

# Programming the Grid with gLite

Laure, E.  
(CERN, Geneva, Switzerland ) *et al*

22 March 2006



EGEE is a project funded by the European Commission  
Contract number INFSO-RI-508833

The electronic version of this EGEE Technical Reports is available  
on the CERN Document Server at the following URL:  
<<http://cdsweb.cern.ch/search.py?p=EGEE-TR-2006-001>>

# Programming the Grid with gLite\*

E. Laure<sup>1</sup>, S.M. Fisher<sup>2</sup>, A. Frohner<sup>1</sup>, C. Grandi<sup>3</sup>, P. Kunszt<sup>4</sup>, A. Krenek<sup>5</sup>,  
O. Mulmo<sup>6</sup>, F. Pacini<sup>7</sup>, F. Prelz<sup>2</sup>, J. White<sup>8</sup>,  
M. Barroso<sup>1</sup>, P. Buncic<sup>1</sup>, F. Hemmer<sup>1</sup>, A. Di Meglio<sup>1</sup>, A. Edlund<sup>6</sup>

<sup>1</sup>CERN, Geneva, Switzerland    <sup>2</sup>RAL, Didcot, UK    <sup>3</sup>INFN, Italy  
<sup>4</sup>CSCS, Manno, Switzerland    <sup>5</sup>CESNET, Prague, Czech Republic  
<sup>6</sup>KTH, Stockholm, Sweden    <sup>7</sup>DATAMAT, Rome, Italy    <sup>8</sup>HIP, Helsinki

March 10, 2006

## Abstract

The past few years have seen the creation of the first production level Grid infrastructures that offer their users a dependable service at an unprecedented scale. Depending on the flavor of middleware services these infrastructures deploy (for instance Condor, gLite, Globus, UNICORE, to name only a few) different interfaces to program the Grid infrastructures are provided. Despite ongoing efforts to standardize Grid service interfaces, there are still significant differences in how applications can interface to a Grid infrastructure. In this paper we describe the middleware (gLite) and services deployed on the EGEE Grid infrastructure and explain how applications can interface to them.

## 1 Introduction

Advances in networking and distributed computing allowed the establishment of production Grid infrastructures during the past few years. Today, large-scale production Grid infrastructures such as EGEE in Europe, OSG in the US, and NAREGI in Japan are offering their services to many scientific and industrial applications, from domains as diverse as Astronomy, Biomedicine, Computational Chemistry, Earth Sciences, Financial Simulations, and High Energy Physics.

Grid infrastructures provide these applications a new means for collaborative research by facilitating the sharing of computational and data resources at an unprecedented scale. The efficient and secure sharing of data resources, which can reach several Tera- to Petabytes in some application domains, is one of the main challenges for Grid infrastructures.

The Enabling Grids for E-sciencE project (EGEE) is Europe's flagship Research Infrastructures Grid project and the world's largest Grid infrastructure of its kind. It involves more than 70 partners from 27 countries, arranged in twelve regional federations, and providing more than 20,000 CPUs, almost 200 sites and 10 petabytes of available network storage. This infrastructure supports 7 scientific domains and more than 20 individual applications.

Started in April 2004, EGEE has rapidly grown from a European to a global endeavor, and along the way learned a great deal about the business of building production-quality infrastructure. The consortium

---

\*This work is co-funded by Enabling Grids for E-sciencE (EGEE), a project of the European Commission (contract number INFSO-RI-508833)

behind this effort represents a significant proportion of Europe’s Grid experts, including not only academic institutions but also partners from the Research Network community and European industry.

Grid systems and applications aim to integrate, virtualise, and manage resources and services within distributed, heterogeneous, dynamic *Virtual Organisations* across traditional administrative and organisational domains (*real organisations*) [31].

A Virtual Organisation (VO) comprises a set of individuals and/or institutions having access to computers, software, data, and other resources for collaborative problem-solving or other purposes. Virtual Organisations are a concept that supplies a context for operation of the Grid that can be used to associate users, their requests, and a set of resources. The sharing of resources in a VO is necessarily highly controlled, with resource providers and consumers defining clearly and carefully just what is shared, who is allowed to share, and the conditions under which sharing occurs [30].

This resource sharing is facilitated and controlled by a set of services that allow resources to be discovered, accessed, allocated, monitored and accounted for, regardless of their physical location. Since these services provide a layer between physical resources and applications, they are often referred to as *Grid Middleware*<sup>1</sup>.

The Grid system needs to integrate Grid services and resources even when provided by different vendors and/or operated by different organisations. The key to achieve this goal is standardisation. This is currently being pursued in the framework of the Global Grid Forum (GGF) and other standards bodies.

EGEE deploys the *gLite* middleware [18], a middleware stack that combines components developed in various related projects, in particular Condor [7], Globus [12], LCG [19], and VDT [29], extended by EGEE developed services. This middleware provides the user with high level services for scheduling and running computational jobs, accessing and moving data, and obtaining information on the Grid infrastructure as well as Grid applications, all embedded into a consistent security framework.

In this paper we describe how the EGEE infrastructure can be programmed with the most recent version of gLite, gLite-3.0. After discussing the overall gLite architecture and the gLite security framework in Section 2 and Section 3, respectively, we highlight particular gLite services used for Information and Monitoring (Section 4), Workload Management (Section 5), and Data Management in Section 6. Section 7 reports on experiences our user communities gained with using the EGEE infrastructure, and we end the paper with some concluding remarks and an outlook on future work in Section 8.

## 2 The gLite Architecture

The gLite Grid services follow a Service Oriented Architecture [25] which will facilitate interoperability among Grid services and allow easier compliance with upcoming standards, such as OGSA, that are also based on these principles. The services are expected to work together in a concerted way in order to achieve the goals of the end-user, however, they can also be deployed and used independently, allowing their exploitation in different contexts.

