EGEE

EGEE Middleware Architecture

AND PLANNING (RELEASE 2)

EU Deliverable DJRA1.4

Document identifier:	**EGEE-DJRA1.1-594698-v1.0**
Date:	**July 15, 2005**
Activity:	**JRA1: Middleware Engineering and Integration**
Lead Partner:	**CERN**
Document status:	**FINAL**
Document link:	**https://edms.cern.ch/document/594698/**

Abstract: This document describes the Service-Oriented Architecture of the EGEE project's *gLite* middleware. It is a revision of DJRA1.1 (https://edms.cern.ch/document/476451/1.0 taking into account the experiences and feedback received on DJRA1.1 and the subsequent releases of the middleware. The JRA1 work plan (https://edms.cern.ch/document/573493/), which presents the detailed plans of how these services will be implemented in the course of the second year of the EGEE project, augments this document. Note, that what is described in this document goes beyond what realistically can be released by the end of the project and as such also presents a road-map for further developments.

Delivery Slip

	Name	Partner	Date	Signature
From	JRA1 Design Team	CERN, CCLRC/RAL, INFN DATAMAT, CESNET, NESC, NIKHEF, KTH/PDC, Univ. of Chicago Univ. of Wisconsin-Madison	31/05/2005	
Reviewed by	PTF Massimo Lamanna Marc-Elian Begin Zdenek Sekera Sara Collins	 CERN CERN CERN UEDIN	27/06/2005	
Approved by	PEB		14/07/2005	

Document Change Log

Issue	Date	Comment	Author
0.2	07/07/2005	Implemented changes requested from reviewers	JRA1 Design Team
1.0	15/07/2005	Update document status for delivery to EU	Erwin Laure

Document Change Record

Issue	Item	Reason for Change

CONTENTS

1 INTRODUCTION

1.1 PURPOSE OF THE DOCUMENT

This document describes the middleware architecture of the EGEE middleware, called *gLite*, by means of a service oriented architecture. It is a revision of DJRA1.1 (https://edms.cern.ch/document/476451/1.0 taking into account the experiences and feedback received on DJRA1.1 and the subsequent releases of the middleware. It is a living document and we expect modifications of the proposed architecture as we gain experience with practical implementations, feedback from our users, and evolution of the requirements.

The detailed specification of these services will be described in a separate design document (Deliverable DJRA1.5) which will be a revision of the original document DJRA1.2 (https://edms.cern.ch/document/487871/).

In the remainder of this paper we discuss some key requirements for our Grid architecture in Section 3 followed by a brief introduction to the principles of service oriented architectures in Section 4. Sections 5-10 present the gLite Grid services in detail.

Open issues, in particular the relationship of our architecture to emerging standards, are covered in Section 11. The document concludes with a brief summary after having discussed a few implementation considerations in Section 12.

1.2 APPLICATION AREA

This document applies to the implementation of the gLite middleware within the scope of the EGEE project and the JRA1 and JRA3 activity mandate. It is also applicable to other activities within EGEE, in particular SA1, JRA4, NA3, and NA4.

1.3 MAIN CHANGES TO DJRA1.1

The main changes to the previous version of this document, DJRA1.1. (https://edms.cern.ch/file/476451/1.0) can be summarized as follows:

1. Section 5 – Security Services has been augmented with a discussion of the *Dynamic Connectivity Service* which replaces the old Section 11.1 – Site Proxy.
2. The discussion on API and Grid Access Service in Section 6 has been changed to a generic description of the APIs and CLIs offered by the gLite Services. The inclusion of a Grid Access Service is not planned in the current version of this architecture.
3. Section 7 – Information and Monitoring Services has been augmented with a discussion of *Service Discovery*.
4. Section 9 – Data Services has been significantly revised taking into account the feedback on the previous architecture and the experiences gained in developing the first release of these services.
5. A new Section on Agreement Service (Section 10.2) has been added.
6. A new Section on Configuration and Instrumentation (Section 10.3) has been added.
7. The original Section 10 – Use Cases has been removed from the document.

8. Section 11 – Issues now contains an updated discussion on related standards, in particular for service coordination, addressing, WSRF, WS-I, notification, and job submission. This combines and updates previous discussions in the original Section 11 – Issues. A discussion on virtualization has been added as well.

9. The discussion of the Network Element (originally in Section 11 – Issues) has been elaborated and split into discussions on Network Performance Monitoring in Section 7.4 and Bandwidth Allocation and Reservation in Section 10.1.

10. Section 12 – Implementation Considerations has been augmented with a discussion on typical deployment scenarios.

1.4 Document Amendment Procedure

This document can be amended by the EGEE JRA1 design team. The document shall be maintained using the tools provided by the CERN EDMS system.

1.5 Terminology

AA	Attribute Authority
AC	Attribute Certificate
ACL	Access Control List
AFS	Andrew File System
API	Application Programming Interface
AuthN	Authentication
AuthZ	Authorization
BT	Bulk Transfer
CA	Certification Authority
CAS	Community Authorization Service
CE	Computing Element
CEA	Computing Element Acceptance
CLI	Command Line Interface
DGAS	DataGrid Accounting System
DNS	Domain Name System
DRMAA	Distributed Resource Management Architecture API
EDG	European DataGrid
EGEE	Enabling Grids for E-Science in Europe
FC	File Catalog
FPS	File Placement Service
FS	File System
FTP	File Transfer Protocol
FTS	File Transfer Service
GAS	Grid Access Service
GGF	Global Grid Forum
GN2	The GEANT2 Project
GOC	Grid Operations Centre
GRAM	Grid Resource Access Manager
GSI	Grid Security Infrastructure
GUID	Global Unique Identifier
GSM	Grid Storage Management
GUI	Graphical User Interface

HEP	High Energy Physics
HLM	Higher Level Middleware
HTTP	Hypertext Transfer Protocol
ISM	Information Super Market
I/O	Input / Output
JC	Job Controller
JDL	Job Description Language
JP	Job Provenance Service
L&B	Logging and Bookkeeping Service
LCG	LHC Computing Grid
LFN	Logical File Name
LHC	Large Hadron Collider
L-NSAP	Local Network Service Access Point
LRMS	Local Resource Management System
NAT	Network Address Translation
NE	Network Element
NOC	Network Operations Centre
NPM	Network Performance Monitoring
NM-WG	GGF's Network Monitoring Working Group
NREN	National Research and Education Network
NSAP	Network Service Access Point
NTFS	(Microsoft) NT File System
OCSP	Online Certificate Status Protocol
OGSA	Open Grid Services Architecture
PAT	Policy Administration Tool
PCI	Policy Communication Interface
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PKI	Public Key Infrastructure
PM	Package Management
POSIX	Portable Operating System Interface (X)
PR	Policy Repository
QoS	Quality of Service
RADIUS	Remote Authentication Dial In User Service
RC	Replica Catalog
RDMS	Relational DataBase Management System
RLS	Replica Location Service
SAML	Security Assertion Markup Language
SE	Storage Element
SIPS	Site Integrated Proxy Service
SLA	Service Level Agreement
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SRM	Storage Resource Manager
SURL	Site URL
TQ	Task Queue
UC	User Context

URL	Uniform Resource Locator
UUID	Universal Unique Identifier
VDT	Virtual Data Toolkit
VLL	Virtual Leased Line
VO	Virtual Organisation
VOMS	Virtual Organisation Membership Service
WM	Workload Manager
WMS	Workload Management System
WS	Web Services
WS-A	Web Services Addressing
WS-E	Web Services Eventing
WS-I	Web Services Interoperability
WS-N	Web Services Notification
WSRF	Web Services Resource Framework
XACML	eXtensible Access Control Markup Language

2 EXECUTIVE SUMMARY

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*) [81].

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. 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 [80].

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*.

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 revised architecture of the EGEE Grid Middleware (called `gLite`). It is influenced by the requirements of Grid applications (cf. Section 3), the ongoing work in the Global Grid Forum (GGF) on the Open Grid Services Architecture (OGSA) [81], the feedback and experiences with the first version of this document and the subsequent releases of the gLite middleware, 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`), the Virtual Data Toolkit VDT (`http://www.cs.wisc.edu/vdt/`), including among others Globus (`http://www.globus.org`) and Condor (`http://www.cs.wisc.edu/condor`), and NorduGrid (`http://www.nordugrid.org`).

The `gLite` Grid services (cf. Sections 5-9) follow a *Service Oriented Architecture* (cf. Section 4) 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 and although the services are expected to work together in a concerted way in order to achieve the goals of the end-user they can be deployed and used independently, allowing their exploitation in different contexts.

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) [7]). Figure 1 depicts the high level services, which can thematically be grouped into 5 service groups plus associated APIs and CLIs.[1]

These services (which might internally be split in further services as described in Sections 5-10) are characterised by the scopes and enforcement of their policies as shown in Table 1. We distinguish between *user*, *site*, *VO*, and *global* (i.e. multi-VO) scope where combinations are possible (authorization policies may for instance be enforced by the VO and the site).

This taxonomy goes inline with recent discussions on resource virtualization (a short discussion of this issue is included in Section 11.3). The *physical resource*, be it computing, storage, or networking resource, is typically managed by a set of services under the control of the resource owner. These services need to provide the resource owner with enough capabilities for managing the resource, in particular allowing or denying access, quota management, and accounting and auditing. These physical resources

[1] Other categorisations (e.g. a layered architecture as discussed in [47]) are possible as well.

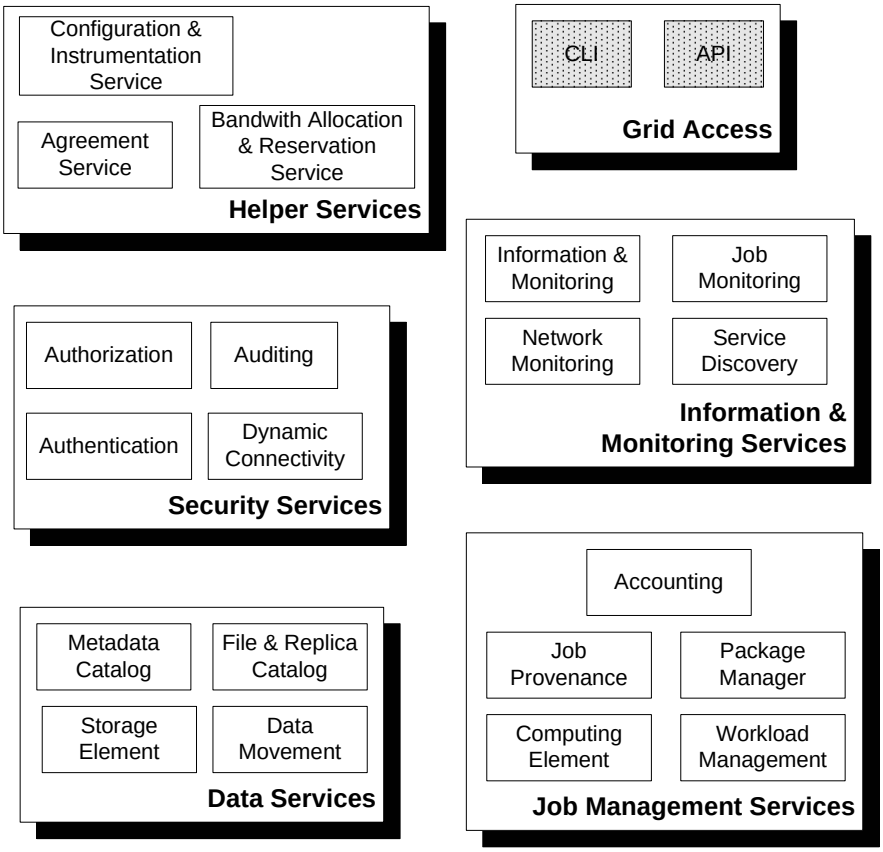

Figure 1: gLite Services

are *virtualized* by additional services providing the user with a virtual environment for using the physical resources. These virtual environments are typically tailored towards the needs of a user community, in our context a VO.

The whole area of virtualization is a relatively new field and the boundaries and relationships between physical and virtual environments is not yet fully specified. We try to take this issue into account in our following discussions of compute and storage elements, however, expect changes as the understanding of the field evolves.

Although most services are managed by a VO, there is no requirement of having independent service instances per VO; for performance and scalability reasons service instances will in most cases serve multiple VOs.

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 utilising 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.

Service	Scope			
	global	user	VO	site
Accounting	√		√	√
Auditing			√	√
Authentication	√			
Authorization			√	√
Agreement Service			√	
Bandwith Allocation and Reservation			√	√
Computing Element			√	√
Configuration and Instrumentation			√	√
Data Movement: Data Scheduler			√	
Data Movement: Data Transfer			√	√
Dynamic connectivity				√
File and Replica Catalog			√	
Information & Monitoring	√		√	
Job Monitoring		√	√	
Job Provenance		√	√	
Metadata Catalog			√	
Network Monitoring	√			√
Package Manager			√	
Service Discovery		√	√	
Storage Element: SRM				√
Storage Element: Data Access			√	√
Workload Management			√	

Table 1: gLite Services and their Scope

More specialised services, such as the Job Monitoring Service and Network Performance Monitoring services, can be built on top. 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.

However 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, 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.

