Grant agreement no. 211714

neuGRID

A GRID-BASED e-INFRASTRUCTURE FOR DATA ARCHIVING/ COMMUNICATION AND COMPUTATIONALLY INTENSIVE APPLICATIONS IN THE MEDICAL SCIENCES

Combination of Collaborative Project and Coordination and Support Action

Objective INFRA-2007-1.2.2 - Deployment of e-Infrastructures for scientific communities

Deliverable reference number and title: **D7.1 Test-bed Installation and API Documentation**

Due date of deliverable: **Month 12**

Actual submission date: **31st January 2009**

Start date of project: **February 1st 2008** Duration: **36 months**

Organisation name of lead contractor for this deliverable: **maat Gknowledge**

Revision: **Version 1**

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013)		
Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Table of contents

Preface

The neuGRID project (http://www.neugrid.eu) aims at developing a new user-friendly Grid-based research e-Infrastructure enabling the European neuroscience community to carry out research required for the pressing study of degenerative brain diseases. In neuGRID, the collection/archiving of large amounts of imaging data will be paired with computationally intensive data analyses. Neuroscientists will be able to identify neurodegenerative disease markers through the analysis of 3D magnetic resonance brain images via the provision of sets of distributed medical and Grid services.

The work presented in this deliverable contributes to the project in the following two areas:

- Installation of the neuGRID test-bed: the so-called PoC (Proof-Of-Concept) environment.
- EGEE gLite grid middleware functionality exposure within the neuGRID Platform (c.f. glossary).

1 Overview

This document aims to report on both the design and development of the neuGRID grid test-bed, and the description of the Application Programming Interface (API) that is delivered to subsequently access it. The prototype grid test-bed is based on the gLite grid middleware [1] from the European Enabling the Grid for E-sciencEs (EGEE) project [2].

This work package (WP7) has the following major objectives:

- To evaluate and if necessary develop appropriate grid interfaces to gLite services, to programmatically make the grid functionality available to developers,
- To undergo education and training in the use of gLite services and its APIs,
- To ensure updates and releases of gLite are available, to test them in an isolated environment prior to deploying those in the production infrastructure,
- To liaise with EGEE and enrich the grid middleware requirements from new ones identified in neuGRID.

WP7 builds upon the project requirements elicitation conducted by WP9. As "D9.1 User Requirements Specification (USR) document first release" is due at Month 14, the following anticipates the finalisation of the tes-bed.

To support the neuGRID software development cycles, trial deployments, evaluation and experimentation with the neuGRID-specific gLite middleware configurations, two sub-infrastructures PoC (Proof-of-Concept)[1] and PROD (Production) were identified. The main goal of the gLite deployment was to expose the grid middleware functionality within the neuGRID platform (see the Glossary section). We decided to build our own PoC infrastructure in order to better understand how the gLite components interact with each other and also to train the WP8 team members for the deployment and delivery of the neuGRID PROD infrastructure. This work also allowed us to deploy and test latest gLite releases and other software components which were needed to support the neuGRID development with project-specific functionality. It was thus decided to mimic as much as possible real PROD sites in the test-bed.

A set of APIs acting as a grid interface to gLite services for the neuGRID platform and client applications was investigated, with the aim of delivering a service-based access.

The deliverable is organised as follows. First, we present briefly the high level requirements of the neuGRID project that helped making decisions on what to deploy within the PoC infrastructure and to set plans for later deployments in the PROD environment. We then provide an overview of the gLite grid middleware functionality. The neuGRID grid infrastructure - PoC (architecture, current status, etc) is described in the section 4**Erreur ! Source du renvoi introuvable.**. The Grid API documentation is then described in the section 5. Finally, we summarise the work conducted within the first 12 months in WP7.

To improve readers' understanding of the document and technical terms, we provide "Bibliography" and "Glossary" sections where links to more extensive project specific or technical information can be found as well as acronyms used throughout the text.

[1] The PoC infrastructure is neuGRID's test-bed.

2 neuGRID Grid Middleware Requirements

gLite is used for building distributed federated infrastructures for Virtual Organizations (VO)[2] bringing together different kinds of geographically distributed resources allowing (among other things) for sharing and exchanging information in a secure way. It is a broad and complex set of software services which in turn comprises a number of components. To actually build and deploy the grid-based infrastructural solution in WP7, the neuGRID infrastructure architects and neuGRID developers had to understand the requirements of the project end-users and software development groups. The requirements thus far gathered span multiple aspects ranging from security and privacy constraints, to networking and hardware needs and to the structure of the project's virtual organization itself. For that purpose, the project had to and still is conducting a series of requirements elicitation meetings with respective end-users groups. At the outset of this process key documents will be delivered including (but not limited to):

- "Users Requirements Specifications – first release" document [3] Month 14;
- "Users Requirements Specifications – final release" document [4] Month 26.

Realization of the project goals (i.e. to deploy e-infrastructures for European neuro-science community, to collect/archive a large amounts of imaging data, computationally intensive data analyses, etc.), requires an infrastructure that is highly dependable and reliable. Physicians may require guarantees that the system is always available and that the processes that integrate data are reliable, even in the case of failures. Moreover, the infrastructure may have to allow for the transparent access to distributed data, to provide a high degree of scalability, and to efficiently schedule access to computationally intensive services by applying sophisticated load-balancing strategies.

WP9, for the sake of accompanying developments, already compiled a set of requirements and circulated internal documents giving early access to the information. These requirements were kept in mind while designing and creating the neuGRID infrastructure. According to these requirements, the system shall provide a distributed computing infrastructure for storing imaging data and results of computationally intensive data analysis. The system uses gLite for virtualizing distributed computing resources and enabling secure access to sensitive medical data. The grid middleware shall provide secure, coordinated and controlled access to the distributed computing resources. The grid middleware shall enable to create/destroy/modify Virtual Organizations to allow co-working between physicians of the neuro-science community. It shall be possible to manage the grid infrastructure allowing adding, removing and modifying nodes on the grid.

As can be noticed, such requirements are high level and quite general, thus inviting WP7 partners to build a flexible enough infrastructure which can be tuned in the near future to align with WP9 conclusions.

[2] A Virtual Organization is a collection of individuals and institutions that is defined according to a set of resource sharing rules [12]

3 EGEE Grid Middleware

3.1 Overview

Grid Middleware refers to the security, resource management, data access, instrumentation, policy, accounting, and other services required for applications, users, and resource providers to operate effectively in a grid environment. Middleware acts as a sort of 'glue' which binds these services together. Grid middleware is built by layered interacting packages.

- A grid middleware is an internet based system that needs efficient and reliable communication and is a blend of high performance systems and high throughput computing,
- A grid middleware is data aware and all data access and replications decisions are based onbase bound at least for the following functions: grid topology management user access and certification dataset locations and replicas resource definition and dynamical management performance and user bookkeeping,
- A grid middleware is bound to efficient matching and scheduling algorithms to find best available resources for the task execution and resource brokering,
- A grid middleware depends on accurate clock performances to synchronize nodes and correctly handle task and job scheduling.

