MIDDLEWARE FOR THE NEXT GENERATION GRID INFRASTRUCTURE*

E. Laure, F. Hemmer, A. Aimar, M. Barroso, P. Buncic,
A. Di Meglio, L. Guy, P. Kunszt, CERN, Switzerland
S. Beco, F. Pacini, Datamat SpA, Italy
F. Prelz, M. Sgaravatto, INFN, Italy
A. Edlund, O. Mulmo, KTH, Sweden
D. Groep, NIKHEF, The Netherlands
S.M. Fisher, RAL, UK
M. Livny, University of Wisconsin, USA

Abstract

The aim of the EGEE (Enabling Grids for E-Science in Europe) project is to create a reliable and dependable European Grid infrastructure for e-Science. The objective of the EGEE Middleware Re-engineering and Integration Research Activity is to provide robust middleware components, deployable on several platforms and operating systems, corresponding to the core Grid services for resource access, data management, information collection, authentication & authorization, resource matchmaking and brokering, and monitoring and accounting.

For achieving this objective, we developed an architecture and design of the next generation Grid middleware leveraging experiences and existing components essentially from AliEn, EDG, and VDT. The architecture follows the service breakdown developed by the LCG ARDA group. Our strategy is to do as little original development as possible but rather re-engineer and harden existing Grid services. The evolution of these middleware components towards a Service Oriented Architecture (SOA) adopting existing standards (and following emerging ones) as much as possible is another major goal of our activity.

INTRODUCTION

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*) [10].

A Virtual Organisation (VO) comprises a set of individuals and/or institutions having direct access to computers, software, data, and other resources for collaborative problem-solving or other purposes. The VO concept is used to set up a context for Grid operations, associating users with their requests and allocated resources. The sharing of resources in a VO 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*.

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.

In this document we present the architecture of the EGEE Grid Middleware (called `gLite`). It is influenced by the requirements of Grid applications, the ongoing work in GGF on the Open Grid Services Architecture (OGSA) [10], as well as previous experience from other Grid projects such as the EU DataGrid (EDG) (http://www.edg.org), the LHC Computing Grid (LCG) (http://cern.ch/lcg), AliEn (http://alien.cern.ch), NorduGrid (http://www.nordugrid.org), and the Virtual Data Toolkit VDT (http://www.cs.wisc.edu/vdt/) which includes among others Condor (http://www.cs.wisc.edu/condor) and Globus (http://www.globus.org).

THE GLITE ARCHITECTURE

The `gLite` Grid services follow a *Service Oriented Architecture* 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 architecture constituted by this set of services is not bound to specific implementations of the services. In order to address the end-user requirements, the services need to work together in a coordinated manner, although individual services can still be deployed and used independently. This allows their exploitation in different contexts. We intend to implement the gLite services using web service technologies. Therefore, all service interfaces are defined using the web service definition language (WSDL).

The gLite service decomposition has been largely influenced by the work performed in the LCG project (the requirements and technical assessment group on an "architectural roadmap for distributed analysis" (ARDA) [2]). Figure 1 depicts the high level services, which can themat-

*This work was funded by the European Commission program INFSO-RI-508833 through the EGEE project.

ically be grouped into 5 service groups (plus *accounting* and *site proxy*).[1]

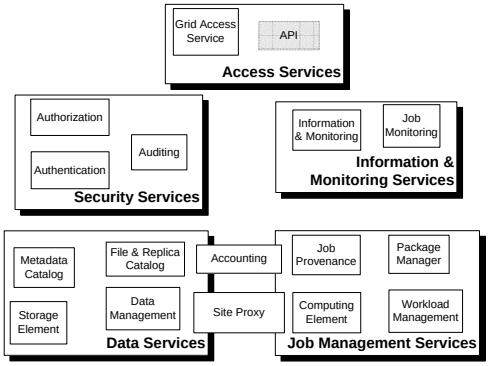

Figure 1: gLite Services

In the following, we provide an overview of the main services and refer the interested reader to the gLite architecture [6] and design [7] documents for further details.