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 signalling to running jobs), the usual technique of wrapping the executable content into "wrapper" scripts will be used. This technique, while not constituting a "service" as defined in Section 4, can be used at various levels in the architecture, as different parties, such as the submitting user and the workload management service itself, require the output of procedures that are inserted in the wrapper. When job wrapping requires services to be created or accessed by the submit/user interface node (before the service port for job submission is contacted), tools will be provided to make this operation transparent to the user.

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

In all of the data management services described below the granularity of the data is on the file level[2]. However, the services are generic enough to be extended to other levels of granularity. Data sets or collections are a very common extension where the information about which files belong to a dataset may be kept in an application metadata catalog. The usual possibility to group files by virtue of directories is provided. Most application data is expected to be located in files (as opposed to relational database systems for example). For application metadata we don't make this assumption. 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 totally transparent, the middleware will provide the capabilities for replica management. The Data Movement services will expose all nontrivial interfaces to the user for data placement in a distributed environment.

We expect the applications to specify the files they intend to access during their jobs either by name or by metadata query. The Metadata Catalog is application (and VO) specific, so the exploitation of existing legacy catalogs and their data needs to be supported. It is this mechanism that enables the concept of virtual datasets as well.

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 (as in AliEn) which can seamlessly navigate this virtual file system, listing files, changing directories, etc.

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.

Helper Services In addition to the services described above, a Grid infrastructure may provide a set of *helper services* that aim at providing a higher level abstraction (like for instance Workflow services or market mechanisms), better quality of service (like for instance reservation and allocation services), or better manageability of the infrastructure (like for instance configuration services).

In our current architecture we include three examples of such helper services, namely the *Configuration and Instrumentation Service*, the *Bandwidth Allocation and Reservation Service*, and the *Agreement Service*.

The Configuration and Instrumentation Service provides a common, standard-based configuration and instrumentation functionality to the gLite services. In particular, the configuration part is concerned with

[2]The issue of database access is briefly discussed in Section 11.2

dynamic changes in the configuration of a Grid service while the instrumentation part is concerned with obtaining the state of a Grid service.

The Bandwidth Allocation and Reservation Service provides mechanisms to control and balance the usage of the network and to categorise and prioritise traffic flows so that users and the layers of Grid middleware receive the required level of service from the network.

Finally, the Agreement Service allows the dynamic establishment of Service Level Agreements between agreement initiators (the service requestor) and the service providers.

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.) unless specifically pointed out in Sections 5-10.3. 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.

3 REQUIREMENTS

Due to the heterogeneous nature of Grid environments and Grid applications, a Grid architecture needs to cope with a large set of (sometimes conflicting) requirements. The OGSA document [81] gives a good overview on the nature of these requirements which are summarised below.

Heterogeneous Environment Support In general, Grid environments tend to be heterogeneous and distributed, encompassing a variety of operating systems, hosting environments, and devices. Moreover, many functions required in distributed environments may already be implemented by stable and reliable existing legacy applications which thus need to be integrated into the Grid. In order to support these requirements, resource discovery, resource interrogation, and life-cycle management of hosting environments are essential tools.

Resource Sharing Across Organisations One of the main purposes of Grid technology is to utilise resources transparently across administrative domains. Virtual Organisations provide the context to associate users, requests, resources, policies, and agreements without being limited by barriers such as existing sites or organisations. The Grid architecture needs to provide appropriate VO management, VO security, and accounting tools.

Resource Utilisation Grid environments must cater to chaotic usage patterns, making an optimal planning for resource utilisation virtually impossible. Nonetheless statistical and local optimisation of resource utilisation are still important; therefore tools for reservation, metering, monitoring, and logging are required so that system administrators can manage resources effectively.

Job Execution The Grid environment must enable submitted jobs to have coordinated access to VO resources and provide functions (such as job monitoring, analysis and projection of resource usage, dynamic adjustments) so that jobs can receive their desired quality of service (QoS). This requires tools for scheduling, job life-cycle management, work-flow management, and service-level agreement (SLA) management.

Data Services An ever-larger number of fields in science and technology require efficient processing of huge quantities of data. In addition, data sharing is important, for example to enable sharing information stored in databases which are managed and administered independently. A Grid architecture needs to provide services for data access, integration, provisioning, and cataloguing.

Security Safe administration requires controlling access to services and resources through robust security protocols and according to security policies. Thus, mechanisms for authentication, authorization, and auditing are required. Moreover, the Grid architecture should work across firewalls as well as allow for network isolation, delegation, and cross-organisational policy exchange and management.

Administrative Cost The complexity of administration of large-scale, distributed, heterogeneous systems increases administration costs and risks of human errors. Support for administration tasks should thus be automated, in particular for provisioning, deployment and configuration, application management, and problem determination.

Scalability Many Grid applications are concerned with scalability and performance, in particular through distributed supercomputing and high-throughput computing applications.

Availability Virtual Organisations enable transparent sharing of alternative resources across organisations as well as within organisations, and thus can be used as one building block to realise stable, highly-reliable execution environments. This requires mechanisms for disaster recovery and fault management.

Application Specific Requirements Apart from these generic requirements, Grid applications may have specific requirements that a Grid architecture needs to fulfil. In the context of EGEE the needs of two pilot user communities must be satisfied: High Energy Physics (HEP) and Biomedical communities.

The detailed requirements, which are mostly specialisations of the generic requirements presented above, can be found in several documents:

- Common use cases for HEP [9]
- Common Use Cases for a HEP Common Application Layer for Analysis [10]
- Joint list of use cases from HEP, Biomedical, and Earth Observation [40]
- Biomedical requirements [41].

EGEE Requirements The EGEE project has established a common repository for collecting, reviewing, and reviewing requirements from EGEE applications and operations. This repository can be found at https://savannah.cern.ch/support/?group=egeeptf.

This repository constitutes the main requirements that the architecture, design, and implementation work of gLite is planned against.

4 SERVICE ORIENTED ARCHITECTURE

Traditionally, applications have been built for a single computational entity, by integrating local system services like file systems and device drivers. Since everything is under local control, this model is very flexible in providing access to a rich set of development resources and provides precise control over how the application behaves. At the same time, this is bound to a single operating system and architecture, which can be error prone and costly, especially for upgrades. For distributed communities, it is not always possible to work in this mode with a single supercomputer as the computing entity.

A more modern approach is to construct complex distributed applications by integrating existing applications and services across the network, adding data entities, facades[3] and business logic. The aim is to reduce development time and increase productivity and software quality. This architectural model is powerful in the sense that it is very flexible and extensible, providing added functionality to the users. However, subsequent modification and architectural reuse of the components may be problematic as it may have complex repercussions on services built on top of them.

The term Service Oriented Architecture (SOA) is increasingly used to refer to a discipline for building reliable distributed systems that deliver application functionality as services with the additional emphasis on loose coupling between interacting services [75, 8]. Tight coupling makes it hard for applications to adapt to changing requirements, as each modification to one application may force developers to make changes in other connected applications.

[3] Provide a unified interface to a set of interfaces in a subsystem. A facade defines a higher-level interface that makes the subsystem easier to use. [36]

A SOA significantly increases the abstraction level for code re-use, allowing applications to bind to services that evolve and improve over time without requiring modification to the applications that consume them. Services provide a clean model to integrate software systems both inside the organisation and across organisational boundaries. This model is clearly very suitable for Grid middleware, which has to deal with Virtual Organisations and Site policies.

The services communicate with each other through well-defined interfaces and protocols, which can involve simple data passing or some coordination of activities.

4.1 SERVICES

If a service-oriented architecture is to be effective, we need a clear understanding of the term service. A service is a function that is well-defined, self-contained, and does not depend on the context or state of other services. A Web service is an application that exposes its features programmatically using standard Internet protocols.

In more detail, services are discrete units or modules of application logic that expose message-based interfaces suitable for access across a network. Typically, services provide both the semantics and the state management relevant to the problem they address. When designing services, the goal is to effectively abstract and encapsulate the logic and data associated with real-world processes, making intelligent choices about what to include and what to implement as separate services. Well-written services expose a semantically simple logic to the application that binds to them, with clean failure modes.

In a distributed environment, services tend to be oriented toward use over a network by other services, though this is not an absolute requirement. Communicating in a distributed system is intrinsically slower and less reliable than when operating in the same processing environment. This has important architectural implications because distributed systems require that developers (of infrastructure and applications) consider the unpredictable latency of remote access, concurrency issues, possibility of partial failure, and inaccessibility of certain services.

4.2 MESSAGES

Services interact by exchanging messages. Ultimately, every service is defined purely by the messages it will accept and produce, and what happens on failures. The routing of messages between services is a complex process that is best handled by a common messaging infrastructure.

Messages are sent in a platform-neutral, standardised format defined by the service interfaces. Service-to-service communication follows the interface contract; by making this contract explicit it is possible to change one service implementation without compromising the interaction. The internal structure of a service, including features such as its implementation language are, by design, abstracted away in the SOA.

For Web services, the Simple Object Access Protocol (SOAP) standard [64] is used to specify how the messages are being passed between services, and the Web Service Definition Language (WSDL) is used to specify the interface a service exposes. From a WSDL document a client or application will know how to bind to a service. Many semantic details are only defined as part of the internal logic and cannot be programmatically exposed; these details must be thoroughly documented.

4.3 POLICIES

All services are also governed by policies. Policies are less static than business rules and may be regional, organisational or user-specific. Policies are usually applied at run-time.

Policies may represent security, quality-of-service, management, and application concerns. Services negotiate using policies. Services must comply with one another's policy requirements in order to interoperate.

In a Grid the responsibilities for the management of the service policies is shared between many organisations and sites. Policies are typically managed by the Virtual Organisations (VO) using the service and the site administrators of the site where the service is actually deployed.

The breadth of the policy management rules needs to be considered at the design phase of the project as later changes to the system are difficult. If the policy-enforcement infrastructure is well-integrated, it allows setting, changing, and evolving the run-time rules that govern communication and service behaviour easily.

4.4 STATE

Services manage state, ensuring through their logic that the state is kept consistent and accurate. State manipulation is governed by the internal semantics of the service, also called business rules. These are relatively stable algorithms and are typically implemented as part of the service application logic.

Almost all services manage durable state; that is, state that is stored on some durable medium such as a file system or in a database. The services receive a request from another service, retrieve some state from that durable medium, and build a response or update the state. This durable state is important; services may be brought down and when they are brought up again, the durable state is still there and they can continue as if nothing has happened. Services do their best to keep that durable state consistent; they would like to keep their application state in memory consistent as well, but if something happens, they can just abort the processing, forget their memory state, and set up again using the durable state.

So in summary, a complete definition of services might be: Services are network-capable units of software that implement logic, manage state, communicate via messages, and are governed by policy.

5 SECURITY SERVICES

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. The following section provides an overview of these services; They are described in more detail in EGEE deliverables DJRA3.3 (Global Security Architecture) [20] and DJRA3.2 (Site Access Control) [21].

5.1 AUTHENTICATION

Authentication is concerned with identifying entities (users, systems, and services) when establishing a context for message exchange between actors: that is, it is the mechanism that enables you to know who you talk to. Such information is used in many policies for resource access and data protection, as well as for auditing and incident response purposes.

One of the key aims for Grid authentication is enabling single sign-on for the user, that is to provide an identity credential with "universal" value that works for more than one purpose, across many different infrastructures, communities, virtual organisations, and projects. Here, particular attention must be paid to the fact that the same identity is also to be used for accessing non-grid resources such as networks[4] and web resources. In fact, some regional deployments use their Grid credentials for secure (signed and/or encrypted) email as well.

[4] see Section 10.1 for a discussion on networks and the Grid

In our architecture, identities can be issued by grid identity providers, by independent non-grid entities (like governments or other national agencies), and by users' home organisations (major institutions and laboratories). Authentication (identity) assertions are independent of any authorization assertions and policies – there is no implied right-of-access associated with an identity.

The most used model for Grid security to date is based on each end-entity (user) holding unique authentication credentials. This system is based on Public Key Infrastructure (PKI), which uses a concept of trusted third parties (Certification Authorities or CAs) that issue the identity credentials. The set of CAs that we use for the EGEE deployment defines our trust domain: the pros and cons of this setup is further discussed in Sections 5.1.1 through 5.1.4. Single sign-on capabilities can be obtained by generating a proxy of the user's credential that is not protected by further activation data. That proxy credential may be based on a long-term credential held by the entity that should be protected by activation data. The proxy should be able to contain any assertions that are needed for establishing the security context when required by the session establishment protocol.

Thus a PKI based authentication system with proxy credentials (GSI[44]) will be the starting point for developing the authentication services. The user will be presenting his or her own identity credential proxies to the services, including any attribute assertions and relevant policies, and whenever actions are taken on the user's behalf these credentials will be transferred to the service invoked in order to retain secure traceability. Services that act, directly or indirectly, on behalf of, or as a result of, action by the user, shall hold credentials that are derived from the credentials of the original user.

Our chosen model allows for straightforward credential mobility, external (governmental) identity providers, and single sign-on capability over multiple domains even in the absence of any direct trust relationship between those domains.

Proper management of PKI credentials is too difficult and too complex for the average end-user to handle. Therefore, a recommended deployment is where the end-user have a specific identity and the capability to obtain credentials, but they are either kept on behalf of the user by some other entity (typically the user's home organisation using e.g. a hardened deployment of MyProxy) or implicitly in the user's infrastructure. Site-integrated credential services (SICS)[5] are services that generate (short-lived) user credentials without the user holding any long-term credentials. Several SICS implementations exist (such as Kerberized Certification Authority, kCA[70], and the Virtual Smart Card project [25]) and are described extensively elsewhere.