The gLite distribution [1] is an integrated set of components designed to enable resource sharing. In other words, this is a middleware for building a grid. It is developed by the EGEE European project. gLite distributions pull together contributions from many other projects/partners, including LCG and VDT. The distribution model is to construct different services ('node-types') from these components and then to ensure simple installation and configuration on the chosen platforms (i.e. currently Scientific Linux version 4).

The gLite Middleware is a quite complex framework which follows a Service Oriented Architecture (SOA) to facilitate interoperability among grid services and to allow easier compliance with upcoming standards. This architecture has the advantage to not be bound to specific implementations, and services are expected to work together but can also be used independently.

3.2 gLite Middleware Architecture

This section will give a short overview of the components currently available in the EGEE gLite middleware. Figure 3-1 below depicts the gLite high level services, which can thematically be grouped into five service groups: Access Services, Information and Monitoring Services, Job Management Services, Data Management Services and Security Related Services. For more detailed descriptions refer to [5].

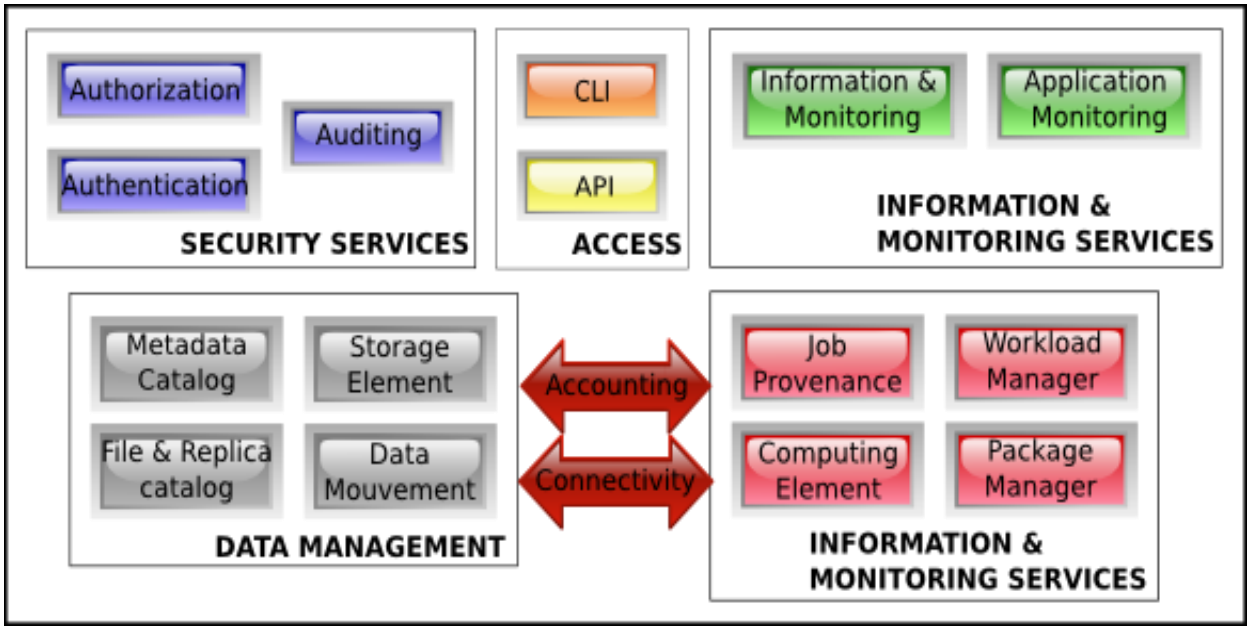

Figure 3-1: The gLite high level architecture

The following subsections briefly present gLite middleware components grouped by service types. Only major gLite components considered to be relevant to the scope of the neuGird project are listed and described.

3.2.1 Access Services

The aim of the Access Services is to give either command line or programmatic level access to the whole stack of publicly available gLite services. As an example, Figure 3-2 below presents schematic view of gLite Data Management APIs and Command Line Interfaces (CLIs).

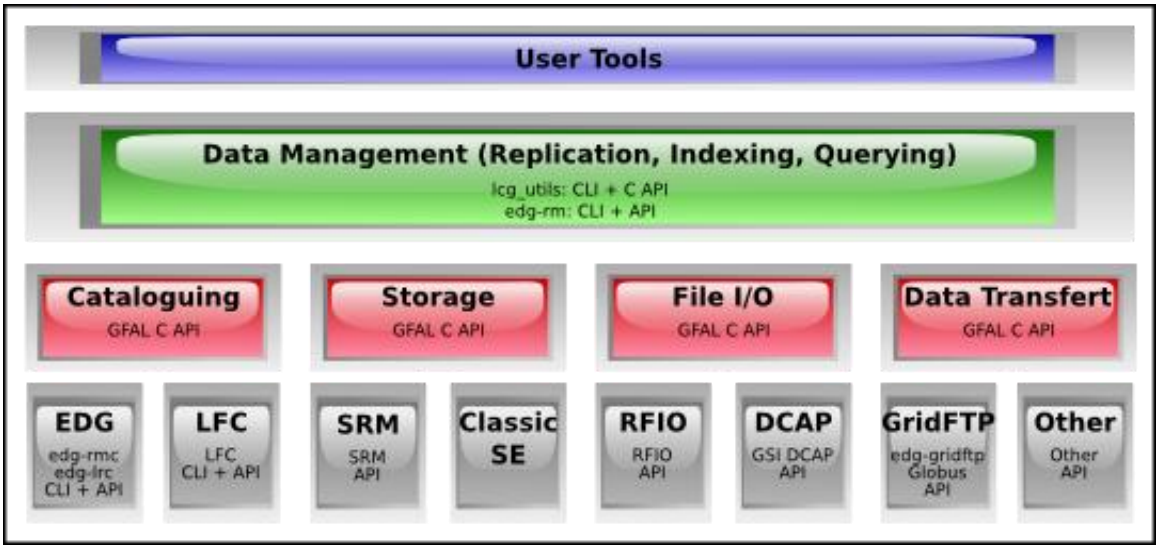

Figure 3-2: gLite Data Management APIs and CLIs

User Interface (UI) is gLite's separate deployment component which belongs to Access Services group. When installed on a server or desktop computer it allows users or user level applications to access the functionalities of the grid services like VOMS, WMS, File Catalog, SE, CE, IS etc. It provides a set of CLIs and different programming languages APIs.

3.2.2 Security Services

Security Services encompass the Authentication, Authorization, and Auditing services which enable the identification of entities of different nature (i.e. users, systems, and services), allow or deny access to services and resources, and provide information for post-mortem analysis of security related events. To carry out the tasks of Authentication and Authorization, gLite uses the Public Key Infrastructure (PKI) x509 technology using Certificate Authorities (CAs) as trusted third parties and MyProxy [6] extended by VOMS. It also provides functionality for data confidentiality and a dynamic connectivity service (i.e. means for a site to control network access patterns of applications and Grid services utilizing its resources).