Figure 1 depicts the high level services, which can thematically be grouped into 5 service groups:

**Security services** encompass the Authentication, Authorization, and Auditing services which enable the identification of entities (users, systems, and services), allow or deny access to services and resources, and provide information for post-mortem analysis of security related events. It also provides functionality for data confidentiality and a dynamic connectivity service, i.e. a means for a site to control network access patterns of applications and Grid services utilizing its resources.

**Information and Monitoring Services** 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.

---

<sup>1</sup>See for instance [12] for a discussion of the different software layers in a Grid infrastructure.

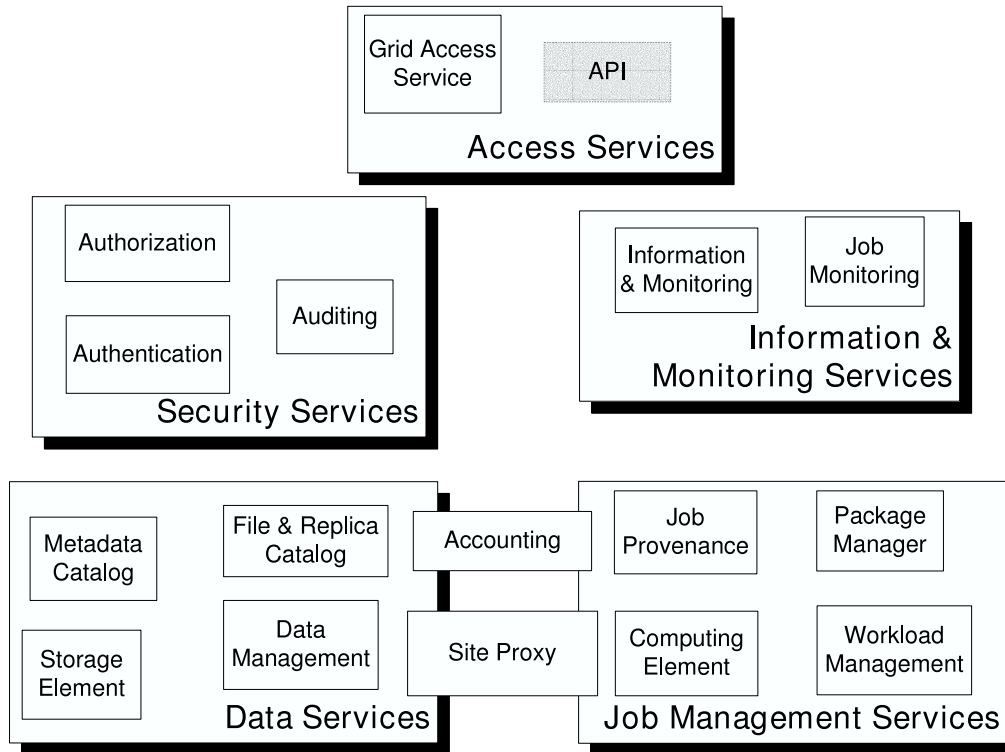


Figure 1: gLite Services

**Job Management Services** The main services related to job management/execution are the computing element, the workload management, accounting, job provenance, and package manager services. 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.

The Computing Element (CE) provides the virtualization of a computing resource (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.

The Workload Management System (WMS) is a Grid level metascheduler 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 via the logging and bookkeeping service.

The Job Provenance (JP) service provides persistent information on jobs executed on the Grid infrastructure for later inspections, data-mining operations, and possible re-runs.

Finally, the Package Manager (PM) service allows the dynamic deployment of application software.

While the CE and WMS are part of the production gLite 3.0 release, the JP and PM are only available as prototypes and will not be further discussed in this paper.

**Data Services** The three main services that relate to data and file access are: Storage Element, File & Replica Catalog Services and Data Management.

In all of the data management services described below the granularity of the data is on the file level. However, the services are generic enough to be extended to other levels of granularity.

The Storage Element (SE) provides the virtualization of a storage resource (which can reach from simple disk servers to complex hierarchical tape storage systems) much as the CE does for computational resources.

The catalog services keep track of the data location as well as relevant metadata (e.g. checksums and filesizes) and the data movement services allow for efficient managed data transfers between SEs. The access to files is controlled by Access Control Lists (ACLs). Application specific metadata is expected not to be stored in the basic gLite services but in application specific metadata catalogs.

All the data management services act on single files or collections of files. To the user of the EGEE data services the abstraction that is being presented is that of a global file system. A client user application may look like a Unix shell which can seamlessly navigate this virtual file system, listing files, changing directories, etc.

Note, that the gLite architecture does not in general impose specific deployment scenarios (i.e. how many instances of a certain service are available to a user, if a service is replicated or distributed, etc.). Most importantly, service instances may serve multiple VOs which will facilitate the scalability and performance of the Grid system although a VO may require its own instance as well.

In the remainder of this paper we focus in particular on the security, monitoring, job management, and data management service, as these are the services a typical user mostly interacts with. Details on the internals of the gLite services are beyond the scope of this paper and can be found in the gLite architecture document [26].

### 3 Security

The EGEE security architecture [15] is based on well established work in the Grid community.

On the Authentication side a credential storage ensures proper security of (user-held) credentials while proxy certificates enable single sign-on. TLS, GSI, and WS-Security transport or message-level security protocols ensure integrity, authenticity and (optionally) confidentiality. The EU GridPMA establishes a common set of trust anchors for the authentication infrastructure.

Attribute authorities enable VO managed access control, while policy assertion services enable the consolidation and central administration of common policy. An authorization framework enables local collection, arbitration, customization and reasoning on policies from different administrative domains, as well as integration with service containers and legacy services.

The functionalities described in EGEE security architecture are in most cases *embedded* in the service container or in the application itself, for performance reasons – they are not rendered as separate Web Services.