It should be noted that while our chosen model, where the user, at least theoretically, owns the credentials has most leverage in current grid-security implementations, this is by no means the only option available. In many systems, the user may attempt to access a service directly, and if authentication is required a security exchange will be initiated. Moreover, such a model allows for implicit privacy preservation: if authentication is not required that step may be omitted and the security context will then be established based solely on authorization data.

It should also be noted that the VO is not the appropriate guardian for the user's credentials, because this implies a proliferation of those credentials to many different entities (with the associated risks of compromising those credentials). Moreover, it entails the risk that this binding to the VO inadvertently leads to identities and authentication tokens being linked to a single VO, thus mixing authentication and authorization.

5.1.1 TRUST DOMAINS

The use of a PKI implies the existence of third parties (Authorities) that are considered "trusted" by all participants of the infrastructure. Traditionally implemented by a hierarchical structure, the absence of a single global root of trust for X.500 compels the use of a different structure. For the European

[5] This was previously referred to as Site-integrated proxy services, SIPS.

(and related world-wide) scientific Grid communities, a single trust domain has been established based on a policy defining a common set of minimum requirements. A policy management authority (the EUGridPMA[30]) has been established to coordinate this trust domain.

The trust anchors pertaining to the domain must be accessible to the services that validate authentication data in a secure, non-tamperable location (e.g. a local disk, modifiable only by the administrator).

5.1.2 REVOCATION

Timely revocation of identity is needed to prevent exploitation of credentials that have been compromised. The allowed response time as specified by resource providers is in the region of 10-60 minutes. Based on experience, we have concluded that this constraint cannot be satisfied by the periodic distribution of validity/revocation lists on a large scale.

Any software component that validated authentication must be able to check the validity of the credentials in real-time, using a suitably-sized mesh of status responders. The protocol for validating authentication credentials will be the Online Certificate Status Protocol, OCSP [28]. As a backup mechanism in case of network partitioning, revocation lists should be distributed periodically (4-6 times per day) and retrieved by all relying parties (users and service providers).

5.1.3 CREDENTIAL STORAGE

The usage pattern for credentials by users and services is slightly different. Services will keep credentials stored locally with the service but typically without activation data. Those services that are created on-demand (or on behalf of a user or process on a remote system) should use delegated credentials from either the originating system or the user responsible. Such a store of delegated credentials is to be managed as a collection of resources (delegation is described in the next section).

On the other hand, users are mobile and their credentials are not only more exposed but also much more "powerful" and attractive to exploit. Thus, long-term credentials held by the user must be stored securely and be protected by a passphrase. Several alternatives are to be provided: another trusted organisation may hold the credentials on the user's behalf, the credential may be stored on a smart card, or the user may not actually possess a long-term credential, instead short-lived credentials are created on-demand. Storing credentials on a local file system should be supported for backwards compatibility reasons, but implies a significant security risk and is not recommended.

In either case, the credentials must be issued by a trusted party close to the user (a certification authority, or the user's home organisation) and valuable credentials held in a secure storage under the user's control.

The user will likely have different ways of obtaining credentials, and may have more than one credential at the same time. Thus, a "credential wallet" function should be provided (e.g. MyProxy). As mentioned earlier, this credential wallet is valuable to the user and may contain sensitive personal data. It must be located where the user can trust the store (e.g. with the user's home organisation).

At all times, the proliferation of long-term authentication credentials should be traceable, so that all copies of the credential can be disabled or revoked when the credential is compromised.

5.1.4 PRIVACY PRESERVATION

One of the basic security requirements in the Grid is traceability of actions across domains. However, linking those actions to actual identities or to long-term reusable credentials exposes a significant amount of personal data that could violate privacy requirements. Moreover, by monitoring the very actions themselves may reveal commercially valuable information that should have remained hidden.

Due to the openness of Grids, privacy issues can only partially be addressed. In order to "raise the bar" a bit at least, we introduce pseudonym providers. Such pseudonym providers issue identity tokens in the same format as CAs in the common trust domain, but do require that they have CA "status" in order to function seamlessly. The related policy work for that to happen is currently underway in the EU GridPMA.

The pseudonym provider must keep a secure, non-tamperable and private account of the associations between issued pseudonyms and the original identities, which may be used for traceability and auditability of actions but only in a controlled fashion and for a well motivated reason (e.g. court order).

5.1.5 SECURITY CONSIDERATIONS

Credential stores, such as MyProxy, are high-value targets for directed attacks when used by many users. Hosting such services requires a trusted computing environment with dedicated machines, active firewalls, and minimal services. Moreover, these services should be compartmentalised so as not to store too many valuable credentials at the same time. Thus, the credential store must never be run as a common service by a VO, since a successful attack on the credential store will disable the entire VO for a long period of time. Hosting this service with the user's home organisation will both increase the trust level of the user in this service, as well as limit any damage by compromise to that single organisation.

5.2 AUTHORIZATION

Authorization (AuthZ) is concerned with allowing or denying access to services based on policies. The core problem with authorization in a Grid setting is how to handle the overlay of policies and other assertions from multiple administrative domains (user policy, VO specific policy, operational procedures, site-local policy), and how to combine them. Where relevant, both the service as well as the client should be able to take and enforce such decisions (mutual authorization). These issues are addressed in more detail in the sections below.

There are three basic authorization models [72], classified as *agent*, *push* and *pull*. In the *push* model, an authorization service issues tokens. The user collects the tokens and presents these to the resource where access is requested. While putting additional burden on the client, the resource does not need to know ahead of time about the user's privileges.

In the *agent* model, the user only interacts with the AuthZ server, which in turn forwards service-specific parts of the request to the underlying resources. While being a centric approach, network bandwidth or connectivity provisioning is best done in this mode, since access to the network in the end must be transparent.

In the *pull* model, the resource asks the AuthZ service on a need-to-know basis. This puts the burden on the resource, as it needs to know up-front all authoritative parties in the system, and how to contact them. Cellular phone roaming, and various RADIUS-based [26] network access services use this model.

The *push* model has been identified as the model that best suits dynamic, distributed and loosely coupled systems such as Grids.

5.2.1 SOURCES OF AUTHORIZATION

There are two kinds of AuthZ sources: attribute authorities (AA), which associate a user with a set of attributes in a trusted manner to a relying party, by way of digitally signed assertions. The relying party (the resource) evaluates the attribute assertions (e.g. role and group membership information) and include it as "evidence" when evaluating an access request against local policy.

The other kind of AuthZ source deals with policy assertions: here, the resources consolidate some of the policy definition to a trusted third party. The policy assertion service will issue claims that gives a user (or set of users) the explicit privilege to perform an action (or a set of actions) on a certain resource (or set of resources). The resource owners in turn may or may not decide to comply with these claims, pending the evaluation of conflicting local policy.

In our architecture, we foresee the need for both the AA and policy statement AuthZ services, and the use of primarily the push model. Initially, an AA service will provide coarse-grained division of users into different groups, which can then be handled by the resources through local configuration. However, with an increasing number of VOs using the EGEE resources, such configuration becomes unmanageable and we therefore will need a consolidation of VO policy that can be provisioned to the resources, e.g. on a need-to-know basis via the client requests.

The services and functionality are provided by

- the VO Membership Service (VOMS) [3], which is an attributes issuing service which allows high-level group and capability management and extraction of attributes based on the user's identity. VOMS attributes are typically embedded in the user's proxy certificate, enabling the client to authenticate as well as to provide VO membership and other evidence in a single operation.
- e.g. the Community Authorization Service (CAS) [66], a policy statement service, which issues its statements encoded in SAML [59].

Note: we do not consider the VO policy server component to be a crucial piece of the core architecture, but we mention it anyhow at this point as it is on the future road-map.

5.2.2 POLICY COMBINATION AND EVALUATION

When making an AuthZ decision, we must be able to combine information from a number of distinct sources. Besides Grid-wide or VO-wide attribute and policy assertions, potentially provided to a service as part of the client request, we also need to include domain specific policy such as POSIX ACLs for file access, or a (sub-)set of VOs that are allowed access to a particular computational resource. Finally, but most importantly, we must also take any locally defined site policy into consideration when making a unified, context specific AuthZ decision on an individual request basis.

In order to be able to combine information from multiple sources in this manner, we must have a way to convert any domain specific access control language into a common language with strong support for combining policies. One such candidate language is XACML (eXtended Access Control Markup Language) [58].

Figure 2 shows the architecture of a general framework that is able to fulfil the requirements set out above. It allows the definition and enforcement of local, VO and Grid wide policies. The *Policy Administration Tool* (PAT) allows the local administrator to specify, remove, modify policies, and accept or refuse policies proposed by an external entity (for example policies defined at the Grid level or set by the VO administrators). The *Policy Communication Interface* (PCI) is used to handle the communication between different "entities", for example to send VO-level policies to the various CEs. The set of active (and past) policies, possibly described in the combined form using standard languages such as XACML, is held in the *Policy Repository* (PR). In this architecture the *Policy Enforcement Point* (PEP) is used by the client (e.g., the Web Service) to make the policy evaluation of a request. The PEP assembles the necessary *evidence* (asserted attributes, and other contextual data), and then forwards the request to a *Policy Decision Point* (PDP) which evaluates the request according to the active policies.

We note that for performance reasons, the PEP and PDP will in most cases be merged into a single component that is embedded in the request handling flow of the service container, rather than being exposed as separate, standalone Web Services.

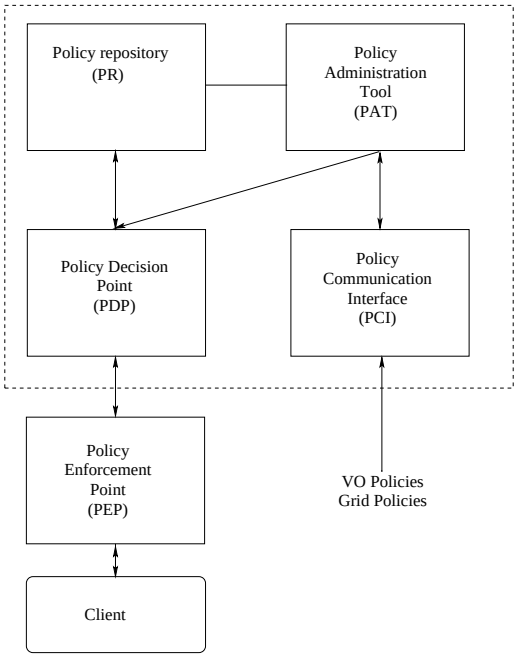

Figure 2: Architecture of the policy combination and authorization framework.

A typical use case of how the policy combination and authorization framework is used is consulting VOMS attributes and a local grid-mapfile, i.e. attribute assertions and policy from two authorities (the VO and the local resource owner) that need to be combined into a final authorization decision. In addition to this, operational policies that states things such as max allowed credential lifetimes, needs to considered as well.

5.2.3 MUTUAL AUTHORIZATION

There are Use Cases where a client wants to authorize a service as well as the service authorizing the client access: for instance, a client wanting to store sensitive data might first ensure that the SE has been approved to service such storage requests. Instead of building this tightly into every service (for instance, by having services authenticate with proxy certificates with embedded authorization information), optional means (e.g. portType) must be provided by the services to interrogate the service about such authorization information.

5.2.4 OTHER SERVICES USED

While it is possible to provide anonymous AuthZ tokens (such as tickets to a ball game), attribute assertions and policy statements are usually tied to a globally known identity for convenience, traceability and ease of management. Therefore, the AuthZ services will have a strong dependency on the Authentication infrastructure which provides the necessary unique identifiers and the cryptographic methods with which the entity-identity association can be challenged and proved.

Whenever a policy statement or an attribute assertion is created or used as evidence, a trace of this action

must be found in the system. Thus, AuthZ services and policy evaluation engines will depend on and make use of available audit and logging services.

5.2.5 SECURITY CONSIDERATIONS

It is vital that all components of the AuthZ infrastructure are as securely managed and operated as the corresponding parts of the authentication infrastructure. For instance, the repository containing the trusted signatures of the AAs need to be as protected against infiltration as the repository of trusted CAs, e.g. by distributing these so-called trust anchors via an authenticated web site, possibly cross-checked against widely available public media.

5.3 DELEGATION

It is often the case that Grid users need to delegate some subset of their privileges to another (dynamically created) entity on relatively short notice, and only for a brief amount of time. For example, a user needing to move a dataset in order to use it in a computation may want to grant to a file transfer service the necessary rights to access the dataset and storage so that the service can perform a set of file transfers on the user's behalf. Since such actions may be difficult to predict, having to arrange delegation ahead of time is prohibitive.

An important security aspect in regards to delegation is the principle of least privilege: you only want to delegate the privileges that are necessary to successfully perform the intended operation, and no more. This is hard to accomplish in reality, but our architecture must support some simple provisioning and enforcement of restricted delegation. It is also often very useful to separate the privilege to delegate from the privilege itself.