VOMS: The Virtual Organisation Membership Service (VOMS) is a service to manage authorization information in a VO scope. The VOMS system should be used to include VO membership and any related authorization information in a user's proxy certificate. These proxies will be said to have VOMS extensions. The user gives the voms-proxy-init command instead of grid-proxy-init, and a VOMS server will be contacted to check the user's certificate and create a proxy certificate with VOMS information included. By using that certificate, the VO of a user will be present in every action that he will perform.

Hydra: is a gLite implementation of secure key storage. Symmetric encryption keys for encrypted files are stored in a specific set of servers called Hydra. Hydra provides controlled access to these keys (through certificate DN and VOMS attributes based ACLs) and secured communication to the requester. Hydra uses Shamir's secret-sharing scheme for splitting keys into 'n' fragments stored in different places. Only 'm<n' fragments are needed to reconstruct a complete key. However, owning less than 'm' key fragments, does not give any information on the complete key. Thus, the system is both resistant to attacks (at least 'm' key stores need to be compromised for an attacker to be able to reconstruct the key) and reliable (the disconnection of a limited number of servers does not prevent the key reconstruction).

Encrypted storage: Users access Hydra through the Encrypted storage C library which provides on-the-fly; block level data encryption and decryption. The component provides command line utilities for managing the keys in the Hydra key store. There are also command-line utilities, which integrate the library with the gLite I/O clients, thus one can retrieve/decrypt or store/encrypt files transparently.

Grid Policy BOX (G-PBox): is a policy framework designed to operate on Grid environments. G-PBox is a tool for VO and site administrators. For VO administrators it allows writing policies for internal VO groups/roles defined in the VO VOMS server and to manage policies received from site G-PBoxes. In turn, by site administrators G-PBox is used write policies for internal sites users and to manage policies received from VO G-PBoxes. G-PBox is queried by resources like CEs and SEs and services (as WMS) also not owned by VOs or Sites. Simply speaking, it helps in creation and application of authentication policies between grid services - for example, between VO WMSes and site CEs and SEs.

MyProxy (PX)[3]: On grids, users authenticate themselves using temporary credentials called proxy certificates, which contain also the corresponding private key. Proxy certificates do not represent a significant security risk only if they are reasonably short-lived (by default, a dozen hours). For longer jobs, PX plays a role of online credential repository [6]. For such long running jobs a proxy renewal system is used, consisting of a **Proxy Renewal Service (PRS)** on the RB and a PX server on a dedicated host. A PX stores long-lived user proxies (with a lifetime of several days, usually) which it uses to generate, on request of the PRS, short-lived proxies for jobs whose proxies are about to expire.

[3] in gLite this service comes with VDT [7] distribution

3.2.3 Information and Monitoring Services

- **Information and Monitoring Services (IMS)** provide a mechanism to publish and consume information and to use it for monitoring purposes. The information and monitoring system can be used directly to publish, for example, information concerning the resources on the Grid. More specialized services, such as the Job Monitoring Service and Network Performance Monitoring services can be built on top. Major components of gLite IMS are: BDII, R-GMA and Service Discovery.
- **BDII:** The Information System (IS) provides information about the status of Grid services and available resources. Job and data management services publish their status through Grid Resource Information Server (GRIS). GRIS runs on every service node and is implemented using OpenLDAP, an open source implementation of the Lightweight Directory Access Protocol (LDAP). Ever grid site also runs one Grid Index Information Server (GIIS). The GIIS queries the service GRISes on the site and acts as a cache storing information about all available site services. Finally, the information from GIISes is collected by Berkeley Database Information Index (BDII). The BDII (also called top BDII: tBDII) queries the GIISes (sometimes also called site BDII: sBDII) and acts as a cache storing information about all available Grid services in its database. Figure 3-3 shows this flow of information. Users and programs interested in status of the Grid usually query the top level BDII as it contains information about all the services that are available.

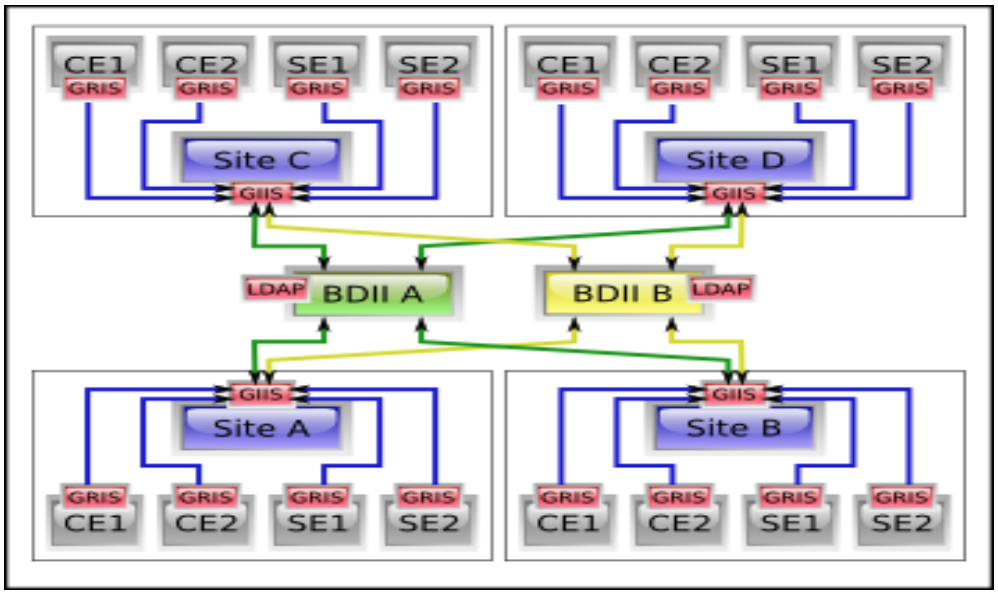

Figure 3-3: gLite Information System hierarchy

- **R-GMA** is a relational implementation of the Grid Monitoring Architecture defined by GGF. It is information and monitoring system for use both by the Grid middleware and by applications. Producer and consumer services are available at every site and currently make use of a single registry and schema service. R-GMA is currently being used by job management services, operational tools and by users for monitoring their applications.
- **Service Discovery** is a facility for locating suitable services offered to both end users and other services. It is implemented as a client library front-end to one or more information systems. The information systems are made available by a plug-in mechanism. It is intended to be lightweight and simple to use. Currently it supports BDII, R-GMA and XML files as back-ends. Service Discovery is used by Workload and Data Management components. This component is also available on UI for users do discover services currently available on an

infrastructure.
- Grid services provide information about their status in a form defined by the Grid Laboratory for a Uniform Environment (GLUE) schema. GLUE schema is the result of an ad-hoc international collaboration. It is maintained in two forms: one for BDII and one for R-GMA. The information is logically the same but one follows a hierarchical schema (for BDII) and the other is relational (for R-GMA).