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. They also provide functionality for data confidentiality and a Site Proxy, i.e. a means for a site to control network access patterns of applications and Grid services utilising its resources.

API and Grid Access Service: provide a common framework by which the user may gain access to the Grid services. The access service will manage the life-cycle of the Grid services available to a user, according to his/her privileges. The access service also provides a convenient backend for Grid portals, like for instance the Genius portal [1]. Despite the existence of the access service, all gLite services may be accessed directly via an API (which will in most cases be generated from the service WSDL) or command line tools using this API. The API is of course not a service on it's own, however it is shown in Figure 1 to make this way of accessing the services explicit.

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 specialised services, such as the Job Monitoring Service, will exploit them. The underlying information service will be able to cope with streams of data and the merging and republishing of those streams. The system relies upon registering the location of publishers of information and what subset of the total information they are publishing. This allows consumers to issue queries to the information system while not having to know where the information was published.

All published information carries with it the time and date when it was first published (i.e. when the "measurement" was made) as well as the identity of the publisher and from where it was published. This information is not modifiable even if data are republished. In fact no data can be modified in the system thus avoiding any inconsistencies when data are republished. There are of course mechanisms to clean out old data (under the control of the publisher of that data).

Finally there is a fine-grained, rule-based authorization scheme to ensure that people can only read or write within their authority.

Job Management Services: The main services related to job management/execution are the computing element (CE), the workload management (WMS), 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. Hence, accounting is depicted in Figure 1 straddling the data and job management services.

These services communicate with each other as the job request progresses through the system, so that a consistent view of the status of the job is maintained.

When this communication needs to occur during execution on the computing nodes (notably in the cases of communication to the job provenance service, to the package manager service, and to any network service that provides "interactive" services such as the bridging and buffering of standard streams, or signaling to running jobs), the usual technique of wrapping the executable content into "wrapper" scripts will be used.

Multiple workflows for job management will be offered, in particular the *push* and *pull* models: In the push model, the WMS decides where to execute a job based on the information on available CEs and their characteristics as obtained from the information system. Additional input like the location of input data may be considered as well. In the pull model, the WMS buffers jobs in a *task queue* and the CEs actively request jobs from the WMS. These requests are matched against the available jobs again taking into account information like the location of input data.

Figure 2 illustrates the main components of the WMS system and its interaction with other gLite services, notably the CE.

Data Services: The three main service groups that relate to data and file access are: Storage Element, Catalog Services and Data Management. Closely related to the data services are the security-related services and the Package Manager.

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

[1] Other categorisations (e.g. a layered architecture as discussed in [5]) are possible as well.

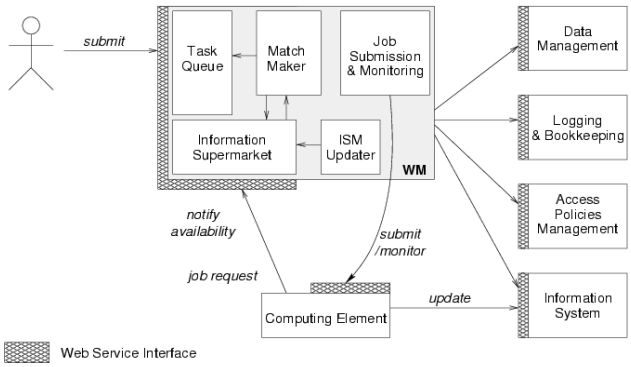

Figure 2: Internal architecture of the Workload Manager

ity. Most application data is expected to be located in files (as opposed to relational database systems for example). This assumption is not made for application metadata. In a distributed environment, there will be many replicas (managed copies) of the user's files stored at different physical locations. To the user this may be transparent, the middleware will provide the capabilities for replica management.