A number of existing mechanisms could satisfy the delegation use case mentioned above. The use of Proxy Certificates [78] has been the mechanism most widely adopted by the Grid community to date, as the technology needs no additional infrastructure services, and at the same time it also solves the single sign-on and dynamic entity identification problems. Besides providing some coarse-grained controls (such as describing whether all or none of a user's privileges are implied in the delegation, and the right to delegate further), there is also a placeholder for adding arbitrary, typically application-specific, policy restrictions.

5.4 SANDBOXING

Many of the applications using the grid today are *legacy* applications, *i.e.* applications that are designed and implemented for running by a local, trusted user on a single administrative domain. When running these essentially 'arbitrary' applications on remote resources, both users and resource owners need isolation of those applications. On the one hand, that enables resource administrators to control the behaviour of these unknown applications, and on the other hand it protects to some extent the different applications running on a shared resource from each other. The choice of isolation technology made at the resource level, should however be transparent for the higher-level authentication and authorization modules, the middleware components initiating this application, and as far as possible also for the applications themselves.

Theoretically, complete virtualisation of resources provides the best possible way of isolating applications. Such virtualisation of the resource is common in some hosting environments like Java. For traditional applications, the same or a similar level of isolation can be obtained by using host virtualisation techniques. In practice, much of the legacy application isolation is accomplished by exploiting traditional Unix-level security mechanisms like a separate user account per 'grid-user' or per job via credential mapping techniques. All information relevant to the mapping of credentials, the credentials

created or linked, and the associated attributes will be recorded by a Job Repository. Details can be found in DJRA3.2 (Site Access Control) [21].

5.5 Dynamic Connectivity Service

As discussed in Section 5.2.2, our design must recognise and respect policies defined by the local sites. Such policies may affect connectivity as resources (storage, worker nodes) may be on a private network, NAT devices or firewalls may block incoming and/or outgoing connections, or certain port intervals may be blocked. [56].

In order to tackle the problems that arise when connectivity is restricted by the resource owner, provision for a *Dynamic Connectivity Service* (DCS) component was identified early in the design process. First, it was envisaged that a DCS would act as a virtual endpoint for service-level translation device/routing of messages between the Internet and the actual endpoint on a local/private/shielded network.

However, such as service would effectively override/bypass local policy, which is not in line with the overall security design philosophy. Furthermore, most of the middleware services communicate over standard protocols for which proven solutions already exist.

Therefore, our initial approach is not to build a DCS ourselves. Instead, we will ensure that our service container(s) and services themselves can be configured such that the local resource owner can control how the services communicate:

- Only using ephemeral (dynamically allocated) ports in a certain port range
- Advertise endpoint URLs with hostname/port numbers different than the physical local addresses and logical hostname
- Route all SOAP over HTTP(S) traffic via a SOCKS server, provided by the resource owner.

For the longer term, we will investigate alternative solutions to this problem. One could also envision a DCS as a front-end service to the local network access policy, with the capability to authorise and trace network access by modifying that policy dynamically. For example, before initiating a data transfer, the application would need to request for the creation of a network path on which it can send or receive. The DCS would then alter the firewall (or port filtering application) configurations to allow traffic through, and log the event for auditing purposes. It has also been suggested that other transport protocols such as BEEP [64] are investigated.

5.6 Auditing

Auditing is a very general term: here, we primarily mean the system security aspects of auditing, such as monitoring and providing for post-mortem analysis of security related events.

In computational Grids, auditing goes hand in hand with accounting, as they share the base requirements on the system's logging capabilities: *who did what, where, and when* (and in the case of accounting, *for how long, or for how much*). [42]

Auditing is not only meant for emergency use, such as a system breach, but is also useful for the validation of continuous operations, verifying that components behave as expected.

5.6.1 Services

The auditing process in itself does not require a separate service. Rather, it imposes a set of common principles and practices on all system components, e.g. to log correct, complete and relevant information.

The audit information should itself be traceable and verifiable throughout its lifetime and throughout service invocations, so that any anomalies in the audit trail can be traced to identifiable service components.

Creating a “real” audit service would clearly have benefits in terms of enforcing uniformity of information, providing secure remote access to the information, easing the process of merging and combining the information from different services. No specific architecture for such a component in currently pursued due to doubts about load, scalability and, not the least, the handling of site-internal information outside the site perimeter.

5.6.2 Other services used

In a (provisional) auditing scenario, we will make use of whatever information sources we have at hand (system logs, L&B services, ...), and attempt to make sure the other middleware components log the necessary information.

5.6.3 Security considerations

Ideally, for a system to have a “proper” auditability according to recognised standards, we would need to invest heavily in detailed documentation of processes and methodologies used, as well as employ additional technology (for example tamper-proof logs). This is more than we can consider in this project.

The audit information stored anywhere in the system should be self-contained and interpretable even if the original source of the audit information is later compromised. Moreover, the information should be clear and concise even if it is to be interpreted without access to the originating service.

6 Grid Access

All of the gLite services are accessible via APIs and CLIs. The move towards web services enables the automatic generation of APIs in multiple programming languages. This is however a tedious and error prone task due to the lack of appropriate tooling. Hence pre-generated APIs are provided as well. These APIs can also offer higher-level functionality than the service WSDL itself.

CLIs are provided for the most common service interactions.

7 Information and Monitoring Services

7.1 Basic Information and Monitoring Services

The information services are a vital low-level component of any grid; most services will publish or consume information. The information service should create the appearance of a single federated database covering all available information. This implies a query language that supports ad-hoc correlations between information in different services. A candidate for the model is the relational model with SQL as the query language. This is not to suggest that end users need to construct complex SQL queries themselves as it is more common to have front ends to automate the generation of SQL.

Any information can be monitored provided it carries a time-stamp. The mechanisms to move the information around are the same. What makes monitoring systems distinctive is normally the GUIs that are provided to visualise time-sequenced data and to highlight problems. These GUIs are simply clients of the information services.

Each VO is assigned a number of name-spaces corresponding to a Virtual Database (VDB). Each VDB is associated with a registry and a schema. The schema holds the definitions of the tables and the registry

lists the available sources of information (producers) and the consumers. A registry contains a list, for each table, of Producers who have offered to publish for the table. A Consumer runs an SQL query against a table, and the registry selects the best Producers to answer the query in a process called mediation. The Consumer then contacts each Producer directly, combines the information, and returns a set of tuples. The mediation process is hidden from the user. Note that there is no central repository holding the contents of the virtual table; it is in this sense that the database is virtual as it holds virtual tables. Some of these components are show in Figure 3.

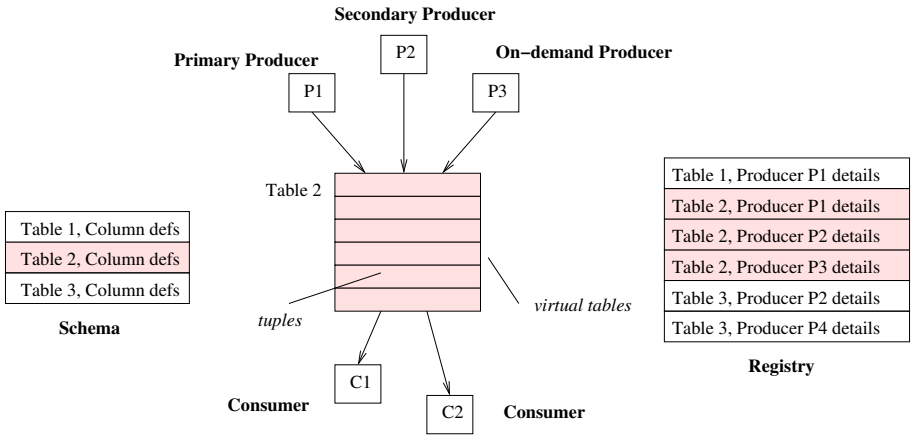

Figure 3: R-GMA and the VDB

Note that there is no global information. Each schema has an owner who controls who can publish what and can read what. If a deployment wishes to simulate some global information a VDB would need to be created. Service providers may choose to publish information to VDBs of the VOs they serve or they may publish to “service provider VDBs”. This is a deployment issue.

The contents of the scheme are not an architectural issue, however we will maintain the relational mapping of the GLUE schema.

A Consumer can pose queries against multiple VDBs using a syntax to indicate that the union of the tables of the specified VDBs should be used.

The registry and schema are vital components of the system though most of the time users just make direct use of the producer and consumer services. The schema is accessed directly to define new tables and to attach authorization information to a table.

Some information of interest changes rapidly and some much more slowly. However, even with the slowly changing information, it is often necessary to know quickly if it does change. Publishing information that is only changing infrequently, along with rapidly changing information is inefficient. This requires thought when designing schemas. It is better to treat the information as two or more entities with one to one relationships between them at any one time, rather than trying to bundle together slowly and rapidly changing quantities.

The Services to be provided are:

- Producers (Primary, Secondary and On Demand)
- Consumer
- Registry
- Schema

7.1.1 OTHER SERVICES USED

VOMS is able to augment a user's certificate with information about groups and roles within a VO. The use of VOMS, or a similar service, facilitates the expression of rules to specify the desired granularity of access.

7.1.2 PRODUCER SERVICES

The Producer services are used to publish information. There are three types of Producer:

- Primary
- Secondary
- On-demand

All behave in different ways, as described below, but have some common features.

Users of Producers declare tables to advertise the type of information that will be made available. The user can also specify a predicate (SQL `WHERE` clause) that defines the precise subset of the global table that will be published. In a Grid environment it is not common for all information contributing to one table to be published from a single source.

Each table has a number of system defined columns which are readable by the user, and a time-stamp column which may also be written by the user to override the automatic time-stamping of the tuples.

The user introduces new data into the information system by inserting it into a Primary Producer. Primary Producers are the initial source of the data. They may be thought of as having two internal stores, one for latest queries and one for other queries.

Producers may be implemented using different persistency technologies to store and organise their internal stores according to need. These include memory for speed and RDBMS to support both system recovery and efficient joins.

Primary Producers transmit tuples to all listening Consumers, including those within Secondary Producers. A Secondary Producer is used to aggregate streams of data and often at the same time make them persistent. They do this by setting up a Consumer to retrieve data for each declared table and republishing the data. Secondary Producers are created by specifying a predicate which defines the subset of a table to be collected and republished along with the persistency attributes associated with the republished data. As the user cannot directly publish new data using a Secondary Producer, it has no insert operation. The Secondary Producer has several uses:

1. It collects and maintains information in one place. This avoids querying Producers individually or accessing remote information sources.
2. It can be set up to record historical data and thereby act as an archiver of information.
3. By moving information to one place it allows joins to be performed over tuples which were originally scattered..

An On Demand Producer interacts with a user-supplied plug-in that returns data in response to an SQL query. It is used when the cost of publishing all the data would be too high compared to retrieving data only when it is requested by a Consumer. The plug-in can be rather complex to construct, and some limitations on the SQL it can accept may be imposed.

7.1.3 CONSUMER SERVICE

The Consumer Service is used to obtain data published by one of the Producer services. A Consumer handles a single query, expressed as an SQL SELECT statement. If you want to use a new query you must create a new Consumer. The Consumer also identifies how the query is executed, referred to as the query type. Depending upon this query type, the Consumer will run its query in one of two ways. The first is called a Continuous query and the second is referred to as a One-Time query. Continuous queries stream data from Producers. When a new tuple is published, the tuple is copied to all interested Consumers. This approach allows a Consumer to keep up-to-date with all Producer events. Conversely, one-time queries involve a single request/response for the Consumer to get information from a Producer. One time queries may in turn be either Historical or Latest; a Historical query has access to all the information and the Latest query only considers tuples with the most recent time-stamp for a table's primary (but without time-stamp) key. For all query types, the user can specify how far back in time to start. In addition there are a pair of retention periods to consider. One retention period is associated with history and continuous queries and is a property of the producer of that table. This is necessary to control the amount of historical data stored by a producer. The latest retention period is associated with the tuple and is not changed as data flows through the system. Latest queries never return tuples which are older than their latest retention period.

For all queries, retrieved tuples are stored within a Consumer buffer, managed by the Consumer service. The user can extract buffered data by invoking the various pop operations available in the Consumer API. There is no callback mechanism. The user must poll the Consumer to obtain the data transferred asynchronously to the Consumer buffer.

Normally the Consumer uses mediator functionality within the Registry, to find the set of Producers to answer a query, however a user may direct a Consumer to a particular set of Producers, if so desired.

7.1.4 REGISTRY AND SCHEMA SERVICES

The registry and schema between them define a "virtual database" (VDB) i.e. a separate namespace. A VO may have more than one VDB to manage. The registry and schema are both distributed and replicated to provide scaling, resilience and performance. The replication algorithms are slightly different because of the different behaviour required of the two services. The registry is of little direct interest to the end user however the schema is used to create and destroy tables and to associate authorization rules with a table.

7.1.5 BOOTSTRAPPING

The various R-GMA services will publish their availability for discovery as described in 7.3. This is fine when the system is running, but how do we install a new machine or a new VO? An R-GMA client will simply use the services on the local R-GMA server. The local server will run a proxy for the registry and schema if it does not have one locally. An R-GMA server will periodically use the Service Discovery (without caching) to locate all the services available and combine the results with the previous known set of services and save the results in a file or files. R-GMA is resilient against missing services so we can afford to keep a few dead services rather than risk losing the last known service from the list. A VO must set up its first machine or machines manually. It is then a deployment issue for other machines to get the initial set of R-GMA services and store them in the startup file. The VO could post them on a web site or e-mail them to site administrators. Provided that this list contains a functioning instance of one of each kind of service for that VDB, the system will then bootstrap itself.