3.2.4 Data Management Services

Architecturally the Data Management Services of the gLite middleware stack consist of three major components: Data Storage, Metadata and Catalog Services and Data Scheduling.

3.2.4.1 Data Storage

- **Grid File Transfer Protocol (GridFTP)** is a high-performance, secure, reliable data transfer protocol optimized for high-bandwidth wide-area networks. It is based on the Internet FTP protocol, and it implements extensions for high-performance operation. GridFTP uses basic Grid security on both control (command) and data channels. Other features include multiple data channels for parallel transfers, partial file transfers, third-party (direct server-to-server) transfers, reusable data channels, and command pipelining. GridFTP is used as a primary data transfer interface to Storage Elements.
- **Storage Resource Managers (SRM)** is a service somewhat similar to the cluster batch system but instead of managing processors and jobs it manages requests for storage space and files. The storage space managed can be disk space, tape space or a combination of the two. There is a number of SRM implementations for disk storage management that are widely deployed (DPM, dCache, CASTOR).
- **Disk Pool Manager (DPM)** is a recommended solution for lightweight deployment of smaller sites because it is easy to install and requires very low maintenance effort. It features full implementation of SRM. Bigger sites usually choose dCache because of robustness, scalability and advanced features. CERN Advanced STORage (CASTOR) is an implementation used by sites that have both disk and tape storage.

3.2.4.2 Metadata and Catalog Services

- **LCG File Catalog (LFC)** offers a hierarchical view of files to users, with a UNIX-like client interface. The LFC provides Logical File Name (LFN) to Storage URL (SURL) mappings and authorization for file access. The LFNs are aliases created by a user to refer to actual data. Simple metadata can be associated to them. The authorization is performed using UNIX-style permissions, POSIX Access Control Lists (ACL) and VOMS support. The LFC uses a client-server model with a proprietary protocol. LFC server communicates with a database (either Oracle or MySQL), where all the data is stored. LFC catalogue also exposes a Data Location Interface (DLI) - a web service used by applications and Resource Brokers. Provided with a LFN, the DLI returns the actual location of the file replicas.

3.2.4.3 Data Scheduling

- **File Transfer service (FTS)** is a reliable, low-level data movement service for transferring files between Storage Elements. It also provides features for administration and monitoring these transfers. The FTS exposes an interface to submit asynchronous bulk requests and performs the transfers using either third-party GridFTP or SRM Copy.

3.2.5 Job Management Services

Three major components constituting the Job Management Services group are Computing Element, Workload Management and Accounting. Although primarily related to the job management services; accounting is a special case as it will eventually take into account not only computing, but also storage and network resources.

- **Computing Element (CE)** is the service representing a computing resource. It provides a virtualization of the computing resource localized at a site (typically a batch queue of a cluster but also supercomputers or even single workstations). It provides information about the underlying resource and offers a common interface to submit and manage jobs on the resource. CE includes: a Grid Gate (GG) - Gatekeeper for CE based on Globus - which acts as a generic interface to the cluster; LRMS (sometimes called batch system); the cluster itself - a collection of Worker Nodes (WN) or just one multiprocessor WN, the node(s) where the jobs are run. There are two GG implementations in gLite 3.0: the LCG-CE and the gLite-CE; sites can choose what to install. The GG is responsible for accepting jobs and dispatching them for execution on the WN(s) via the LRMS.
  Another type of CE developed in EGEE project is CREAM (Computing Resource Execution And Management). It is a simple, lightweight service that implements all the operations required at the CE level. Its interface is defined using WSDL. The service is compliant with the existing BES standard. CREAM can be used by the WMS, via the ICE component (see next description of WMS), or by a generic client, e.g. an end-user willing to directly submit jobs to a CREAM CE. A C++ command line interface and Java clients are available for this purpose.
  The interface of gLite-CE and CREAM with the underlying LRMS is implemented via BLAH. All the resource management systems supported by BLAH are automatically supported by the CEs. In gLite 3.0 the supported LRMS types are OpenPBS, LSF, Maui/Torque, BQS and Condor.

- **Workload Management System (WMS)** is a Grid level meta-scheduler that schedules jobs on the available CEs according to user preferences and several policies. It also keeps track of the jobs it manages in a consistent way. The core component of the WMS is the **Workload Manager (WM)**, whose purpose is to accept and satisfy requests for job management coming from its clients (i.e., computational job submission). In particular the meaning of the submission request is to pass the responsibility of the job to the WM. The WM will then pass the job to an appropriate Computing Element for execution, taking into account the requirements and the preferences expressed in the job description. The decision of which resource should be used is the outcome of a matchmaking process between submission requests and available resources. The Resource Broker (RB) or Matchmaker as WMS component offers support to the WM in taking the above mentioned decision. It provides a matchmaking service based on a given user's job description – find a resource that best match the request. A WMS instance interacts with several other services. Tracking job lifetime relies on the Logging and Bookkeeping Service. Information on service availability, resource status and data localization is gathered from appropriate sources, such as Service Discovery, LFC, BDII, RGMA. Security-related aspects are addressed interacting with VOMS, Proxy Renewal and G-PBox.
  Another important component of the gLite WMS is the Interface to CREAM Environment (ICE). It provides the connection between the gLite WMS and the CREAM CE. ICE, running in the gLite WMS node along with the other processes of the gLite WMS, receives job submissions and other job management requests from the WM component of the WMS and then invokes the appropriate CREAM methods to perform the requested operation.

- **Logging and Bookkeeping Service (LB):** The primary purpose of the Logging and Bookkeeping service (LB) is tracking Grid jobs as they are processed by various Grid middleware components. It collects and stores in a database the job status information supplied by the different components of the WMS system. The collection is done by LB local-loggers, which run on the RB and on the CE, while the LB server, which normally runs on the RB, saves the collected information in the database. The database can be queried by the user from the UI, and by RB services. The information that is gathered in LB is used to inform Grid users on the job state. This information is also useful for example for debugging of user jobs.

3.3 gLite Services Deployed for neuGRID

Based on initially gathered requirements and from the technical partners experience, a list of gLite services which have to be installed for the neuGRID project was determined.

- For the security area, a **VOMS** service will be needed to manage the neuGRID community. Also, a **MyProxy** service might be needed to handle long running jobs. Then, for data management, a **LFC** service might be needed (might not be used). Also, a **DPM** service will be needed to physically store the imaging data. For the gLite information system, we need to install a **tBDII** and **sBDII** services. Finally, for the computational power part of neuGRID, **CE/WN** and **WMS/LB** services will be installed in the coming months as progress is made on the gridification model implementation (see D10.1 for more details). All these services will be dispatched in the GCC and in the different DACS sites.