It is important that the security architecture used by EGEE allows for basic interoperability with other Grid deployments or middleware projects.

Figure 2 depicts an overview on how the components in the security architecture interact in the following typical request flow:

1. The user <sup>2</sup> obtains Grid credentials from a credential store, and the necessary tokens that assert the user's rights to access the resource <sup>3</sup>. The credentials are short-lived and often derived from longer-term credentials, such as X.509 identity certificates issued by a Certification Authority (CA).

EGEE uses myProxy [16] as credential store and the Virtual Organization Membership Service VOMS [22] as attribute authority. VOMS is also used to manage the membership of VOs.

2. The user and the service container authenticate identities to each other and establish a secure communication channel across the (open) network with integrity, authenticity and confidentiality protection, and over which a SOAP message payload is conveyed. By default, this is accomplished by use of HTTP over TLS. The established connection event is logged.

---

<sup>2</sup>We use the word “user” in wide terms: for instance, it also encompasses the software agents that act on the user's behalf

<sup>3</sup>A “resource” in Web Services terminology is practically anything that is managed by a service: it can be a compute element, a file transfer client, an information index etc.



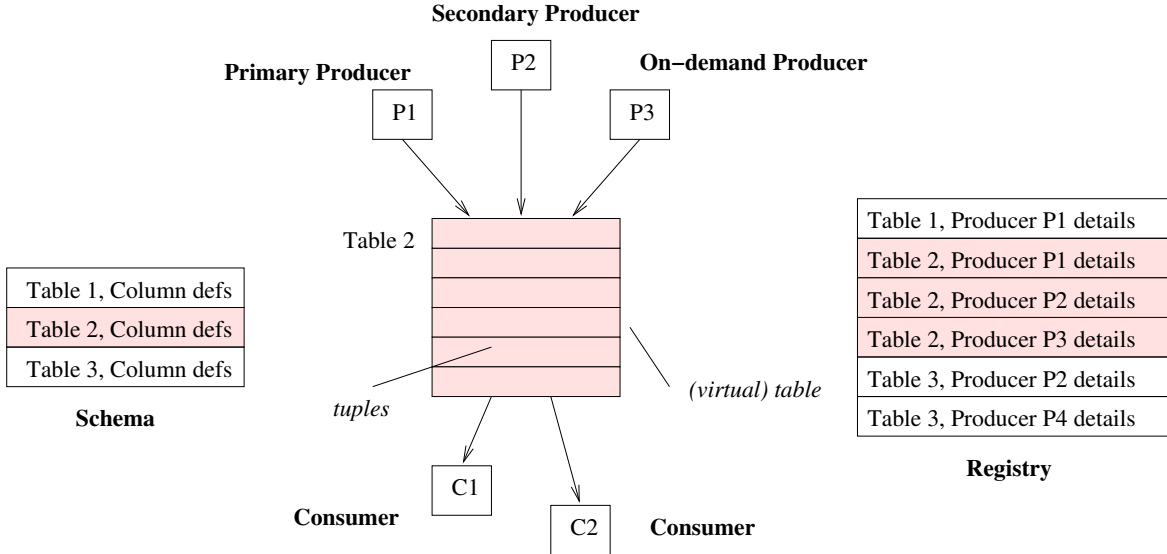


Figure 3: R-GMA Components

boundaries, avoiding malicious or unintended usage or in the worst scenario a security breach. These include

- a. Operating the resource in a different user space than that of the service container.
- b. Consulting the Dynamic Connectivity Service in order to temporarily enable direct inbound and/or outbound network connectivity to the resource.
- c. Providing additional protection of the delegated credentials by use of an Active Credential Store. This is also useful in the case of long-term use of a resource, where a renewal of the delegated credentials may be necessary.

## 4 Information and Monitoring Services

The gLite system for information and monitoring is R-GMA[23, 14], which is a Relational implementation of the Grid Monitoring Architecture[27] from the GGF[9]. R-GMA has been designed to be easy for end users to publish information (from a batch job or otherwise) and query that information in a grid environment.

Figure 3 shows the principal components of R-GMA. Data is written into the R-GMA virtual database by *producers* and read from it by *consumers*.

R-GMA is not a distributed database management system. Instead, it provides a useful and predictable information system built on a much looser coupling of data providers across a grid.

**Defining the schema** The first task for the user is to define what needs to be published. This has to be one or more tables following the relational model. A common technique in design of a relational schema is to make use of “surrogate keys”: a small integer which can be used as a foreign key to establish a relationship. A traditional case would be to assign an *departmentId* to each department and then to include this as a column of the employee table. This works well for a single managed database with a mechanism to assign *departmentId* values, but it does not work in the grid. You should not assume anything about what anybody else is publishing. It is best to think of publishing a series of measurements of the same quantity but made at different times; all R-GMA tuples (records) have an associated timestamp and the R-GMA query types take advantage of this.

**Producers** Producers are the data providers for the virtual database. Writing data into the virtual database is known as *publishing*, and data is always published in complete rows, known as *tuples*. There are three classes of producer: *Primary*, *Secondary* and *On-demand*. Each is created by a user application and returns tuples in response to queries from consumers. The main difference is in where the tuples come from.

There are three ways considered here to for a job to publish data into R-GMA. The least intrusive is to use a job wrapper which can publish information on the state of the job picked up by looking at stdout. This can be done without any modifications to the job itself, provided that useful information can be gleaned from stdout. The job wrapper will insert data into the R-GMA system by means of a primary producer which will have four important R-GMA calls:

1. Create primary producer with appropriate properties
2. Declare table with predicate - this information goes into the registry
3. Insert tuples into virtual database
4. Close primary producer

A second alternative is to insert R-GMA calls directly into the application code. This might be done using any of the supported APIs: C, C++, Java and Python. The code, from an R-GMA viewpoint, is identical to that used above in the job wrapper.