In the Grid the user identifies files by logical file names (LFNs). The LFN is the key by which the users locate the actual locations of their files. We refer to file *replicas* (as opposed to file copies) if the instances of a given file are being tracked by the Replica Catalog (RC). To the user of the 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 (as in AliEn) which can seamlessly navigate this virtual file system, listing files, changing directories, etc.

The replicas are identified by Site URLs (SURLs). Each replica has its own SURL, specifying implicitly which Storage Element needs to be contacted to extract the data. The SURL is a valid URL that can be used as an argument in an SRM interface [3]. Usually, users are not directly exposed to SURLs, but only to the logical namespace defined by LFNs. The Grid Catalogs provide mappings needed for the services to actually locate the files. To the user the illusion of a single file system is given.

The data in the files can be accessed through the Storage Element (SE). The access to the files is controlled by Access Control Lists (ACL). The detailed semantics of file access will be different depending upon what kind of storage back-end is being used beneath an SE; there may be substantial latencies for reads and a large number of possible failure modes for write.

The Data Management System will expose all nontrivial interfaces to the user for data placement in a distributed environment where data transfers will be scheduled much like jobs. The services comprise the Data Management System are the Data Scheduler (DS), the Transfer Fetcher, the File Placement Service (FPS) and the File Transfer Library (FTL).

The Data Scheduler is a top-level service, keeping track of data movement requests in a VO that are being submitted directly by the user through a portal or user interface or by computational jobs submitted to the WMS. The Transfer Fetcher polls the Data Scheduler and fetches transfers whose destination is the local site for the given VO, inserting new requests into the File Placement Service. The File Placement Service coordinates the transfer performed by the File Transfer Service and makes sure that the File and Replica catalogs are updated properly.

Figure 3 provides a schematic overview on the various data management services and their interplay.

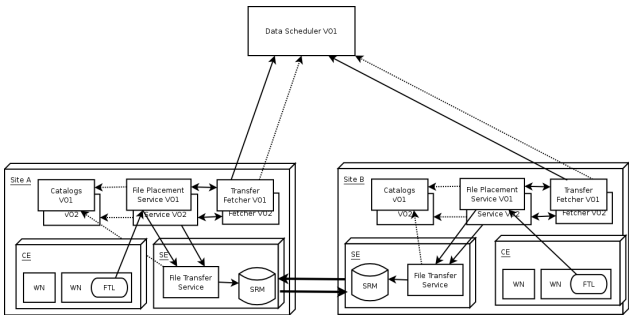

Figure 3: Architecture overview of the Data Management Services. The dotted arrows represent queries, the solid arrows represent requests.

IMPLEMENTATION CONSIDERATIONS

The Grid system realised by the services described above should allow a maximum of flexibility in service deployment, service composition, and service interoperability. In order to achieve this, implementations of the services need to take into account the requirements discussed in Section 3 of the Architecture document [6]. The main issues include:

Interoperability: Service implementations need to be interoperable in such a way that a client may talk to different independent implementations of the same service. Following a strict SOA approach with well-defined interfaces specified in WSDL will help achieve this goal. In addition, the Grid services need to be able to co-exist with, and leverage existing Grid infrastructures like LCG (`http://cern.ch/lcg`), Grid2003 (`http://www.ivdgl.org/grid2003/`), or NorduGrid (`http://www.nordugird.org`). This can be achieved in developing lightweight services that only require minimal support from their deployment environment.

Service Deployment: Grid services need to be easily deployable and configurable across a wide range of platforms. This goes along the lines of the goal of interoperability with existing infrastructures discussed above. In addition, several deployment scenarios need to be supported (e.g. services running together on the same physical machine, services supporting single or multiple VOs).

Service Autonomy: Although the services constituting the gLite architecture are supposed to work together in a concerted way, they should be usable also in a stand-alone manner in order to be exploitable in different contexts. Ideally, if a user only requires a subset of services to achieve his task, he should not be forced to use additional services.

Re-use of existing services: As much as possible, existing, well-tested components should be adopted according to the presented design, rather than implementing everything from scratch. We envisage in particular the re-use of components originating from projects like AliEn, Condor, EDG, Globus, LCG, VDT, NorduGrid and others.