4 The neuGRID Infrastructure

Requirements for deployment of neuGRID infrastructure were overviewed in section 2 of the deliverable. Based on the above high level requirements and availability, functionality and properties of the services comprising EGEE Grid middleware stack architecture and deployment strategy of the neuGRID infrastructure were proposed.

To support neuGRID software development cycles, experimentation with neuGRID specific gLite middleware deployments, developers' and infrastructure administrators' trainings, 2 sub-infrastructures were identified:

- PoC (Proof-of-Concept)
    - evaluation of new gLite releases
    - experimentation with gLite deployment strategies
    - development and early testing of neuGRID services
    - training of neuGRID developers and infrastructure administrators
    - Pipelines and client applications development and integration
- PROD (Production) – stable and secure environment for end-users
    - Platform deployed and operational for the neuGRID community usage.

PoC (WP7 responsibility), PROD (WP8 responsibility) are two distinct non-overlapping sub-infrastructures forming the neuGRID network. The Figure 4-1 schematically represents them and also shows Grid middleware evaluation/validation/deployment and software development cycles on the neuGRID infrastructure.

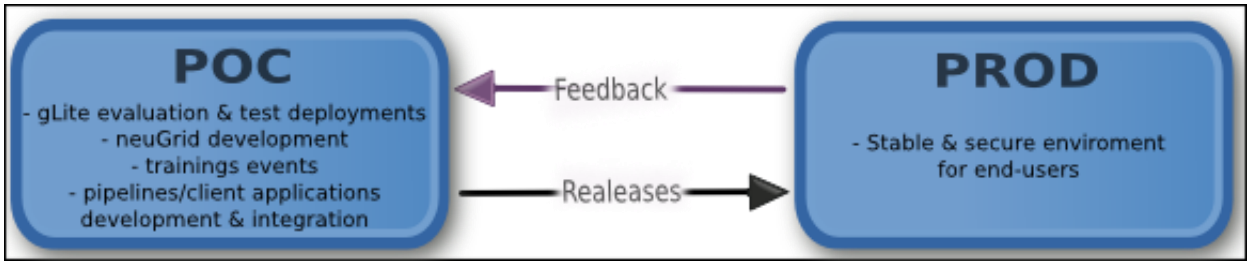

Figure 4-1: neuGRID infrastructure (schematic view).
Grid middleware validation/deployment and software development cycles.

In the next section the POC infrastructure will be presented and you must have in mind that it will be mimicked in the PROD infrastructure.

4.1 Certificate Authority (CA)

On grids public key infrastructure (PKI) arrangement is used for binding electronic cryptographic keys (key-pair) with respective user identity. In this scheme Certification Authority (CA) has a role of trusted third party and of high importance. It plays a vital role on the infrastructure while not directly being a part of the grid infrastructure.

The neuGRID project considered two possibilities of having cryptographic certificates:

- buying them from widely accepted CA like for example VeriSign [8],
- setting up an instance of CA belonging to neuGRID project only.

In order to be more flexible, it was decided that, in the PoC infrastructure, a specific neuGRID CA would be used. This CA is provided by MAAT (based on Open Source free software OpenCA). However, in the PROD infrastructure, all eu-gridpma CAs are meant to be accepted.

4.2 Test-bed Installation: PoC infrastructure

In the section 3.3, a list of gLite services that will be install in the neuGRID infrastructure was provided. Here is a recap of this list:

- VOMS
- MyProxy
- LFC
- WMS/LB
- tBDII
- sBDII
- DPM
- CE/VN

The 5 first services (VOMS, MyProxy, LFC, tBDII and WMS/LB) will be part of the GCC as core services of the neuGrid infrastructure. Indeed, they can be seen as gLite core services under which all other neuGRID services can be added / mixed. The VOMS server can be shared for the 2 sub-infrastructures (PoC, PROD) as it is able to deal with sub groups. This is also the case for MyProxy as it's only a proxy certificate repository into which each user can dump their own. But concerning LFC, tBDII and WMS/LB, there will have one instance of each service per sub-infrastructure to avoid all the potential resource sharing problems and to ease user access rights management. Concerning the other services (sBDII, DPM, CE/WN), they will be installed on each DACS to provide them with data storage and computing power capability. This approach is considered to be

a standard practice for gLite middleware deployment.

In the following figure (Figure 4-2), you can find a representation that emphasizes the gLite services that are deployed inside of the PoC, as of today.

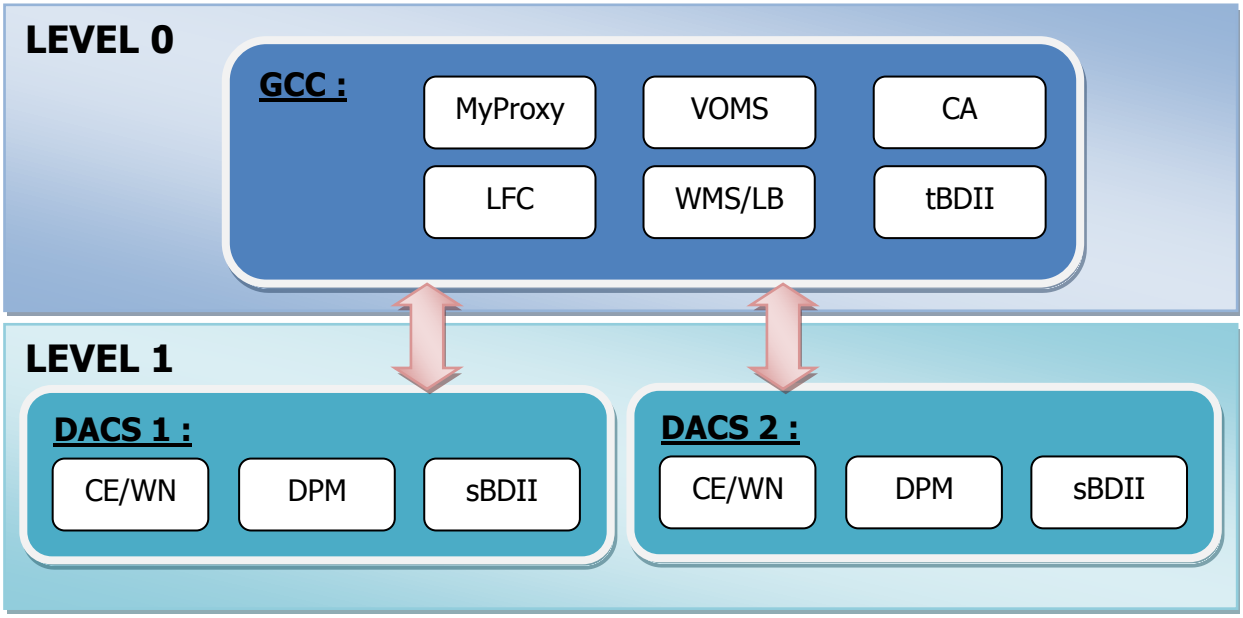

Figure 4-2: gLite services repartition inside POC

As it is described in the project description of work document, "LEVEL 0" represents the "infrastructure ground truth" level and "LEVEL 1" represents the Data Archiving and Computing Sites (DACS). The number of WN will depend on the available hardware at each DACS. The more hardware will be available, the more WNs will be installed to increase the DACS computing power capability.