A third approach is to use the native logging API (e.g. `log4cxx` or `log4j`) to log useful things. You will then need to run the program with an R-GMA appender which we have provided for Java and C++. This takes the messages which might otherwise have gone to the terminal or to syslog and sends them to an R-GMA producer. This is an attractive solution in that it requires that the user can just use his existing logging mechanisms but has the disadvantage that it is not possible to modify the schema.

You may wish to collect information together into a secondary producer which is capable of answering latest or history queries. If so you should probably set up two of them for some redundancy. For the sake of this example we will assume that you wish to store history so you create a secondary producer to answer history queries.

**Consumers** In R-GMA, each consumer represents a single SQL SELECT query on the virtual database. The query is first matched against the list of available producers in the registry and a set of producers capable of answering the query is selected.

There are four query types: *continuous*, *latest*, *history* and *static*. They are all expressed by a normal SQL query though there are some restrictions on the continuous query as this simply acts as a filter on published tuples and so joins and aggregate functions are not permitted. If you issue a continuous query you will receive every tuple satisfying the query as it is published. Such a query has no natural end. The latest query only considers those tuples which were most recently published. Tables have a primary key defined to allow latest tuples to be defined.

You can then query the information - if you perform a continuous query you will be connected to the primary producers but if you carry out a history query you will be connected to the secondary producer which was created to answer history queries.

**Command Line Tool** An easy to use command line tool (written in Python) is also provided with a built-in help system. This tool accepts short commands and provides defaults for as much as possible. For example:

```
rgma> SELECT Name, Endpoint FROM Service
```

where `rgma>` is the prompt, will issue a query using the current values of parameters such as the type of query, the timeout etc. The current values can be changed or displayed:

```
rgma> SET QUERY CONTINUOUS
rgma> SET TIMEOUT 3 minutes
rgma> SHOW MAXAGE
```

Command history and command completion are also provided.

**Service Discovery** The approach taken to service discovery was an API hiding the underlying information system. The information system is linked in via a plug-in mechanism for which we currently support R-GMA, bdII and an XML file. APIs are provided in C and Java and allow a user (or another service) to select a suitable service.

To understand more of how to use R-GMA for monitoring and of how to use the Service Discovery APIs please consult the R-GMA documentation [14].

## 5 Workload Management Services

The Workload Management System (WMS) comprises a set of Grid middleware components responsible for the distribution and management of tasks across Grid resources, in such a way that applications are efficiently executed.

The specific kind of tasks that request computation are usually referred to as “jobs”. In the WMS, the scope of tasks needs to be broadened to take into account other kinds of resources, such as storage or network capacity. This change of definition is mainly due to the move from batch-like activity to applications with more demanding requirements for data access or interactivity, both with the user and with other tasks.

The core component of the Workload Management System is the Workload Manager (WM), whose purpose is to accept and satisfy requests for job management coming from its clients. The other fundamental component is the Job Logging and Bookkeeping Service, which is described below.

For a computational job there are two main types of request: submission and cancellation. The status request is managed by the Logging and Bookkeeping Service.

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 CE for execution, taking into account the requirements and the preferences expressed in the job description. The decision of which resources should be used is the outcome of a *matchmaking* process between submission requests and available resources. The availability of resources for a particular task depends not only on their state, but also on the utilization policies that the resource administrators and/or the administrator of the VO the user belongs to have put in place.

### 5.1 The gLite Job Description Language

A job passed to the gLite WMS needs to be described in a specific language, the gLite Job Description Language (JDL).

The JDL used for gLite, and originally developed for the EU DataGrid project, is based on the Condor ClassAd language [24]. Its central construct is a record-like structure, the *classad*, composed of a finite number of distinct attribute names mapped to expressions. An expression contains literals and attribute references composed with operators in a C/C++ like syntax.

These *ads* conform to a protocol that states that every description should include expressions named Requirements and Rank, which denote the requirements and preferences of the advertising entity. Two entity descriptions match if each *ad* has an attribute, Requirements, that evaluates to **true** in the context of the other *ad*.

The main advantages of this framework can be summarized by the following three points:

- it uses a semi-structured data model, so no specific schema is required for the resources description, allowing it to work naturally in a heterogeneous environment

- the language folds the query language into the data model. Requirements (i.e. queries) may be expressed as attributes of the job description
- ClassAds are first-class objects in the model, hence descriptions can be arbitrarily nested, leading to a natural language for expressing resources and jobs aggregates (e.g. DAGs) or co-allocation requests

The gLite JDL defines specific attributes to specify:

1. batch or interactive, simple, MPI-based, checkpointable and partitionable jobs;
2. aggregates of jobs with dependencies (Directed Acyclic Graphs);
3. constraints to be satisfied by the selected computing and storage resources;
4. data access requirements: appropriate conventions have been established to express constraints about the data that a job wants to process together with their physical/logical location within the grid;
5. preferences for choosing among suitable resources (ranking expressions)

As mentioned, the JDL is semi-structured and extensible. A set of predefined attributes have a special meaning for the underlying components of the Workload Management System. Some of them are mandatory, while others are optional. The set of predefined attributes [13] can be decomposed in the following groups:

- *Job attributes*: representing job specific information and specifying actions that have to be performed by the WMS to schedule the job
- *Data attributes*: representing the job input data and Storage Element related information. They are used for selecting the resources from which the application has the best access to data
- *Requirements and Rank*: allowing the user to specify respectively which are the needs and preferences, in term of resources, of their applications.