7.1.6 SECURITY

Authorization rules, which are local to a VDB, will define what actions an individual (certificate holder) may carry out. This includes the ability to publish information (via a Producer), to query (via a Consumer) or to discover what Producers exist. The authorization rules are passed to the createTable method. This holds a set of rules, each one a pair: (View : AllowedCredentials). Each View defines a view on a table in the form of a SELECT statement. If you match the allowed credentials you will have read access to the data defined in that view. It is possible that your credentials match two rules in which case you will be able to see the union of the two views. If you issue a query to see data you are not allowed to see, you will only be returned results derived from the data you are allowed to view. Both the View and the AllowedCredentials are parameterised in terms of VOMS attributes.

7.2 JOB MONITORING

Within a Java application, the natural way for a job to publish information about its progress is to use the Java logging service or log4j[34]. This makes use of components known as *appenders* to handle the log messages. Following the success of log4j, facilities supporting applications in other languages have been developed and now we see a new Logging Services[35] activity within Apache which *"is intended to provide cross-language logging services for purposes of application debugging and auditing."* A tool, Chainsaw[33], is also part of the Apache logging service project. This is able to pick up logs and visualise them.

From an application point of view, unstructured messages are published using the normal logging API, which makes use of an R-GMA appender to publish the data via the information system. The set of attributes which can be published will follow the log4j specification. As indicated, log4j messages are unstructured. If structure is required then the information and monitoring service should be used directly.

Facilities to make the data available to Chainsaw (or equivalent) will also be provided for visualisation.

7.2.1 OTHER SERVICES USED

Information and Monitoring will be used to publish the logging information.

7.2.2 SERVICES

This is not a separate service but rather an API making use of an R-GMA appender.

7.2.3 SECURITY

The underlying information system must protect job information from those not permitted to access it. The Authorization cannot be specified via the API as it is not part of the existing logging APIs, so instead the information will be passed directly to the component publishing the information into the Grid.

7.3 SERVICE DISCOVERY

Service Discovery is a facility offered to both end users and to other services to locate suitable services. It is a front end to some kind of directory system. We will support multiple pluggable backends. Initially this facility will be offered as a client library and not as a service as it is more efficient and avoids the problem of discovering the service discovery service. It is intended to be lightweight and simple to use.

The backends can make use of the (R-GMA) information and monitoring service, bdII, file, and UDDI. The selection of information service backend or backends will be by static configuration, under program control or by interrogation of an environment variable.

Initially we will assume that the necessary information is published by the services. A mechanism, similar to the service discovery one, needs to be defined for service registration as well.

In addition there will be an optional caching mechanism to provide resilience. Results of successful queries are saved in the cache but will *only* be used if the information and monitoring system should be unavailable. This caching mechanism is independent of the backend, though it would not be useful with a file based backend. This cache may be either written to for each lookup or periodically the complete service information could be looked up and stored in a file or files available to the client. Both approaches have their advantages.

The API will allow any of the properties of the service to be used to select a list of suitable services. They will be returned in a pseudo-random order to avoid the tendency to keep using the same service.

7.3.1 Other Services Used

This will make use of the Information and Monitoring services, using producers to publish available services and consumers to look up services.

7.3.2 Services

Initially service discovery will be offered as a client library. In future it may also be offered as an explicit service.

7.3.3 Security

It is not yet clear whether or not the existence of services should be hidden. We assume that there is such a requirement.

The information and monitoring services provide an authorization mechanism which can be used easily for the underlying information and monitoring implementation.

The difficulty comes with the caching layer. The caching must of course be on the client. It could be made read only (at the file system level) by the user who obtains the information.

7.4 Network Performance Monitoring

Within EGEE the users of network performance data are NOCs/GOCs and the gLite middleware. This data is used by NOCs/GOCs to maintain smooth operation of the Grid, and is used by gLite to inform the decisions made by components of the middleware.

Network performance data is made available through monitoring frameworks running on the network. Most of these frameworks expose their data through mutually incompatible interfaces. It is the aim of NPM to define a standard interface to such frameworks. By doing this, and by providing intermediary services, NPM clients can access network performance data without needing to be aware of the type and location of monitoring frameworks generating network performance data.

The usefulness of network monitoring services depends on the amount of the EGEE network covered by network monitoring frameworks. Monitoring data on EGEE is currently exposed to the NPM services by two different monitoring frameworks: WP7/R-GMA [19] and Perfmonit [16]. WP7/R-GMA was developed during EDG and gathers end-site to end-site data. Perfmonit is a prototype network monitoring

framework developed by GN2 [37] to gather backbone data. It should be noted that other monitoring frameworks are free to implement this interface and so provide their network data to the network monitoring services.

Two network monitoring architectures have been defined:

- NPM Mediator, for use by GOCs/NOCs; and
- NPM Publisher, for use by gLite components.

These are described in subsequent sections.

7.4.1 INTERFACE TO NETWORK MONITORING FRAMEWORKS

NPM defines a standard interface to network monitoring frameworks based on schema created by the GGFs Network Measurements Working Group (NM-WG) [57]. The NM-WG schema provides a standard mechanism to describe and publish network performance data to the Grid.

The monitoring framework interface consists of one method: a request for historical network monitoring data. This method takes an NM-WG request and returns and NM-WG response (or error).

Within EGEE, this interface must be implemented as a web service providing access to data generated by the underlying monitoring framework.

7.4.2 NPM MEDIATOR

Overview NOCs and GOCs must diagnose network problems that occur on the network. For this they need access to current and historical network performance data from end-sites and from the backbone network. The NPM Mediator provides access to this data.

The NPM Mediator takes a request for network monitoring data and returns the result from the network monitoring point (either end-site or backbone) capable of answering the request. Each of these network monitoring points exposes a web service that implements a single known interface.

For data from the backbone network, if no single backbone site can satisfy a request the NPM Mediator will attempt to aggregate data from a number of suitable backbone sites.

Though the NPM Mediator provides information on the network, NOCs/GOCs must perform post processing to analyse this data. As such a diagnostic tool will also be provided to help bridge the gap between network monitoring data and analysis of such data.

The architecture of the NPM Mediator is shown in Figure 4, and is described in the following sections.

NPM Mediator Web Service Interface The NPM Mediator web service interface provides access to the functionality exposed by the NPM Mediator.

Aggregator The aggregator forms the heart of the NPM Mediator. It deals with all requests from the NPM Mediator web service and coordinates subsequent activity within the NPM Mediator.

On receipt of a performance monitoring request, the aggregator will check if a response to the request already exists within the response cache. If such a response exists it will be returned. If such a response does not exist, the aggregator will use the discoverer to locate the network monitoring point web service that can satisfy the request. The aggregator will then contact the network monitoring point web service to obtain the requested data.

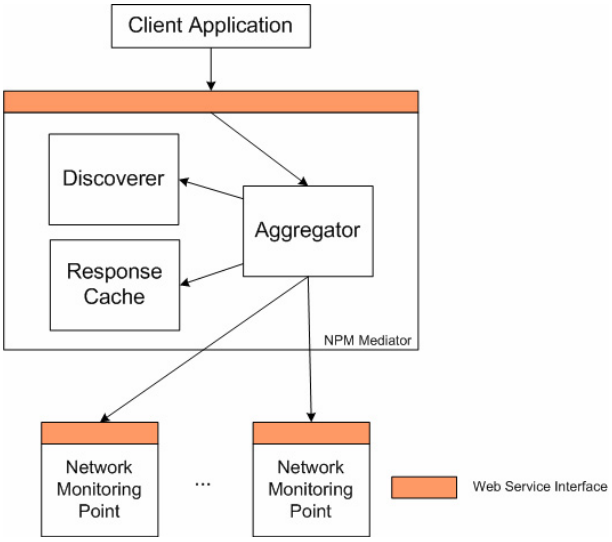

Figure 4: NPM Mediator Architecture

It may be the case that a request for network performance data between two points cannot be satisfied because an explicit measurement has not been made between these points. On the backbone it is possible to calculate such measurements by aggregating explicit measurements between hops that make up the route between the requested points. In this situation the discoverer will return more than one service. The aggregator will gather the results from each service and then aggregate them into a single result which satisfies the request. It should be noted that such aggregation is not possible for end-site network monitoring data.

The feasibility of aggregation of network data is a research topic within the network community. The inclusion of such functionality in the NPM Mediator is intended to allow our network partners to experiment with aggregation and feed into their discussions. As such, this is low priority functionality and may not be present in the delivered solution.

Once a result has been obtained (either directly or through aggregation), it will be added to the cache and then returned to the NPM Mediator web service interface.

Discoverer The discoverer is responsible for locating a network monitoring web service that can satisfy a particular network monitoring request. For backbone sites, it may be that no single network monitoring site can answer the request: for example with a cross backbone domain request, it may be that each backbone monitoring site only gathers data for the ingress and egress points of the domain. In this case it would be necessary to build up data from that provided by each domain. For such cases the discoverer will return a list of the network monitoring web services that must be aggregated to resolve the request.

It is possible that the Discoverer will use the EGEE Service Discovery component described in Section 7.3.

Response Cache The response cache provides a limited local repository of network performance data to enable timely access to responses for common requests. The cache will be space-limited, and will have policies that keep responses to common requests within the cache, while discarding responses to infrequently made requests. Entries within the cache will also be time limited, so that the cache does not contain stale information. The exact lifetime allotted to cache entries will be determined empirically when the cache is implemented.

Security Figure 5 shows the overall security architecture for the NPM Mediator. The entities involved, their certificates, and the information stored pertaining to authorisation are shown.

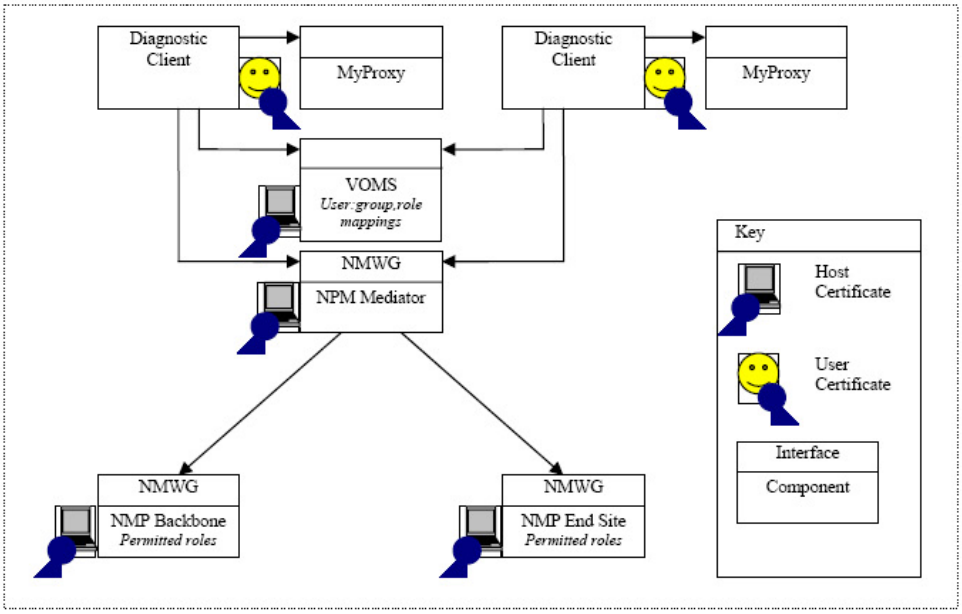

Figure 5: NPM Mediator Security Architecture

Security is required at the following levels:

- Between the Diagnostic Client and the Mediator, to ensure no-one can alter or intercept the data, and to ensure that only authorised users can make use of the Mediator
- Between the Mediator and the Network Monitoring Points (NMPs), to ensure no-one can alter or intercept the data
- Between the Diagnostic Client and the NMPs (via the Mediator), to ensure only authorised users can make use of the NMP facilities
- Between the Diagnostic Client and MyProxy to ensure only authorised users can add their certificates, and authorised users can only receive back proxies for their own certificates
- Between the Diagnostic Client and VOMS to ensure authorised users only receive attribute certificates for permitted attributes

7.4.3 NPM PUBLISHER ARCHITECTURE

Overview gLite requires information on the state of the network in order to inform decisions on where to place jobs on the Grid and from where to retrieve data on the Grid. To ensure the performance of the middleware such network monitoring information must be available quickly. As such, gLite is expected to retrieve network performance data from a Grid Information System (GIS) owned and run by gLite. The NPM architecture will publish network monitoring data into this GIS. The data published into the GIS will allow gLite to easily retrieve any end-to-end measurement it requires. A GIS is used as an intermediary between gLite and the NPM publisher for efficiency reasons: GIS are designed to efficiently handle a high volume of queries.

The main component in this architecture is the NPM Publisher. This component has knowledge of the end-to-end measurements required by gLite. At regular intervals the NPM Publisher will use the NPM

Mediator to request these measurements and publish them into the GIS. Initially, the location of the NPM Mediator service instance will be well known to the NPM Publisher. In the future the service registration and discovery features discussed in Section 7.3 will be used.