In the context of the PoC deployment, two partners are members of the PoC infrastructure:

- **MAAT** which owns the GCC and one DACS,
- **PRODEMA** which own another DACS.

The current development status is the following for the GCC:

HOSTNAME/IP	SERVICES
ng-maat-server2.maat-g.com	VOMS
ng-maat-server4.maat-g.com	LFC + tBDII
ng-maat-server7.maat-g.com	MyProxy
ng-maat-server9.maat-g.com	WMS + LB
openca.ng-maat-server1.maat-g.com	CA

For the DACS mimicked by MAAT, the deployment is the following:

HOSTNAME/IP	SERVICES
ng-maat-server3.maat-g.com	DPM + sBDII
ng-maat-server8.maat-g.com	CE
ng-maat-server10.maat-g.com	WN
ng-maat-server11.maat-g.com	WN
ng-maat-server12.maat-g.com	WN

5 Grid API Documentation

5.1 Introduction

The gLite middleware is quite powerful in terms of functionality and one of the big challenges is to expose its functionality in a more user friendly manner. Natively, the gLite middleware provides all the necessary APIs in C/C++. Also it provides few java/Python APIs for some services. Usually, gLite is used through what is called a "gLite User Interface" (gLite-UI) [9]: this is a suite of clients (binaries) and APIs (mostly C/C++) that users and applications can use to access the gLite services.

In the context of the neuGRID project, most of the developments will be done in Java. By consequence, the use of the gLite-UI is feasible through for instance the use of the *java.lang.Runtime.exec()* java function [10].

The *java.lang.Runtime.exec()* method is able to run external programs within Java program and, so, would allow us to execute all the grid command line interfaces (CLIs). Never the less, this implementation would have two main disadvantages:

The first disadvantage is that this implementation requires a lot of memory. Indeed, each call to this kind of function will emulate a new environment into which the CLIs will be executed.

The second disadvantage is that this implementation forbids interactions between the caller of the method and the underlying process. Indeed, once an external program is lunched using this *java.lang.Runtime.exec()* method, you cannot interact anymore with it and if a CLI, for instance, ask for some confirmation before executing concretely the command, the program is stuck and will never finished.

Hopefully, since a couple of month, a new possibility to interact with the gLite middleware is now available through a well known APIs named Java Grid Application Toolkit (javaGAT) [11]. JavaGAT is a set of generic and flexible APIs for accessing grid services. It implements a plug-in architecture which allows the community to extend it to be able to access multiple grid middleware. Until the last release (2.0.3), javaGAT supported, GridLab, Globus, Zorilla, etc... but not gLite. This is now the case even if the support is not complete.

One of the big advantages of the javaGAT solution is that it is open source, which implies that it can be easily extended if needed. Also, the adoption of such abstraction layer could be interesting at some point for the neuGRID project interoperability. Indeed, once JavaGAT is integrated in your program, this later is able to deal with all the middleware that are supported by javaGAT.

5.2 The javaGAT APIs

The javaGAT APIs are divided in five main areas:

- **File operations**: allow file manipulation like copy, delete, move and create operations.
- **File stream operations**: allow to read or to write file content.
- **Job submissions**: allow to run, stop, cancel, etc. different job within the underlying Grid infrastructures.
- **Monitoring**: allow to monitor the previously launched jobs.
- **Access to information service**: allow to discover the available resources of the underlying Grid infrastructure.

In the next sections, we will describe the five areas more in detailed.

5.2.1 File Operations

One of the most important objects for this functionality is the *org.gridlab.gat.io.File* one. This object is an abstract representation of a physical file and is used to perform operations on an entire file. The file can be either local or remote. Thanks to this object, a lot of functionality is available. The most important ones are:

- **canRead**() - Detect whether a file or a directory can be read or not.
- **canWrite**() - Detect whether a file or a directory can be written or not.
- **copy**(URI loc) - Copy the file to a specific destination (URI).
- **createNewFile**() - Create the file.
- **delete**() - Delete the file.
- **exists**() - Test whether the file exists or not.
- **isDirectory**() - Test if the file object represent actually a directory.
- **isFile**() - Test if the file object represent actually a file.
- **lastModified**() - Test when the file was modified for the last time.
- **length**() - return the size of the file.
- **listFiles**() - List the files that are inside the provided directory.
- **mkdir**() - Create the directory.
- **mkdirs**() - Create the directory and all the parents if they don't exist.
- **move**(URI location) - Move the file to a specific location (URI).
- **recursivelyDeleteDirectory**() - Delete the directory and all his content.
- **renameTo**(File dest) - Rename the file.

5.2.2 File Stream

In this area, two classes are important: *org.gridlab.gat.io.FileInputStream* and *org.gridlab.gat.io.FileOutputStream*. The first one is mainly used to read the content of a file and the second one is used to write the content of a file. Those both interfaces can mange local or remote files.

The main important functions of the *org.gridlab.gat.io.FileInputStream* object are:

- **available**() - Returns the number of bytes that can be read from this file input stream without blocking.
- **close**() - Close the connection to the file.
- **mark**(int arg0) - Mark the current position in the input stream. A subsequent call to the reset method repositions this stream at the last marked position so that subsequent reads re-read the same bytes.
- **read**() - Reads a byte of data from this input stream.
- **read**(byte[] b) - Reads up to *b.length* bytes of data from this input stream into an array of byte.
- **read**(byte[] b, int off, int len) - Reads up to *len* bytes of data from this input stream into an array of bytes.
- **reset**() - Repositions this stream to the position at the time the *mark* method was last called on this input stream.
- **skip**(long n) - Skips over and discards *n* bytes of data from the input stream.

For the *org.gridlab.gat.io.FileOutputStream* object those are the most important methods:

- **close**() - Close the connection to the file.
- **flush**() - Flushes this output stream and forces any buffered output bytes to be written out.
- **write**(byte[] b) - Writes *b.length* bytes from the specified byte array to this output stream.
- **write**(byte[] b, int off, int len) - Writes *len* bytes from the specified byte array starting at offset *off* to this output stream.
- **write**(int b) - Writes the specified byte to this output stream.

5.2.3 Job Submissions

For job submission, the most interesting objects are *org.gridlab.gat.resources.ResourceBroker* and *org.gridlab.gat.resources.Job*.

An instance of the *ResourceBroker* class is used to submit jobs or to reserve resources for a future job and the Job object is here to monitor the job status.