The Requirements and Rank expressions are built using the *Resources Attributes*, which represent the characteristics and status of the resources and are recognizable in the job description as they are prefixed with the string “other”. The *Resources attributes* are not part of the predefined set of attributes for the JDL as their naming and meaning depends on the adopted Information Service schema [10] for publishing such information. This independence of the JDL from the resources information schema allows targeting for the submission resources that are described by different Information Services without any changes in the job description language itself. Here is an example of the JDL used to describe a simple job:

```
[
  Type = "Job";
  JobType = "Normal";
  VirtualOrganisation = "biomed";
  Executable = "/bin/bash";
  StdOutput = "std.out";
  StdError = "std.err";
  Arguments = "./sim010.sh";
  Environment = "GATE_BIN=/usr/local/bin";
  OutputSandbox = {"std.out","std.err","Brain_radioth000.root"};
  InputData = {"lfn:BrainTotal", "lfn:EyeTotal"};
  DataAccessProtocol = {"file", "gridftp"};
  OutputSE = "grid011.pd.infn.it";
  InputSandbox = {
    "/home/fpacini/JOBS/bin/sim010.sh",
    "/home/fpacini/JOBS/jobsRAL/required/prerunGate.mac",
```

```

    "/home/fpacini/JOBS/jobsRAL/required/GateMaterials.db"
};
rank = -other.GlueCEStateEstimatedResponseTime;
requirements = Member("GATE-1.0-3",other.GlueHostApplicationSoftwareRunTimeEnvironment)
                && (other.GlueCEStateFreeCPUs >= 2);
]

```

The job description above represents a Monte Carlo simulation of radiological imaging. It asks to run the *sim010.sh* simulation script on a resource on which the GATE (Geant4 Application for Tomographic Emission) is installed and which has at least 2 CPUs available for the computation. Image data to be accessed for the simulation are identified in the grid with the logical names *BrainTotal* and *EyeTotal*. For further details on the meaning of the JDL attributes the reader can refer to [13].

## 5.2 The WMS User Interfaces

After having created the descriptions of their applications, users expect to be able to ignore the complexity of the grid resources and to be enabled to submit them to the Workload Management System and monitor their evolution over the Grid.

The functionalities the WMS provides include the following:

- Job (including DAGs) submission for execution on a remote Computing Element, also including:
  - automatic resource discovery and selection,
  - staging of the application input sandbox,
  - restart of the job from a previously saved checkpoint state,
  - interactive communication with the running job,
- Listing of resources suitable to run a specific job according to job requirements,
- Cancellation of one or more submitted jobs,
- Retrieval of the output files of one or more completed jobs,
- Retrieval of the checkpoint state of a completed job,
- Retrieval of jobs bookkeeping and logging information.

All this functionality is made available through a command line interface and an API providing C++ and Java bindings. GUI components have been developed on top of the Java API.

### 5.2.1 Command Line Interface

Here's a short reference of the basic commands of the gLite command line interface. More details can be found at [8]. Information about the usage of each command can be found by issuing <sup>5</sup> :

```
<command> -help
```

**glite-wms-job-submit** submits a job to a WMS (more precisely, WMPProxy) Service. It requires a JDL file as input and returns a WMS job identifier.

**glite-wms-job-status** queries the Logging & Bookkeeping service (the information collection and retrieval partner of the WMS) about the status of a given job.

**glite-wms-job-logging-info** lists the events collected in the Logging & Bookkeeping service that were collected over the lifetime of a given job, and that allow to determine its current status.

---

<sup>5</sup>Before using any command you should make sure that the `GLITE_WMS_LOCATION` and `GLITE_LOCATION` environment variables point to a valid WMS-UI installation path (i.e. the path containing the `etc` and `bin` directories).

**glite-wms-delegate-proxy** allows the user to delegate her proxy credential to the WMPProxy service. This delegated credential can then be used for job submissions.

**glite-wms-job-list-match** lists the identifiers of jobs submitted to a WMPProxy Service by the user issuing the command.

**glite-wms-job-cancel** cancels one or more jobs previously submitted to WMPProxy Service.

**glite-wms-job-output** retrieves output files of a job, when finished. After this operation the job context is purged and no more operations are possible on it

**glite-wms-job-perusal** manages the perusal (access to files in the working area of a running job) functionality for a given job.

### 5.2.2 Application Programming Interface

The WMS client API supplies the client applications with a set of interfaces over the job submission and control services made available by the gLite WMS through a web service based interface. The API provides the corresponding method for each operation published in the WSDL description of the WMPProxy Service (<http://egee-jra1-wm.mi.infn.it/egee-jra1-wm/wmproxy>).

The request types supported by the WMPProxy Service are:

- Job: a simple application
- DAG: a directed acyclic graph of dependent jobs
- Collection: a set of independent jobs

Jobs in turn can be batch, interactive, MPI-based, checkpointable, partitionable and parametric. The specification of the JDL for describing the request types is available at [13]. Besides requests submission, the WMPProxy also exposes additional functionality for request management and control such as cancellation, job files perusal and output retrieval. Requests status follow-up can be instead achieved through the functionality exposed by the Logging & Bookkeeping (LB) service [17].

The documentation describing the WMPProxy Client API providing C++, Java and Python bindings can be found at [32]. Pointers to usage examples are also provided in these web pages.

## 5.3 Logging and Bookkeeping

The Logging and Bookkeeping service (L&B) [17, 3] is used by WMS internally to gather various information on running jobs and provide the user with an overall view on the job state. The service collects events in a non-blocking asynchronous way with a robust delivery mechanism. The job state is computed on the fly at the bookkeeping database, using a state machine that tolerates even out of order event delivery. Besides gathering the “system” information on running jobs, the service can also collect user information in the form of arbitrary “name = value” tags (annotations) assigned to a job, both from a running application or independently.