The architecture of the NPM Publisher is shown in Figure 6, and is described in following sections.

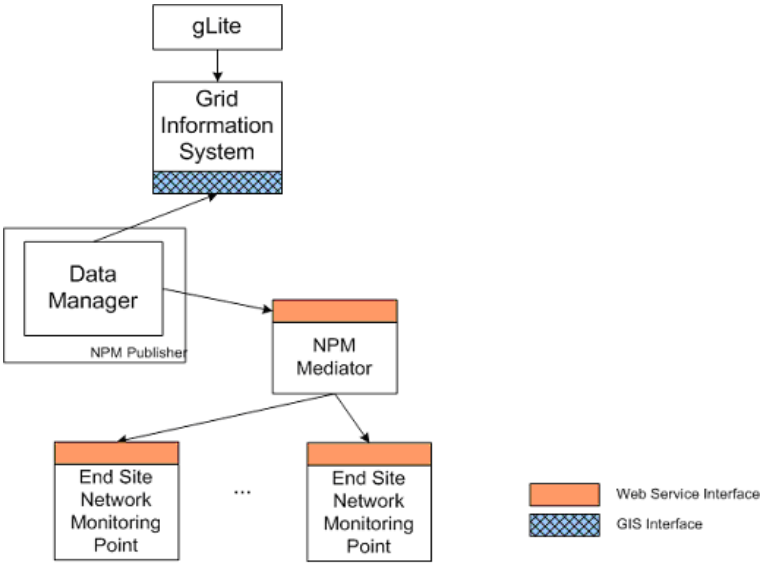

Figure 6: NPM Publisher Architecture

Data Manager The data manager is responsible for collecting network monitoring data from the end-sites and publishing these data to gLite's GIS.

At regular intervals the data manager will contact the NPM Mediator to retrieve end-site data and publish it to the GIS.

Errors encountered by the data manager while accessing the network monitoring web services or the GIS will be logged in a file at the site hosting the NPM Publisher. This will allow administrative staff at that site to diagnose problems that may occur during normal operation of the NPM Publisher.

Security Figure 7 shows the overall security architecture for the NPM Publisher. The entities involved, their certificates, and the information stored pertaining to authorisation are shown.

Security is required at the following levels:

- Between the Publisher and the GIS to provide reliable and secure information to the GIS
- Between the NMPs and the Publisher so that only NMPs that wish to publish information to the GIS will do so
- Between the Publisher's Registry of NMPs and the NMPs so that only valid NMPs can be registered to provide information to the GIS. Currently, the Publisher's Registry is a static list of sites and the mechanism for dynamic registration has not been determined, so this document revision does not consider this security level.

8 JOB MANAGEMENT SERVICES

In this section the job management services are presented.

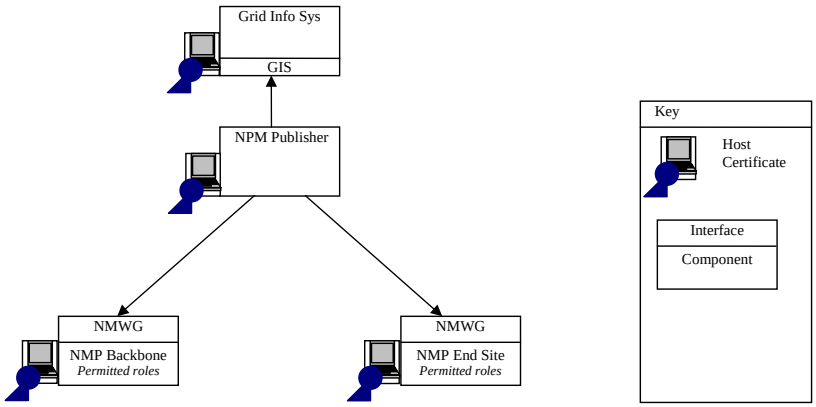

Figure 7: NPM Publisher Security Architecture

Subsect. 8.1 discusses about the accounting service, responsible to collect, manage and present information about the usage of Grid resources by the users or by groups of users. Subsect. 8.2 reports about the Computing Element (CE) service, which represents a computing resource where user jobs get executed. In Subsect. 8.3 the Workload Management System (WMS) is presented: it encompasses a set of components responsible for the distribution and management of tasks across the available Grid resources. In Subsect. 8.4 the Job Provenance service (JP) service, responsible to keep track of the definition and execution conditions of the submitted jobs, is presented. Subsect. 8.5, which discusses about the Package Management (PM) Service (able to automate the installation, upgrading, configuration, and removing of software packages on a Grid site) concludes this section.

8.1 ACCOUNTING

The accounting service accumulates information about the usage of Grid resources by the users and by groups of users, including Virtual Organisations as groups of users.

This information allows preparation of statistical reports, to track resource usage for individual users, to discover abuses and to help avoid them. Accounting information could be used to charge users for the Grid resources they have utilised.

The information available from the accounting service can also be used to implement submission policies based on user quotas or on resource usage (fair share). In principle it also allows the creation of a real exchange market for the Grid resources and services. The subsequent economic competition should result in market equilibrium, thereby promoting load balancing on the Grid.

Among the services from other Grid projects that can be compared with the EGEE Accounting Service it is worth mentioning Nimrod-G [4] and the EDG DGAS [67].

8.1.1 RESOURCE METERING

Figure 8 represents the actors involved in the “Grid accounting” schema. The key elements are the Grid resources and the Grid services. In order for an accounting system to work it is necessary to have reliable information about the Grid resource usage, thus each Grid resource or Grid service needs to be instrumented with dedicated sensors in order to measure the usage of the resource or service. This step of the process is the “resource metering” activity. The number of different type of resources is, in principle, very high. It is therefore necessary to foresee many different types of base metering sensors, that in the figure are shown as “Dedicated resource metering”. Each type of resource will have its dedicated sensor. For example, different types of Local Resource Management Systems (LRMS) in a CE need different

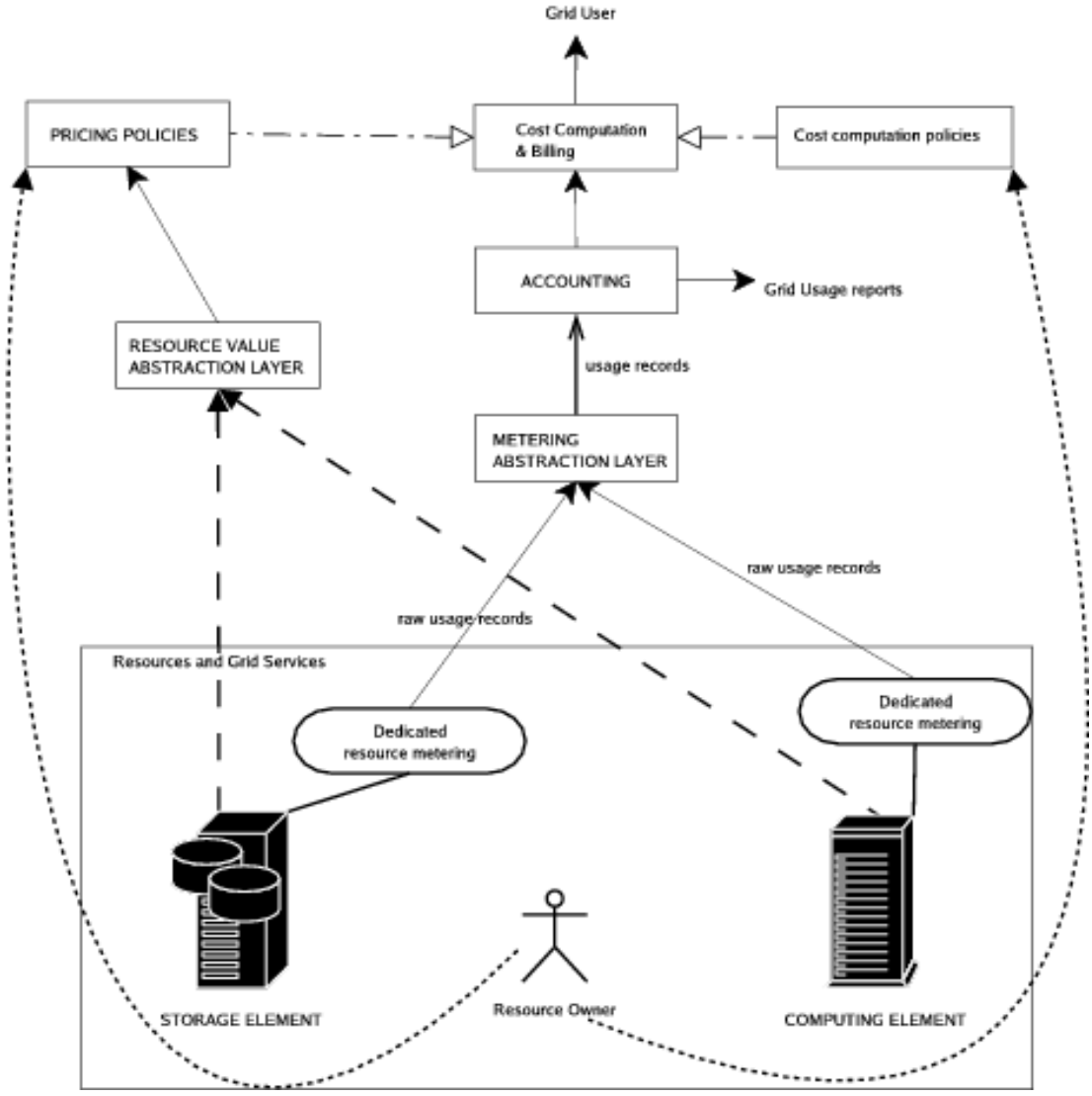

Figure 8: Fundamental actors in the provision of an accounting service.

sensors and the metering infrastructure will also be different for a disk based Storage Element than for a tape library. It can also happen that there will be different implementations of a metering info provider for the same resource type, to preserve the freedom of the resource owner to choose one of the many available implementations, or to create a new one.

To allow the operation of the accounting layer, any distributed metering system has to report a basic set of information:

- A globally unique identifier of the user job or service request.
- A globally unique identifier of the user himself.
- A globally unique identifier of the Grid resource or service requested.
- User Accounting coordinates (see below).
- Resource Accounting coordinates (see below).

The purpose of this set is to identify the actors involved in the transaction: the user, the resource, the job and the accounting services responsible for archiving the information.

The metrics collected are not covered in this list since it is not mandatory for an accounting system to collect usage metrics. For example a resource owner may want to charge the user on a Pay Per Access basis, instead of Pay Per Usage. In that case usage metrics are not necesary for charging purpose.

In order for the accounting system to deal with these varied sources of information a “metering abstraction layer” service is needed. Its purpose is to translate the various Usage Records reporting protocols used by the metering sensors into an appropriate Usage Records format that the accounting system can understand. It is infact important to decouple the information providers layer from the accounting service layer. This ensures that the server layer not needs to be able to understand the many different raw metrics collected on the resources. Moreover a well defined abstraction layer enables other developers to implement their own sensors and use them to sen metrics to the accounting service.

8.1.2 ACCOUNTING SERVICE

Once the raw usage records coming from the many different info providers are translated, they can be forwarded to the Grid Accounting system. The accounting system is responsible for archiving these Usage Records and providing querying functionalities. We identify the following basic query functionality:

- Report aggregate or detailed Usage Records for a User.
- Report aggregate or detailed Usage Records for a resource.
- Report aggregate or detailed Usage Records for a group of users (including VOs).

As for every Grid service, it is important for the accounting to be scalable, so its architecture must be distributed. Many local accounting infrastructures are based upon a central database where the Usage Records for each user are kept: this is clearly not feasible in a Grid environment.

To satisfy this requirement, many independent accounting servers are foreseen. Both Grid users and resources have an “account” on one of these servers and all the information concerning their resource usage is kept on that server. There is no limit on the number of the servers. Upon each Grid service request, the user specifies his accounting coordinates (very close in concept to banking coordinates). Each Grid service or resource also specifies its accounting coordinates (these should be published on the resource information provider as well). Then, on completion of a job, or service request, the relevant

usage records are pushed to both the user and the resource accounting servers. Practically we require the user to specify accounting coordinates as a parameter when submitting a job. This parameter then needs to be propagated upon each subsequent Grid service request originating from the job.

Concerning the reliability and fault tollerance of the system a few aspects need to be stressed:

All the communication of the usage metrics to the HLR services are asyncronous. The clients that push information to the servers have a persistent cache of information. If a network or service failure should happen the client retries the communication until it succeeds. This ensures that no Usage Record loss or permanent failures can happen.

DGAS servers are provided with an High Availability Service that continously monitors that the listener of the server is up and running. Thus, in case the service should die, it is automatically restarted in a few seconds.

User jobs alraedy running or scheduled do not encounter any problem due to HLR services being down or unreachable nor any metric loss happens.

If it is requested that WMS checks the user account balance as a mandatory condition for a job to be accepted, then it will be necessary for the HLR to be up and running. Only in this scenario the HLR becomes a key service in the job submission phase, just like NS or LB are. Common databases replication thecniques should be used to have secondary instances of the user HLR servers if needed, thus minimising the risk of long service unavailability.

A practical example of how to partition users over the accounting servers is to have one accounting server per VO (or more for large VOs). In this way the accounting coordinates specified by the user will identify the VO related account that the user intends to charge for a given job.