The most relevant functions for the *ResourceBroker* object in our case are:

- **findResources**(ResourceDescription resourceDescription) - This method attempts to find one or more matching hardware resources which correspond to a given resource description.
- **reserveResource**(ResourceDescription resourceDescription, TimePeriod timePeriod) - This method attempts to reserve the specified resource for the specified time period.
- **reserveResource**(Resource resource, TimePeriod timePeriod) - This method attempts to reserve the specified resource for the specified time period.
- **submitJob**(JobDescription description) - This operation takes a JobDescription, and submits the specified job to some underlying resource management or allocation system.
- **submitJob**(JobDescription[] descriptions) - This operation takes an array of JobDescriptions, and submits the specified jobs to some underlying resource management or allocation system.
- **submitJob**(JobDescription[] descriptions, MetricListener listener, String metricDefinitionName) - This operation takes an array of JobDescriptions, and submits the specified jobs to some underlying resource management or allocation system.
- **submitJob**(JobDescription description, MetricListener listener, String

metricDefinitionName) - This operation takes a JobDescription, and submits the specified job to some underlying resource management or allocation system.

Concerning the *Job* object, the relevant functions are:

- **getExitStatus**() - Returns the exit status of a job.
- **getInfo**() - This method returns information about the associated job.
- **getJobID**() - Returns the job id, a globally unique identifier for the physical job corresponding to this instance.
- **getState**() - This method returns the state of the Job
- **stop**() - Stops the associated physical job.

5.2.4 Monitoring

The most important objects of this functionality are *org.gridlab.gat.monitoring.Metric*, *org.gridlab.gat.monitoring.MetricEvent*, *org.gridlab.gat.monitoring.MetricListener* and *org.gridlab.gat.monitoring.Monitorable*. A *Metric* object represents a measurable quantity within a monitoring system and is used to specify a measurable quantity. A *MetricEvent* object represents the measured value of a quantity measured by a monitoring system and is used to specify to interested parties that a measurement of a quantity corresponding to a *Metric* has taken place. The *MetricListener* object is implemented by classes which wish to be informed of *MetricEvents* and is used to inform instances of such classes of *MetricEvent*'s. The *Monitorable* object is implemented by classes which wish to be monitored for *MetricEvents* and is used to inform interested parties of such events.

For the *Metric* object the most interesting functions are:

- **Metric**(MetricDefinition definition, Map<String,Object> metricParameters) - Constructor: Constructs a Metric instance from the passed Metric name and concrete values for the Metric parameters.
- **Metric**(MetricDefinition definition, Map<String,Object> metricParameters, long frequency) - Constructor: Constructs a Metric instance from the passed Metric name and concrete values for the Metric parameters.
- **getDefinition**() - Gets the MetricDefinition.
- **getFrequency**() - Gets the measurement frequency in milliseconds.
- **getMetricParameterByName**(String name) - Gets the Metric parameter value associated with the passed Metric parameter name.
- **getMetricParameters**() - Gets the Metric parameters associated with this Metric.

For the *MetricEvent* object, the available functions are:

- **getEventTime**() - This method returns the number of milliseconds after January 1, 1970, 00:00:00 GMT when the event happened.
- **getMetric**() - This method returns an instance of the Metric to which this MetricEvent corresponds.
- **getValue**() - This method returns the value corresponding to this MetricEvent.

The 2 other objects (*MetricListener* and *Monitorable*) must be instantiated when needed.

5.2.5 Access to Information Service

The most important object of this functionality is the *org.gridlab.gat.advert.AdvertService* one. The

AdvertService allows Advertisable (An interface which is realized by any class which wishes to get advertised in the advert service.) instances to get published to and queried in an advert directory. Such an advert directory is a meta data directory with an hierarchical namespace attached.

Important functions of the *AdvertService* object are:

- **add**(Advertisable advert, MetaData metaData, String path) - Add an Advertisable instance and related meta data to the AdvertService, at path.
- **delete**(String path) - Remove an Advertisable instance and related meta data from the AdvertService, at path.
- **find**(MetaData metaData) - Query the AdvertService for entries matching the specified set of meta data in the MetaData.
- **getAdvertisable**(String path) - Gets an Advertisable instance from the given path.
- **getMetaData**(String path) - Gets the MetaData of an Advertisable instance from the given path.

5.2.6 Conclusion

The complete javaGAT documentation in a javadoc format can be found on this web page: http://www.cs.vu.nl/ibis/javadoc/javagat/index.html. As we said previously, the gLite connector is really new and a lot of functionality is currently missing. For the neuGRID project, the most important areas that need to be covered by the connector are "File operations" and "Job submissions". The other area could be useful at some point but not mandatory as the neuGRID infrastructure will mainly insert/move files and execute jobs into the gLite grid middleware.

In the following section, we will describe what is currently available using the gLite connector.

5.3 The gLite javaGAT connector

At time of writing (January 2009), in the last version of javaGAT that can be found in there source code repository, the following areas are covered by the gLite adaptor:

- File operations,
- Job submissions

As it was said previously, those are the two areas that are needed for the neuGRID project. Unfortunately, everything is not yet available in each of these areas. In the next section, a state of the completeness of these areas will be done.

5.3.1 Completeness of the File Operations API

The gLite connector currently contains two objects for the file operations:

- GliteGuidFileAdaptor: This object is able to handle the files which have Uniform Resource Identifier (URI) like "guid://".
- GliteSrmFileAdaptor: This object is able to handle the files which have URI like "srm://".

Basically, with these two adaptors, we are able to interact with the gLite Logical File Catalogue (LFC), the gLite information system (tBDII) and the gLite Storage Element (SRM). In the next table, you can see the currently implemented functionality for each adaptor:

20avaGat API: *File*	Adaptors	Implementation
canRead()	GliteGuidFileAdaptor	NO
	GliteSrmFileAdaptor	NO
canWrite()	GliteGuidFileAdaptor	NO
	GliteSrmFileAdaptor	NO
copy(URI loc)	GliteGuidFileAdaptor	DONE
	GliteSrmFileAdaptor	DONE
createNewFile()	GliteGuidFileAdaptor	DONE
	GliteSrmFileAdaptor	NO
exists()	GliteGuidFileAdaptor	NO
	GliteSrmFileAdaptor	NO
delete()	GliteGuidFileAdaptor	DONE
	GliteSrmFileAdaptor	DONE
isDirectory()	GliteGuidFileAdaptor	NO
	GliteSrmFileAdaptor	NO
isFile()	GliteGuidFileAdaptor	NO
	GliteSrmFileAdaptor	NO
lastModified()	GliteGuidFileAdaptor	NO
	GliteSrmFileAdaptor	NO
length()	GliteGuidFileAdaptor	NO
	GliteSrmFileAdaptor	NO
listFiles()	GliteGuidFileAdaptor	NO
	GliteSrmFileAdaptor	NO
mkdir()	GliteGuidFileAdaptor	NO
	GliteSrmFileAdaptor	NO
mkdirs()	GliteGuidFileAdaptor	NO
	GliteSrmFileAdaptor	NO
move(URI location)	GliteGuidFileAdaptor	NO
	GliteSrmFileAdaptor	NO
recursivelyDeleteDirectory()	GliteGuidFileAdaptor	NO
	GliteSrmFileAdaptor	NO
renameTo(File dest)	GliteGuidFileAdaptor	NO
	GliteSrmFileAdaptor	NO

As you can see, a lot of functionality is missing. Never the less, most of the missing functionality is trivial to do and will be done soon.