The job status information gathered by the LB is made available through the gLite user-interface commands. In addition to this simple querying mechanism, the user can pose simple or more complex queries with the public L&B API (available in C and C++ or as a web-service interface). Examples of such queries are:

- state of a concrete job,
- details on all user’s running jobs,
- jobs that are running on a concrete computing element,
- user’s jobs that returned exit code between e.g. 2 and 7,
- user’s jobs resubmitted in last two hours,

- user's jobs, annotated as green or red color, that started execution in the first week of January,
- user's failed jobs that were marked as red first, and then re-colored to green,
- red-colored jobs, heading to a computing element at which the user's job have recently failed.

The list of more or less random examples presented here demonstrates the strength of the L&B API. The user can also register for receiving notifications when a job enters a state matching conditions specified in a similar way. Job state information is also fed into the R-GMA infrastructure to provide yet another way of accessing the job bookkeeping information. More detailed examples of use of the LB service are discussed in detail in [17], including appropriate code fragments.

## 6 Data Management Services

For the gLite data management service stack we make the assumption that the lowest granularity of the data is on the file level. We deal with files rather than data objects or tables in a relational database if it comes to application data. The reason for this arguably very restrictive assumption is twofold. Primarily the initial two application groups, the High Energy Physics and Biomedical communities, that work with the EGEE gLite implementation store their data in file format. The second reason is that the semantics of files are very well understood by everyone, both on the service provider and application side. This is not the case for generic data objects for example, where every application group has their own definition.

In the Grid the user identifies files by logical file names (LFNs). The LFN namespace is hierarchical, just like a conventional filesystem. The semantics of the LFN namespace is also almost exactly like that of a Unix filesystem. The LFN is not the only name/identifier that is associated with a file in the Grid, although the average user may never use any other filename and is given the benefit of a single global namespace. To maintain this view, the Grid data management middleware has to keep track of logical to physical file instance mappings in a scalable manner (see Section on Catalogs below).

We have the following names identifying data in the Grid:

**LFN** Logical File Name: A logical, human readable, identifier for a file. LFNs are unique but mutable, i.e. they can be changed by the user (the files can be renamed). The namespace of the LFNs is a global hierarchical namespace, which is how file-based data is organized on any computerized system today. The same tools and semantics may be provided to the user on the logical namespace of the Grid as on any local filesystem. Each Virtual Organisation can have its own namespace.

**GUID** Global Unique Identifier: A logical identifier, which guarantees its uniqueness by construction (based on the UUID mechanism [20]). Each LFN also has a GUID (1:1 relationship). GUIDs are immutable, i.e. they cannot be changed by the user. Once a file acquires a GUID it must not be changed otherwise consistency cannot be assured. GUIDs are being used by Grid applications as immutable pointers between files. If these should change, the application may suddenly point to a wrong file. In the filesystem analogy, GUIDs would be the unique inode number of the file. The 1:1 relation means that we do not allow hard links in this virtual filesystem – experience tells that implementing a globally distributed filesystem with hard links is very difficult and introduces unnecessary complexities, especially for the *delete* operation.

**Logical Symlinks** The logical namespace also provides the concept of symbolic links. Symbolic links always point to an LFN. There may be many Symlinks to an LFN (N:1 relation). If an LFN is removed or renamed, the Symlinks are left dangling, in analogy with the usual filesystem semantics.

**SURL** The Site URL specifies a physical instance (replica) of a file. In other projects the SURL is also referred to as the Physical File Name (PFN). A file may have many replicas, so the mapping between GUIDs and SURLs is a one-to-many mapping. Each file replica has its own unique SURL. In gLite, SURLs are always fully qualified SRM names, accepted by the Storage Element's SRM interface (see the storage Section below). An example SURL is

```
srm://castorgrid.cern.ch:8443/srm/managerv1?SFN=/castor/cern.ch/file1
```

The SRM endpoint is implicitly given by the part of the SURL that comes before ?SFN. Usually, users are not directly exposed to SURLs, but only to the logical namespace defined by LFNs. (The Storage URL StURL is another term used by the SRM specification, for the actual file name inside the storage system. To the Storage, the Site URL is a logical name and the StURL is the real location of the file on disk.)

**TURL** Transport URL. It is a URL that can be used to actually transfer a file using any standard transport protocol. The TURL is a fully qualified URL starting with the protocol to be used for transfer or direct file access through some native I/O mechanism.

The data services can be put into three basic categories: storage, catalogs and movement, which we describe below.

**Storage** gLite relies on storage systems exposing an SRM [11] interface. Current systems supported include Castor (<http://cern.ch/castor>), dCache (<http://www.dcache.org/>) and the gLite Disk Pool Manager (DPM).

The DPM has been developed as a lightweight solution for disk storage management offering much of the functionality of dCache but avoiding its complexity. DPM is security enabled, providing ACL based authentication to file access. In addition to the SRM interface, DPM offers an rfiio interface for posix like data access and gridFTP [6] for data transfer. This is also the mechanism the gLite file transfer service described below, is using.

In order to shield the user from the differences the current storage systems expose in their posix-like access libraries, gLite provides a Grid File Access Library (GFAL), a C API posix-like interface that provides methods such as `gfal.open`, `gfal.read`, etc. GFAL interfaces with the different SRM implementations (including their native posix access mechanisms) and gridFTP.

**Catalogs** gLite provides a catalog, named LFC, to store the location(s) of their files and replicas. LFC will map LFNs or GUIDs to SURLs. It is a high performance file catalogue that builds on the experiences gathered from the EGEE user communities. The LFC supports Oracle and Mysql as database backends, and is integrated with the GFAL interface. It shares the codebase with the name service part of the DPM, discussed above.

Similarly to the DPM, the LFC exposes methods to the user through the GFAL interface that, in turn, interacts with the SRM implementations and gridFTP. The LFC client has a POSIX-like command line interface with commands such as `lfc-chmod`, `lfc-ls`, `lfc-rm`.