8.1.3 Cost computation and Billing

The next layer in a general purpose accounting system is the “billing” for resource usage. The infrastructure described so far is absolutely independent from the billing infrastructure and can, in principle, be used stand-alone.

When billing is needed, resources need to be priced, so that a cost can be assigned to the users’ activities.

The price for a Grid resource should be assigned according to its real value, that is, according to the overall performance it offers to the users. The information used to estimate the resource value may change depending upon the resource purpose. The information needed for pricing a computing element is intrinsically different from that of a tape library.

Pricing information may also come from providers in different formats, so another abstraction layer can be introduced to translate such information into a format that the upper pricing layer can understand.

Once the relevant information about the resources is available, a service responsible for assigning the prices uses it, along with a set of predefined pricing policies to calculate the resource prices. It is important for the resources owners to be able to interact with the price calculation by defining the pricing policies. In practice it is the resource owner who decides the algorithm used to compute the price, even if some type of limitations can be imposed at an upper (“Grid” or “VO” management) level.

From an architectural point of view the pricing service has the same requirements of scalability as the accounting service, so it is expected that Grid resources will register themselves with a pricing service and publish the service address.

The last step is the computation of the cost for a user job, or service request, according to the resource price, the amount of resources used (reported by the accounting service as Usage Records) and the cost computation policies.

The cost computation policies are a set of rules imposed by each resource owner that specify how to compute the cost that has to be billed to the user. Usually it will be a formula that computes the job cost

starting from the resource price and the job usage records. As an example the usage of a Grid service can be billed on a per-transaction basis instead of usage metering. In this scenario the service owner will bill a fixed amount of credits for the service usage: this can be represented at the cost computation policy level. Another possible scenario is a resource owner who wants to sell resources at a discounted rate (or offer them free) to a group of users, but wants a standard rate to be applied to the other users. Resources prices should be publicly available in order for users to choose where to submit jobs. Additional pricing policies such as discounts arranged with a particular VO could be confidential. Such policies are specified at resource level.

It has to be stressed that even if the resource is free of charge, or users pay for the usage on a per-transaction basis, it is important for users to preserve the ability to retrieve the Usage Records corresponding to their work in order for them to check, for example with a properly instrumented work request, whether the metering infrastructure is reliable.

Once the amount to be charged is decided, it is communicated to the accounting level where a bank service implements the virtual payment between the user Grid bank service and the resource one. It is the accounting layer itself that manages the economic accounts for users and resources.

A design aspect that needs to be stressed is that the security of the communication involving accounting information is essential. The Usage Records treated by the accounting system can be used by the Grid site managers and service providers to charge the user for their resource usage or, simply, to track abuses. This implies that this information must be trustworthy and it must be impossible for the Grid user to repudiate it. A practical way to achieve this goal is to mutually authenticate every network communication involving accounting information: the user authenticates to both the service provider and the accounting system. The Monitoring System sends the Usage Records to the accounting on behalf of the user, for example using delegated credentials. At the same time the user requires the remote peer to authenticate itself, so that he can be sure that his Usage Records are sent to the desired Accounting service and not to a malicious one. Another key aspect of accounting information is confidentiality. It is clear that the accounting service treats sensitive information, and users (or the law) may require such information not to be public. Communications should therefore be encrypted. Also, for confidentiality reasons, a proper authorization policy must be enforced for access to the accounting database. The Accounting records should be accessible only by the owner or by trusted administrators. A typical ACL would allow users to access their own records but not those of other users, while allowing VO administrators to access the records belonging to users of that VO.

8.2 COMPUTING ELEMENT

The Computing Element (CE) is the service representing a computing resource. Its main functionality is job management (job submission, job control, etc.), but it must also provide other capabilities, such as the provision of information about its characteristics and status. Comparable services from other Grid projects include: the EDG CE, the Alien CE and the Globus GRAM.

The CE, exposing a Web Service interface, may be used by a generic client: an end-user interacting directly with the Computing Element, or the Workload Manager, which submits a given job to an appropriate CE found by a matchmaking process (see Section 8.3).

A CE refers to a set, or *cluster* of computational resources, managed by a LRMS. This cluster can encompass resources that are heterogeneous in their hardware and software configuration. When a CE encompasses heterogeneous resources, it is not sufficient to let the underlying LRMS dispatch jobs to any worker nodes. Instead, when a job has been submitted to a CE, the underlying resource management system must be instructed to submit the job to a resource matching the requirements specified by the user.

The interface with the underlying LRMS must be very well specified (possibly according to existing

standards), to ease the integration of new resource management systems (even by third party entities) as needed. The definition and provision of common interfaces to different resource management systems is still an open issue. In fact only recommendations (such as the Distributed Resource Management Architecture API, DRMAA, currently discussed within the GGF) but not consolidated standards exist now in this area.

8.2.1 JOB MANAGEMENT FUNCTIONALITY

The main functionality that the Computing Element has to provide (as recommended in all requirement specifications, as in [9]), is job management. It must provide facilities:

- To run jobs (which includes also the staging of all the required files).
  Characteristics and requirements of jobs that must be executed are specified relying on a subset of the Job Description Language (JDL) [65], used within the whole Workload Management System.
- To get an assessment of the foreseen "quality of service" for a given job to be submitted. This reports, first of all, if there are resources matching the requirements and available according to the local policies. It then might provide an estimation of the local queue traversal time, that is the time elapsed since the job entered the queue of the LRMS until it starts execution.
- To cancel previously submitted jobs.
- To send signals to jobs.
- To get the status of some specified jobs, or of all the active jobs "belonging" to the user issuing the request.

For job submission, the CE will be able to work in *pull model* (where the job dispatching is initiated by the CE itself), or *push model* (where this is triggered by other services, in particular by the Workload Management Service).

When a job is pushed to a CE, it gets accepted only if there are resources matching the requirements specified by the user, and which are usable according to the local policies set by the local administrator. The jobs gets then dispatched to a worker node matching all these constraints.

In the pull model, instead, when a CE is willing to receive a job (according to policies specified by the local administrator, e.g. when the CE local queue is empty or it is getting empty) it requests a job from a known Workload Management Service. If the CE local queue is assigned to run jobs belonging to the users of a specific VO, the request will be done to a WMS assigned to that VO. This notification request must include the characteristics and the policies applied to the available resources, so that this information can be used by the Workload Management Service to select a suitable job to be executed on the considered resource.

Various scheduling mechanisms can be considered when a CE willing to receive a job for execution, has to refer to multiple Workload Management Services. Possible mechanisms include:

- The CE requests a job from all known Workload Management Services. If two or more Workload Management Services offer a job, only the first one to arrive is accepted by the CE, while the others are refused.

- The CE requests a job from just one Workload Management Service. The CE then gets ready to accept a job from this Workload Management Service. If the contacted Workload Management Service has no job to offer within a certain time frame, another Workload Management Service is notified. Such a mechanism would allow supporting priorities on resource usage: a CE belonging to a certain VO would contact first a Workload Management Service referring to that VO, and only if it does not have jobs to be executed, the Workload Management Services of other VOs are notified, according to policies defined by the owner of the resource.

8.2.2 OTHER FUNCTIONALITY

Besides job management capabilities, a CE must also provide information describing itself.

In the push model this information is published in the information Service, and it is used by the match-making engine (see Section 8.3) which matches available resources to queued jobs. In the pull model the CE information is embedded in the "CE availability" message, which is sent by the CE to a Workload Management Service. The matchmaker then uses this information to find a suitable job for the CE.

The information that each CE should provide will include:

- the characteristics of the CE (e.g. the types and numbers of existing resources, their hardware and software configurations, etc.);
- the status of the CE (e.g. the number of in use and available resources, the number of running and pending jobs, etc.);
- the policies enforced on the CE resources (e.g. the list of users and/or VOs authorized to run jobs on the resources of the CE, etc.).

As described in Section 8.1, each CE is also responsible for the *Grid accounting resource metering*, that is, it must measure user activities on the CE resources, providing resource usage information. This information, after having been properly translated in an appropriate format, is then forwarded to the Grid Accounting System.

8.2.3 INTERNAL CE ARCHITECTURE

Figure 9 represents the architecture of the Computing Element.

Considering a job submission, the *Computing Element Acceptance* (CEA) service, exposing a Web Service interface, represents the entry point for submitting jobs to the resources of the CE.

The CEA includes the functionality of a *Site Gatekeeper*, responsible for the mapping between Grid users and local users, and for checking if the job can be accepted according to the configuration options that could have been set to limit the load caused by job processing: this is implemented by a *AuthZ - Auditing* box, which also takes care of logging the required auditing information.

The job is then forwarded to a *Job Mgmt* box, responsible to deal with the actual submission and execution of the job. After having checked that the considered job can be executed in the CE (that is there are resources matching the constraints specified in the job JDL expression and which can be used according to the local policies), the job must be submitted to the resource management system. This task is encompassed by a specific software module, which "hides" and abstracts the underlying resource management system.

The *CE Monitor* (CEMon) service deals with notifications. It can be customized in particular to:

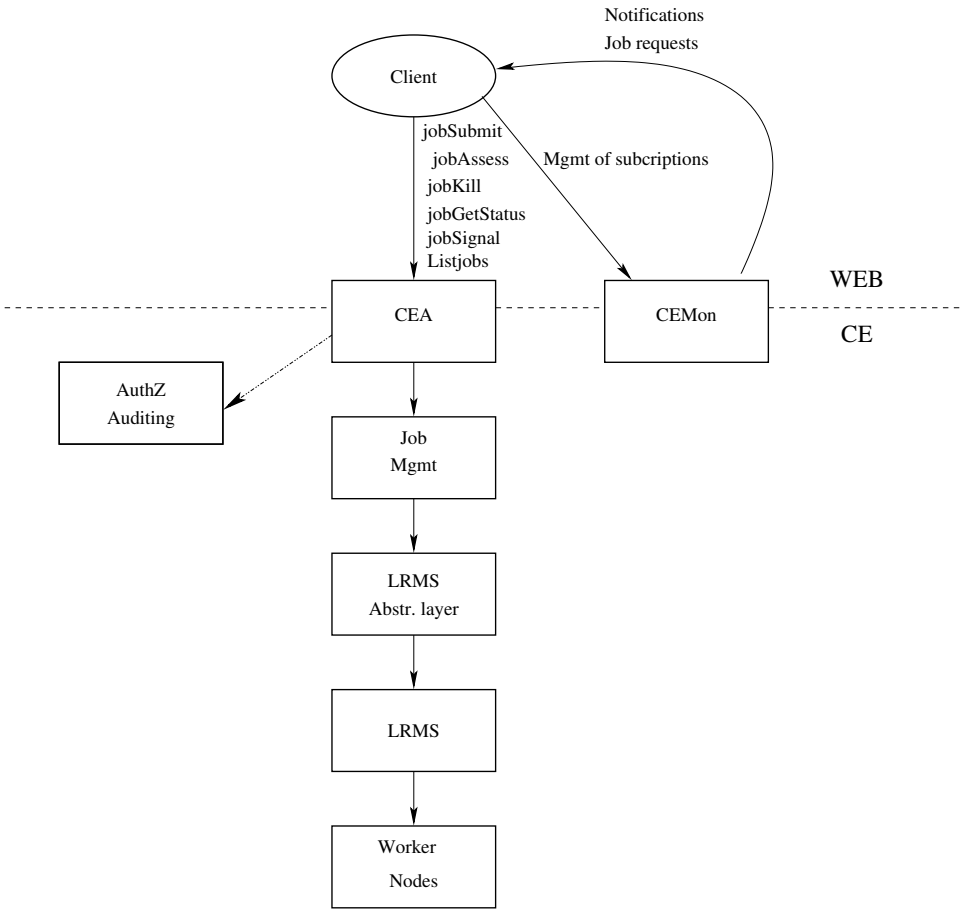

Figure 9: Architecture of the Computing Element.

- asynchronously notify users on job status events, according to policies specified by users (e.g. when a job changes its status, when a job reaches a certain status, etc.).
- notify about the CE characteristics and status. In particular, for a CE working in pull mode, this service is used to request jobs to the Workload Management Service.

This architecture is compliant with the recent discussions (introduced in Section 2) concerning resource virtualization: the *CEMon* and the *Job Mgmt* services in fact can provide virtual environments which can be customized and tailored towards the needs of specific user communities, and the *LRMS Abstraction Layer* virtualizes the actual LRMS. The *CEA*, the *Authorization/Auditing* , and the *LRMS* services, instead, represent and guard the physical resources.

8.2.4 POLICY DEFINITION AND ENFORCEMENT

As already mentioned, the resources belonging to Computing Elements must be used according to some specified policies, set by the administrators of these resources. Examples of policies include the list of users or VOs allowed to use the resources of the CE, the share of the CE resources available to the users belonging to a certain VO, etc.

All the defined policies have to be taken into account when choosing the CE where to submit user jobs, and when selecting the computational resources within a single CE to dispatch jobs to (in case the computational resources within a single CE are managed by different policies).

In order to accomplish this task, the CE will make use of the policy combination and authorization framework described in Section 5.2.2.

8.3 WORKLOAD MANAGEMENT

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 conveniently, efficiently and effectively 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 in areas like data access or interactivity, both with the user and with other tasks. The WMS will broaden its scope accordingly.