Moreover, currently, there is no object which is able to deal with "lfn://" URI. This kind of adaptor will be needed if developers need to organize the grid files inside the LFC. If we discover that this functionality becomes needed, it will be developed (the work is already in progress).

5.3.2 Completeness of the Job Submission API

For this area, the gLite connector provides one implementation of the *ResourceBroker* API and one for the *Job* one:

- GliteResourceBrokerAdaptor
- GliteJob

As it was done in the previous section, the following table shows the completeness of the

GliteResourceBrokerAdaptor object:

javaGAT API: *ResourceBroker*	Implementation:
findResources(ResourceDescription resourceDescription)	DONE
reserveResource(ResourceDescription resourceDescription, TimePeriod timePeriod)	DONE
reserveResource(Resource resource, TimePeriod timePeriod)	DONE
submitJob(JobDescription description)	NO
submitJob(JobDescription[] descriptions)	NO
submitJob(JobDescription[] descriptions, MetricListener listener, String metricDefinitionName)	NO
submitJob(JobDescription description, MetricListener listener, String metricDefinitionName)	DONE

As you can see, this functionality is now well exposed. The only restriction is that you cannot submit multiple jobs on the same time. This restriction is not really important as you can always do that in the application source code really easily.

Bellow, the same table for the *Job* object:

javaGAT API: *Job*	Implementation: *GliteJob*
getExitStatus()	NO
getInfo()	DONE
getJobID()	DONE
getState()	DONE
stop()	DONE

Here it is the same; most of the functionality is implemented.

5.3.3 Conclusion

As you can see, the gLite javaGAT connector is already well advanced for the job submission area and a lot of stuff remains for the data management area. The WP7 team already started to complete it for the neuGRID project needs and will continue to do it until that the neuGRID requirements will be complete.

6 Conclusion

The work presented in this deliverable contributes to the project in the following two areas:

- Building of neuGRID Grid-based test-bed: the PoC infrastructure.
- Exposing the gLite functionalities within the neuGRID Platform

To actually build and deploy the Grid-based infrastructural solution in WP7, the neuGRID infrastructure architects should have been clearly aware of the project users' requirements. As the document "Users Requirements Specifications" [3] was not yet released, internal meetings were done with the WP9 team. Based on the result of these meetings and on the experience that already have some partners involve in this work package, a list of gLite middleware components to install was created. Based on that, a standard and flexible architecture was put in place.

Moreover, different manners of exposing the gLite functionality to the whole platform were studied. It was decided to expose it through the very well know javaGAT API which aim to abstract users from the different Grid APIs that can exists and to expose everything under a set of coordinated, generic and flexible APIs. JavaGAT has the advantage of been open source, which mean that it can be easily enhanced for the neuGRID project needs if necessary. Also, in longer term, this would allow the neuGRID community to interact with other grid infrastructures.

The conclusions above demonstrate that WP7 team has successfully followed the WP7 program of work. As it was promised in the description of work, a set of functionalities were investigated and deployed.

Future plans are mainly focused on two tasks. First, maintenance of the POC infrastructure has to be done. This include to update it with the very latest gLite components versions and to provide feedback to EGEE if needed. Second, the JavaGAT gLite connector will have to be completed to at least cover the neuGRID needs.

7 Glossary

API	Application Program Interface
BDII	Berkeley Database Information Index
BES	Basic Execution Services
BLAH	Batch Local ASCII Helper Protocol
CA	Certification Authority
CASTOR	CERN Advanced STORage Manager
CE	Computing Element
CEMon	Computing Element MONitor
CLI	Command Line Interface
CREAM	Computing Resource Execution And Management
DGAS	Distributed Grid Accounting System
DLI	Data Location Interface
DN	Distinguished Name
DPM	Disk Pool Manager
EGEE	Enabling Grids for E-sciencE
FTS	File Transfer Service
GFAL	Grid File Access Library
GG	Grid Gate
GIIS	Grid Index Information Server
gLite	EGEE Grid middleware stack
GLUE	Grid Laboratory for a Uniform Environment
GMA	Grid Monitoring Architecture
GRIS	Grid Resource Information Server
ICE	Interface to CREAM Environment
IS	Information System (grid-level)
LSF	Local Sharing Facility
LB	Logging and Bookkeeping
LFN	Logical File Name
MDS	Monitoring and Discovery Service
neuGRID platform	neuGRID services + gLite middleware
PBS	Portable Batch System
PKI	Public Key Infrastructure
POC	neuGRID Proof Of Concept sub-infrastructure – *neuGRID test-bed*
PROD	neuGRID Production sub-infrastructure
R-GMA	Relational Grid Monitoring Architecture

RFIO	Remote File Input/Output
RB	Resource Broker
SAM	Service Availability Monitoring framework
SD	Service Discovery
SE	Storage Element
SL	Scientific Linux
SL4	Scientific Linux 4
SRM	Storage Resource Manager
SURL	Storage URL
SWIG	Simplified Wrapper and Interface Generator
TURL	Transport URL
UI	User Interface
VDT	Virtual Data Toolkit
VO	Virtual Organization
VOMS	Virtual Organization Membership Service
WN	Worker Node
WMS	Workload Management System

8 Bibliography

[1] gLite Grid middleware website - http://www.glite.org/

[2] EGEE website - http://www.eu-egee.org/

[3] neuGRID WP9 deliverable D9.1 - "Users Requirements Specifications (URS) document first release" – due month 14 of the project

[4] neuGRID WP9 deliverable D9.2 - "Users Requirements Specifications (URS) document final release" – due month 26 of the project

[5] EGEE-II-MJRA1.2 - "Functional description of grid components" - https://edms.cern.ch/document/736259/2

[6] MyProxy - http://grid.ncsa.uiuc.edu/myproxy/

[7] VDT, Virtual Data Toolkit - http://vdt.cs.wisc.edu/

[8] VeriSign - http://www.verisign.com

[9] The gLite-UI - http://glite.web.cern.ch/glite/packages/R3.1/deployment/glite-UI/glite-UI.asp

[10] java.lang.Runtime online javadoc: http://java.sun.com/j2se/1.5.0/docs/api/java/lang/Runtime.html

[11] JavaGAT - http://www.cs.vu.nl/ibis/javagat.html

[12] I. Foster, C. Kesselman and S. Tuecke, The Anatomy of the Grid, International Journal of High performance Computing Applications, 15, 3, 2001