**Data Movement** The gLite File Transfer Service FTS is a low level data movement service, responsible for moving sets of files from one site to another while allowing participating sites to control the network resource usage. This control includes the enforcement of site and usages policies such as fair-share mechanisms on dedicated network links. It is designed for point to point movement of physical files. The FTS has dedicated interfaces for managing the network resource and to display statistics of ongoing transfers. The FTS is also able to communicate with external Grid File Catalogs, i.e. the file to be transferred can also be specified using an LFN.

The FTS has three interfaces that can be used for programming. The **File Transfer Interface** is used to submit File Transfer jobs, get status on current jobs, list requests in a given job state, cancel transfers, set priority of transfers; and to add, remove and list VO managers. The **Channel Management Interface** can be used to add, list and delete channels for the FTS instance, and set channel parameters. It has also methods to add, remove and list channel managers and to apply policies for jobs that need manual intervention, such as being in HOLD state. Finally, the **Status Interface** can be used to list or summarize the channel and VO activity, and to list all running background Transfer Agent processes.

There is a set of command line tools available that interact with these interfaces, performing these tasks by contacting the FTS. All the FTS interfaces come with WSDL descriptions and the user can actually use the WSDL to generate clients for any language needed. The gLite distribution includes a set of client APIs for Java, C/C++ and Perl. As a secure connection is used to talk to the FTS web service, a valid GSI proxy is necessary. The VOMS extensions are needed if the client wants to contact for example the Channel Management interface. This should only be used by VO and site managers, who should have an extra “admin” group membership signed by VOMS.

The FTS Transfer Interface’s `transferSubmit` method takes as input a `TransferJob` object, which consists of

- an array of `TransferJobElements` each describing an individual file transfer within the job (source and destination pairs).
- a list parameters (key, value pairs) for transfer layer specific parameters that are applied to each file transfer (e.g. gridFTP parameters)
- the credential that is used by the transfer system to retrieve the appropriate proxy for the transfer.

The rest of the FTS Transfer Interface, the Channel Management Interface and Status Interface methods are simple and straightforward setters and getters very much in java style, that can easily be used like any other RPC call through SOAP. The detailed syntax for the API and all command line tools is described in the user guide [4].

## 7 Application Usage

The gLite middleware has been exposed to users both on the EGEE production infrastructure and in the EGEE pre-production service, a smaller scale service for testing upcoming middleware releases.

The production infrastructure is now being used by many applications on a regular basis. On average, over 10,000 concurrent jobs are being served every day. It is worth noting that the EGEE applications involve quite different workflows and hence most applications customize the infrastructure to their particular needs. This is for instance done by using additional services to those provided by gLite, for instance workflow systems or Grid portals.

To show these differences in application usage, we highlight the experiences from the two EGEE pilot applications, Biomed and High Energy Physics (HEP) below. Full details on these as well as the other EGEE applications can be found in [1].

One focus of the HEP activity was on massive data transfer tests using the gLite FTS. In January 2006 a rate up to 1 GB/s sustained over several days was achieved in conditions similar to those that the CERN LHC experiments will have at the beginning of data taking in 2007. The test involved 12 main computing centers and 20 other computing facilities all over the world.

The HEP communities are using the EGEE production infrastructure also for their daily activities, such as Monte Carlo generation of simulated physics events. About 4 million jobs were executed on the production infrastructure in the past year. Job submission is not limited to specialized users producing data for the whole community. It is now possible for end users, not only Grid experts, to use the Grid infrastructure for their daily data analyses.

The Biomed use of the infrastructure is different from the HEP one. Both the amount of data and number of computing cycles required are less than for HEP, but the complexity of the calculations and the diversity of the needs are higher. This is particularly challenging from the point of view of middleware design. A large fraction of the Biomed Grid activity has been performed on the PPS as they require the most recent features of the workload and data management systems.

The main needs of the biomed community are:

- Fast responses are needed for interactive usage

- The submission overhead is not negligible, given that their job duration is much smaller;
- Data security: privacy issues arise when dealing with medical data management. A system for accessing encrypted data stored on DICOM [2] servers through an SRM [11] interface was demonstrated in October 2005 during the 4<sup>th</sup> EGEE conference in Pisa.

It is this diversity of requirements that make the provision of middleware for a multi-purpose Grid infrastructure like EGEE a challenging task. The clear need for customization has led to the service oriented architecture approach followed by gLite that allows the gLite services to be used independently in many different settings.

## 8 Conclusions and Future Work

In this paper we gave a brief overview on how the EGEE Grid infrastructure can be programmed using the services provided by the gLite middleware distribution. We focused on the information & monitoring, workload management, and data management services which are the most frequently used ones. A discussion of the security framework was also given.

As mentioned before, EGEE is only one of the large scale Grid infrastructures that were created in the past years. Each of the different infrastructures is deploying different services built on different software stacks. Hence, interoperability between Grid infrastructures is becoming an important issue, in particular for applications such as the HEP ones that need to exploit multiple infrastructures. Although the Global Grid Forum, GGF, is active in defining Grid related standards, in particular OGSA [31], it is not practical for a production level infrastructure to follow quickly evolving standards. These systems need to take a more conservative approach and wait for established standards to arrive.

As a consequence, gLite was not following proposed standards such as OGSI [28] that was quickly superseded by WSRF [5] but is carefully moving towards web services adhering to WS-Interoperability [21] wherever possible. In addition, interoperability efforts are ongoing with major other Grid projects such as OSG, NAREGI and others which is resulting in a seamless integration of the systems. It is now for instance possible to run EGEE jobs on OSG and vice versa. The experiences gained are fed back to GGF and other relevant standardization bodies.

The gLite middleware is constantly being improved and enriched with further services. The most important improvements planned comprise:

R-GMA will introduce support of multiple virtual databases (VDBs), each defined by its own registry and schema to act as separate namespaces. Each VDB will have its own authorization rules and a query will be able to span VDBs. Other work will be on the resilience and performance of the services. For service discovery we plan to introduce a bootstrapping mechanism.

