brain*, vol. 120, Feb. 1997, pp. 377, 339.

[62] M. Richardson, "Epilepsy and surgical mapping," *Br Med Bull. 65*, 2003, pp. 179-192.

[63] P. Vemuri, J.L. Gunter, M.L. Senjem, J.L. Whitwell, K. Kantarci, D.S. Knopman, B.F. Boeve, R.C. Petersen, and C.R.J. Jr, "Alzheimer's disease diagnosis in individual

subjects using structural MR images: Validation studies," *NeuroImage*, vol. 39, Feb. 2008, pp. 1186-1197.

[64] J.L. Whitwell, M.M. Shiung, S.A. Przybelski, S.D. Weigand, D.S. Knopman, B.F. Boeve, R.C. Petersen, and C.R. Jack, "MRI patterns of atrophy associated with progression to AD in amnestic mild cognitive impairment," *Neurology*, vol. 70, Feb. 2008, pp. 512-520.

[65] A. Hamalainen, S. Tervo, M. Grau-Olivares, E. Niskanen, C. Pennanen, J. Huuskonen, M. Kivipelto, T. Hanninen, M. Tapiola, M. Vanhanen, M. Hallikainen, E. Helkala, A. Nissinen, R. Vanninen, and H. Soininen, "Voxel-based morphometry to detect brain atrophy in progressive mild cognitive impairment," *NeuroImage*, vol. 37, Oct. 2007, pp. 1122-1131.

[66] C. Horenstein, M.J. Lowe, K.A. Koenig, and M.D. Phillips, "Comparison of unilateral and bilateral complex finger tapping-related activation in premotor and primary motor cortex," *Human Brain Mapping*, vol. 30, 2009, pp. 1397-1412.

[67] T. Hanakawa, M.A. Dimyan, and M. Hallett, "Motor Planning, Imagery, and Execution in the Distributed Motor Network: A Time-Course Study with Functional MRI," *Cereb. Cortex*, Mar. 2008, p. bhn036.

[68] A. Michell, A. Goodman, A. Silva, S. Lazic, A. Morton, and R. Barker, "Hand tapping: A simple, reproducible, objective marker of motor dysfunction in Huntington's disease," *Journal of Neurology*, vol. 255, 2008, pp. 1145-1152.

[69] J. Jovicich, S. Czanner, D. Greve, E. Haley, A.V.D. Kouwe, R, Y. Gollub, D. Kennedy, F. Schmitt, G. Brown, J. MacFall, B. Fischl, and A. Dale, "Reliability in multi-site structural MRI studies: Effects of gradient non-linearity correction on phantom and human data," *NeuroImage*, vol. 30, Apr. 2006, pp. 436-443.

[70] Y. Wang, R. Schultz, R. Constable, and L. Staib, "Nonlinear Estimation and Modeling of fMRI Data Using Spatio-temporal Support Vector Regression," *Information Processing in Medical Imaging*, 2003, pp. 647-659.

[71] R. Andrade, I. Oliveira, J.M. Fernandes, and J.P.S. Cunha, "A Grid Framework for Non-Linear Brain fMRI Analysis,"
Proceedings of HealthGrid, Geneva. 2007.

[72] "SPM - Statistical Parametric Mapping" , http://www.fil.ion.ucl.ac.uk/spm/ [accessed May 3, 2009].

[73] "FSL" , http://www.fmrib.ox.ac.uk/fsl/ [accessed May 4, 2009].

[74] "Sun Grid Engine: Home" , http://gridengine.sunsource.net/ [accessed June 2, 2009].

[75] J. Montagnat, A. Gaignard, D. Lingrand, J.R. Balderrama, P. Collet, and P. Lahire, "NeuroLOG: a community-driven middleware design," *Studies in Health Technology*

and Informatics, Global Healthgrid: e-Science Meets Biomedical Informatics - Proceedings of HealthGrid 2008, 2008.

[76] "neuGRID" , http://www.neugrid.eu/pagine/home.php [accessed May 26, 2009].

[77] J. Geddes, S. Lloyd, A. Simpson, M. Rossor, N. Fox, D. Hill, J. Hajnal, S. Lawrie, A. Mclntosh, E. Johnstone, J. Wardlaw, D. Perry, R. Procter, P. Bath, and E. Bullmore, "NeuroGrid: using grid technology to advance neuroscience," *Computer-Based Medical Systems, 2005. Proceedings. 18th IEEE Symposium on*, 2005, pp. 570-572.

[78] M. Fowler and K. Scott, *UML distilled: applying the standard object modeling language*, Addison-Wesley Longman Ltd., 1997 , http://portal.acm.org/citation.cfm?id=270005 [accessed June 23, 2009].

[79] K. Arnold, J. Gosling, and D. Holmes, *Java(TM) Programming Language, The (4th Edition)*, Addison-Wesley Professional, 2005 , http://portal.acm.org/citation.cfm?id=1051069 [accessed June 23, 2009].

[80] G. Booch, R. Maksimchuk, M. Engle, B. Young, J. Conallen, and K. Houston, *Object-oriented analysis and design with applications, third edition*, Addison-Wesley Professional, 2007 , http://portal.acm.org/citation.cfm?id=1407387 [accessed June 23, 2009].

[81] E. Gamma, R. Helm, R. Johnson, and J.M. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, 1994.

[82] "jlite Java API" , http://code.google.com/p/jlite/ [accessed April 18, 2009].

[83] J. Novotny, S. Tuecke, and V. Welch, "An online credential repository for the Grid: MyProxy," 2001, pp. 104-111.

[84] T. Glatard, J. Montagnat, D. Lingrand, and X. Pennec, "Flexible and Efficient Workflow Deployment of Data-Intensive Applications On Grids With MOTEUR," *International Journal of High Performance Computing Applications*, vol. 22, Aug. 2008, pp. 347-360.

[85] "JavaServer Faces Technology" , http://java.sun.com/javaee/javaserverfaces/ [accessed March 3, 2009].

[86] "jBoss Richfaces" , http://www.jboss.org/jbossrichfaces/ [accessed March 3, 2009].

[87] D. Distante, P. Pedone, G. Rossi, and G. Canfora, "Model-Driven Development of Web Applications with UWA, MVC and JavaServer Faces," *Web Engineering*, 2007, pp. 457-472.

[88] "PostgreSQL: The world's most advanced open source database" , http://www.postgresql.org/ [accessed March 3, 2009].

[89] "hibernate.org - Java Persistence with Hibernate" , http://www.hibernate.org/397.html

[accessed March 3, 2009].

[90] I. Blanquer, V. Hernandez, D. Segrelles, and E. Torres, "Enhancing Privacy and Authorization Control Scalability in the Grid Through Ontologies," *Information Technology in Biomedicine, IEEE Transactions on*, vol. 13, 2009, pp. 16-24.

[91] "GFAL Java API" , https://grid.ct.infn.it/twiki/bin/view/GILDA/APIGFAL [accessed March 1, 2009].

[92] "SEE-GRID File Management Java API - EGEE" , http://wiki.egee-see.org/index.php/SEE-GRID_File_Management_Java_API [accessed March 1, 2009].