Client libraries: [7] defines the gLite services by means of WSDL which will allow the creation of client libraries on demand. However, this procedure exposes all the low level details of the involved protocols to the user and thus proper client libraries hiding these details should be provided where appropriate.

FROM PROTOTYPE TO RELEASE

For implementing the services described above we have adopted a fast prototyping approach, exploiting existing services originating from AliEn, EDG, LCG, and VDT as much as possible. This prototype, which currently is installed at two main sites (CERN and Wisconsin) with some service components being installed also at Bologna and RAL, offers the early adopters of the gLite services the opportunity to assess the service breakdown, semantics, and interfaces and, through their feedback, guide the further development of gLite. Since May 2004 selected users from the HEP and Biomedical communities are using the prototype on a regular basis. In particular the members of the LCG ARDA project are assessing the prototype for distributed analysis and provide invaluable feedback [8].

In order to move from this fast prototyping approach to a robust and deployable release, rigorous integration and testing processes need to be applied. In particular, all of the gLite software is maintained in a common CVS repository and automatically built every night. These nightly builds are tagged and forwarded to the internal testing team for (largely automated) tests on at least three geographically distributed sites. See [9] and [4] for more details on the integration and testing activities.

Once a service passes all the testing steps, it is forwarded to the EGEE operation activity for deployment on a *pre-production service* comprising about 20 sites where it will become available to a larger user community. The first gLite services are expected to be delivered to the pre-production service in autumn 2004.

We believe that the continuous interaction with our user communities from the early prototyping stages up to the pre-production services combined with rigorous integration and testing activities will from the basis of the success of the gLite middleware.

CONCLUSION AND FUTURE WORK

In this paper we presented a high level overview on the services constituting the next generation Grid middleware as developed within the EGEE project. We outlined the architecture of the main Grid services and discussed the main issues and approaches in implementing these services.

The first release of the gLite services is expected in spring 2005, although individual services will be available to a large user community via the pre-production services much earlier. We expect that this first release will encompass the basic services needed to run successfully applications on the Grid. In the remaining lifetime of EGEE (up to spring 2006) these services will be further stabilized and augmented with additional, higher level services that will allow a more efficient usage of the available Grid resources. Example of these future services include advanced reservation and the consideration of the network resources in job and data scheduling. In addition, significant work is needed to unify and ease the installation and configuration of the gLite services.

REFERENCES

[1] A. Andronico, R. Barbera, et al. GENIUS: a simple and easy way to access computational and data grids. *Future Generation of Computer Systems*, 2003.

[2] P. Buncic, F. Rademakers, R. Jones, et al. Architectural Roadmap towards Distributed Analysis. Technical report, LHC Computing Grid Project, October 2003. `http://lcg.web.cern.ch/lcg/PEB/arda/public_docs/ARDA_report_final.pdf`.

[3] The GGF Grid Storage Resource Manager Working Group. SRM Documents. `http://sdm.lbl.gov/gsm/documents.html`.

[4] L. Guy et al. Distributed Testing Infrastructure and Processes for the EGEE Grid Middleware. In *Proceedings CHEP04*, September 2004.

[5] 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.

[6] EGEE JRA1. EGEE Middleware Architecture. EU Deliverable DJRA1.1, July 2004. `https://edms.cern.ch/document/476451/`.

[7] EGEE JRA1. EGEE Middleware Design. EU Deliverable DJRA1.2, September 2004. `https://edms.cern.ch/document/487871/`.

[8] B. Koblitz et al. First Experiences with the EGEE Middleware. In *Proceedings CHEP04*, September 2004.

[9] A. Di Meglio et al. A Pattern-based Continuous Integration Framework for Distributed EGEE Grid Middleware Development. In *Proceedings CHEP04*, September 2004.

[10] GGF OGSA WG. The Open Grid Services Architecture, Version 1.0. `https://forge.gridforum.org/projects/ogsa-wg`.