The WMS components will move towards Web Services based interfaces, in particular for the CE, compatible with standards for job and workflow descriptions such as JSDL, and also see performance optimizations. The appearance of established standards will also help in consolidating the user interfaces and reducing the number of translation layers implemented in user space.

The LB service will become an independent component with a Web Service interfaces, providing a general “job” tracking and event collecting tool. Tight integration with Job Provenance will be also established.

The overall security model of the data management components is constantly being improved and will allow for consistent access control not only at the storage (SE) level but also on the catalogs. A Generic metadata service, AMGA, will also be provided. This service allows users to attach metadata to files stored in the file catalogue and to handle simple relational data stored in a relational database system. Using AMGA, users will be able to select logical files by searching through the metadata describing the files’ content.

More information on gLite as well as the latest version of the software can be obtained from gLite web page <http://www.glite.org>.

## Acknowledgments

gLite is a collaborative effort of the EGEE-JRA1 (<http://cern.ch/egee-jra1>) and EGEE-JRA3 (<http://cern.ch/egee-jra3/>) teams. The authors would like to thank all the members of the teams for their commitment to gLite and their contributions to the overall gLite effort. The authors particularly thank Miron Livny from the Condor team and Kate Keahey from the Globus team for many stimulating discussions and their contributions to the overall design of gLite.

## References

- [1] V. Breton et al. EGEE Deliverable 4.4: Second Revision of EGEE Application Migration Progress Report. <https://edms.cern.ch/document/707799/>.
- [2] DICOM Digital Imaging and Communication in Medicine. <http://dicom.nema.org/>.
- [3] F. Dvořák et al. Services for Tracking and Archival of Grid Job Information. In *Proceedings Cracow Grid Workshop'05*, 2006. <http://www.cyfronet.krakow.pl/cgw05>.
- [4] EGEE JRA1. EGEE File Transfer Service User Guide. <https://edms.cern.ch/document/591792/>.
- [5] Karl Cajkowski et. al. The WS-Resource Framework, 2004. <http://www-106.ibm.com/developerworks/library/ws-resource/ws-wsrf.pdf>.
- [6] W. Allcock et al. GridFTP Protocol Specification. Global Grid Forum Recommendation GFD.20, March 2003.
- [7] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke. Condor-G: A Computation Management Agent for Multi-Institutional Grids. *Cluster Computing*, 5(3):237–246, 2002.
- [8] glite command line interface reference. <https://edms.cern.ch/document/674643/1>.
- [9] Global Grid Forum. <http://www.gridforum.org/>.
- [10] Homepage of the GLUE Schema Activity. <http://infnforge.cnaf.infn.it/glueinfomodel>.
- [11] The GGF Grid Storage Resource Manager Working Group.
- [12] I. Foster and C. Kesselman and S. Tuecke. The Anatomy of the Grid. *The International Journal of High Performance Computing Applications*, 15(3):200–222, Fall 2001.
- [13] JDL Attributes Specification. <https://edms.cern.ch/document/590869/1>. EGEE-JRA1-TEC-590869-JDL-Attributes-v0-4.
- [14] JRA1-UK. <http://hepunix.rl.ac.uk/egee/jra1-uk/>.
- [15] EGEE JRA3. EGEE Deliverable 3.3: Global Security Architecture (1st revision). <https://edms.cern.ch/document/602183/>.
- [16] D. Kouril and J. Basney. A Credential Renewal Service for Long-Running Jobs. In *Grid 2005 - 6th IEEE/ACM International Workshop on Grid Computing*, Seattle, US, November 20005.
- [17] A. Křenek et al. L&B Users Guide. <https://edms.cern.ch/file/571273/1/LB-guide.pdf>.
- [18] E. Laure, F. Hemmer, et al. Middleware for the Next Generation Grid Infrastructure. In *Computing in High Energy and Nuclear Physics (CHEP)*, Interlaken, Switzerland, September 2004.
- [19] The LCG Project. <http://cern.ch/lcg>.

- [20] Paul J. Leach and Rich Salz. UUIDs and GUIDs, February 1998.
- [21] The Web Services Interoperability Organization. WS-I Documents. <http://www.ws-i.org/Documents.aspx>.
- [22] R. Alfieri and others. VOMS, an Authorization System for Virtual Organizations. In *Grid Computing, First European Across Grids Conference*, 2004.
- [23] R-GMA. <http://www.r-gma.org/>.
- [24] R. Raman, M. Livny, and M. Solomon. Matchmaking: Distributed Resource Management for High Throughput Computing. In *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing (HPDC7)*, Chicago, IL, July 1998.
- [25] D. Sprott and L. Wilkes. Understanding Service-Oriented Architecture. <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnmaj/html/aj1soa.asp>.
- [26] EGEE Middleware Design Team. EGEE Deliverable 1.4: EGEE Middleware Architecture. <https://edms.cern.ch/document/594698/>.
- [27] B. Tierney, R. Aydt, et al. A grid monitoring architecture. Technical Report GWD-Perf-16-1, GGF, 2001.
- [28] S. Tuecke et al. Open Grid Services Infrastructure (OGSI) - Version 1.0. <https://forge.gridforum.org/projects/ogsi-wg>.
- [29] The Virtual Data Toolkit. <http://www.cs.wisc.edu/vdt/>.
- [30] GGF OGSA WG. Open Grid Services Architecture – Glossary of Terms. <https://forge.gridforum.org/projects/ogsa-wg>.
- [31] GGF OGSA WG. The Open Grid Services Architecture, Version 1.0. <https://forge.gridforum.org/projects/ogsa-wg>.
- [32] WMproxy API documentation. <http://egee-jra1-wm.mi.infn.it/egee-jra1-wm/glite-wmproxy-api-index.shtml>.