8.3.1 FUNCTIONALITY

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 the subject of Section 8.3.5.

For a computation 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 resource 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 the state of the resources, 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 (see Section 8.2.4).

8.3.2 SCHEDULING POLICIES

A WM can adopt different policies to schedule a job. At one extreme, eager scheduling dictates that a job is bound to a resource as soon as possible and, once the decision has been taken, the job is passed to the selected resource for execution, where, very likely, it will end up in a queue. At the other extreme, lazy scheduling foresees that the job is held by the WM until a resource becomes available, at which point that resource is matched against the submitted jobs and the job that fits better is passed to the resource for immediate execution. Intermediate approaches are of course possible.

At match-making level the main difference between the two extremes is that eager scheduling implies matching a job against multiple resources, whereas lazy scheduling implies matching a resource against multiple jobs.

The WM internal architecture will accomodate the contemporary application of the different policies, implemented as easily interchangeable plugins, depending first of all on the requirements and preferences expressed in the job description, but also on the overall state of the Grid, according to appropriate euristics. Such knowledge can only come from proper investigation (including the evaluation of relevant metrics, covering both resource utilization and user satisfaction), with the purpose to understand merits and weaknesses of the different scheduling policies in different scenarios.

8.3.3 THE INFORMATION SUPERMARKET

The mechanism that allows the flexible application of different policies is the decoupling between the collection of information concerning resources and its use. The component that implements this mechanism is named *Information Supermarket* (ISM) and represents one the most notable improvements in the WM as inherited from the EU DataGrid (EDG) project.

The ISM basically consists of a repository of resource information that is readily available in read only mode to the matchmaking engine and whose update is the result of either the arrival of notifications or active polling of resources or some arbitrary combination of the two. Moreover the ISM can be configured so that certain notifications can trigger the matchmaking engine. These functionalities will not only improve the modularity of the software, but will also support the implementation of lazy scheduling policies.

8.3.4 THE TASK QUEUE

The second most notable improvement in the WM internal design is the possibility to keep a submission request for a while if no resources are immediately available that match the job requirements. This technique is used, among others, by the AliEn and Condor systems. Non-matching requests will be retried either periodically (in an eager scheduling approach) or as soon as notifications of available resources appear in the ISM (in a lazy scheduling approach). Alternatively such situations could only lead to an immediate abort of the job for lack of a matching resource.

The component that implements this feature is named *Task Queue* (TQ) and, as for the ISM, provides a necessary mechanism for the support of lazy scheduling policies.

8.3.5 JOB LOGGING AND BOOKKEEPING

The Logging and Bookkeeping service (L&B for short) tracks jobs in terms of *events*—important points of job life, e.g. submission, finding a matching CE, starting execution etc.—gathered from various WMS components as well as CE's (all those have to be instrumented with L&B calls). The events are passed to a physically close component of the L&B infrastructure (aka *locallogger*) in order to avoid any sort

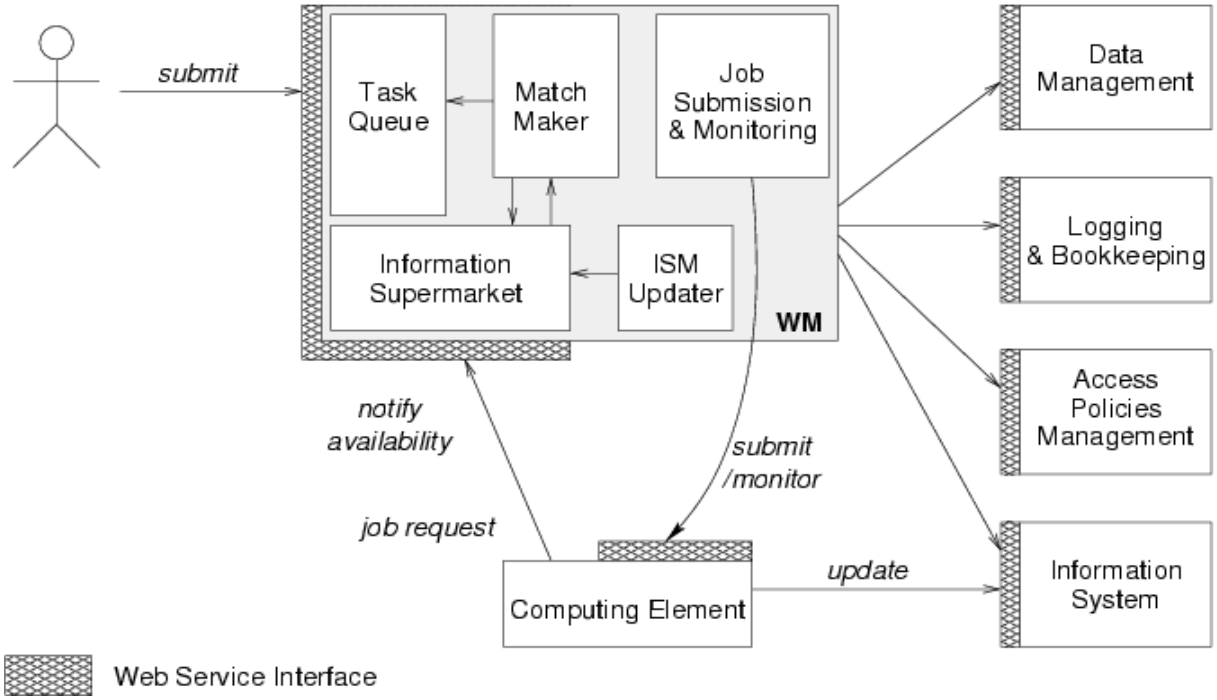

Figure 10: Internal architecture of the Workload Manager.

of network problems. This component stores them in a local disk file and takes over the responsibility to deliver them further.

The destination of an event is one of *bookkeeping servers* (assigned statically to a job upon its submission). The server processes the incoming events to give a higher level view on the job states (e.g. *Submitted*, *Running*, *Done*) which also contain various recorded attributes (e.g. JDL, destination CE name, job exit code, etc.). Retrieval of both job states and raw events is available via legacy (EDG) and WS querying interfaces.

Besides querying for the job state actively, the user may also register for receiving notifications on particular job state changes (e.g. when a job terminates). The notifications are delivered using an appropriate infrastructure.

8.3.6 THE OVERALL ARCHITECTURE

Figure 10 shows the overall architecture of the Workload Manager, together with the interactions with external entities. Among these the most coupled with the WM is the Logging and Bookkeeping Service, which keeps events generated by different components as a job traverses them. Such events contribute to the generation of the status of a job. Other entities are the Information System (see Section 7), used, for example, to fill the Information Supermarket, the Data Management services (see Section 9), assisting the WM when the scheduling involves knowledge concerning location of data on the Grid, the Access Policies infrastructure (see Section 8.2.4).

Both the WM and the other services are expected to offer a Web Service interface.

8.4 JOB PROVENANCE

8.4.1 PURPOSE, EXPECTED USAGE, AND LIMITATIONS

The purpose of the Job Provenance service (JP) is keeping track of the definition of submitted jobs, execution conditions and environment, and important points of the job life cycle for a long period (months to years). Those data can be used for debugging, post-mortem analysis, comparison of job execution within an evolving environment, as well as assisted re-execution of jobs. Only data of completed (either successful or failed) jobs are handled; tracking jobs during their active life is the task of L&B and Job Monitoring services described in Section 8.3.5 and Section 7.2, respectively.

In general, gathered data are stored (i.e. copied) within the JP storage in order to really conserve a partial snapshot of the Grid environment when the job was executed, independently of changes of other Grid services. Obviously there are practical limitations of the extent to which it is feasible to record the entire job execution environment. (In the ideal case this would encompass a snapshot of the entire Grid!) We restrict the recorded data to those that are processed or somehow affect processing of the Workload Management and Computing Element services. On the other hand, snapshots of the state of other Grid services are not done, namely queries to the Data Catalog and their results are not stored, neither are the contents of data files downloaded from and uploaded to Storage Elements—only references to those are recorded if required.

8.4.2 ENCOMPASSED DATA AND THEIR SOURCES

Data required by the Job Provenance service are gathered by various Grid middleware components. Depending on the design of a particular component the data are either “pulled” by the JP from the component, or the component has to be instrumented to “push” the data towards JP.

Certain minimal records of a job (see the job life log below) are always stored in order not to lose the job completely. However, the complete JP data may grow rather large, and it strongly depends on the specific context whether it makes sense to gather the complete data or only a subset. In general, policies at several levels (at least WM and CE) define which data should and could be gathered, i.e. enforce what is mandatory and what is prohibited, allowing the user to specify preferences within those bounds. In general, those policies are specified by “owners” of the resources (WM, CE).

The following data are gathered:

Job life log taken over from the L&B database (see Sect. 8.3.5) after job completion. Among other information useful mainly for debugging and detailed analysis of job execution it contains the complete definition of the job (in terms of the submitted JDL), various timestamps (e.g. when the job was submitted, matched for a particular CE, started and finished execution), information on the chosen CE (or more of them, if the job was resubmitted), various ID’s assigned to the job (Condor, Globus, LRMS, . . .), and the result of execution including the return code.

Another important information available in L&B are the user-defined annotations (“tags”) which can be specified either statically upon job submission, during its execution, or even overridden after the job terminates.

All the information is copied out from L&B and can be purged eventually.

Executable file(s) are provided by the user upon submission. In the case of high-throughput jobs the executable is the same for many (thousands or more) jobs. Therefore for efficient storage of the executable, JP has to allow sharing a stored executable among multiple jobs.

The executable is managed by the submission interface (aka Network Server) of the Workload Management service. This service has to be instrumented to cooperate with the JP storage.

Input/Output sandbox The input sandbox of a job contains miscellaneous files required for execution, and the output sandbox may contain various output files, e.g. debug logs, or even a core file in the case of crash.

By default only the input sandbox is stored—the output is assumed to be reproducible by the job. The user (or a WM/CE policy) may trigger storing the output sandbox as well. For both input and output, either the whole sandbox or enumerated files only may be stored.

The sandbox files to be stored by JP are specified upon job submission. By default, JP should store sandbox files specified as local ones on the user-interface machine (hence transferred to WM sandbox area)—these are assumed to be volatile. Other sandbox file (accessed via http, ftp etc.) may be copied to JP on explicit request (via JDL).

Staging the sandboxes in and out is managed by the Network Server too. Therefore the same instrumentation applies.

Input/output files In contrast to the files contained in the sandboxes these are downloaded from and uploaded to Storage Elements or accessed via glite-IO. JP does not copy those files but it is desirable to record references to them, i.e. GUID and optionally LFN(s) or URL valid at the time of job execution.

Execution environment comprises of operating system and installed software versions, and specific configuration information (including setting of environment variables) of the particular worker node of the CE where the job has been executed.

We assume that the primary source of the most up-to-date information of this type is the worker node itself. Therefore the script which wraps the invocation of the job executable will be instrumented to log the required information. However, certain information can be also retrieved from CE configuration management service.

The potential extent of information in this category is huge, with varying meaningfulness for particular purpose. Therefore a rigid generic approach is inappropriate. Instead we let the user (or policy) choose what should be recorded. JP provides a predefined set of typical data items to gather (e.g. output of “rpm -qa”, enumerated environment variables, . . .), as well as the facility to execute a custom plugin script to collect specific data.

Custom data may be provided by the user upon submission, gathered during job execution (via plugin script called by the job wrapper), or even after the job termination.

The described items form a complete set of data that can be gathered by the JP service. However, depending on conditions and requirements which could not be estimated at the design time, some of those data may be irrelevant for a certain type of jobs. As this can result in wasting considerable storage resources we allow restricting the complete set by setting user preferences and/or applying site policy of a particular instance of the JP service.

8.4.3 SERVICE COMPONENTS

Primary storage servers keep the JP data in the most compact and economic form. The access key to the data is job ID, JP relies on its uniqueness, hence collisions are not resolved but reported as an error. All the data related to a single job form a *Job Record* containing the input and output sandboxes, executable (or a reference to it), and the L&B job dump.

In order to allow access to Job Records without the explicit knowledge of job ID’s, certain basic metadata are managed by the primary storage. The exact set of attributes has to be clarified, we are considering job owner DN, submission time, and virtual organisation.

Primary storage provides public interfaces for data storing, retrieval based on basic metadata, and registration of Index servers for incremental feed (see below).

Index servers provide a limited data mining capability on the JP data. Each index server is configured by its administrator to support a set of *queryable attributes* chosen from various L&B system attributes like job submission, start, and termination time, CE name, exit code (more or less the set supported by indices of the L&B server), arbitrary L&B user tags, as well as non-L&B attributes related to the other JP data (E.g. input/output sandbox file names, environment variable settings. For the sake of coherent processing those are transformed, solely for the purpose of indexing, into L&B-like pseudo events.).

A result of a query is the set of matching job IDs and corresponding URLs (`gridftp://`, `lfn://`, ...) according to which the data from the JP storage can be retrieved.

There are two modes of feeding data into the index server:

- *batch*—the server is populated from scratch with data retrieved from the JP storage, typically jobs submitted in a given time interval.
- *incremental*—whenever data on a new job are stored to the JP storage server, those have to be propagated to the index server as well so that further queries are able to find this job.

In general, the relationship between the index and storage JP servers is many-to-many; several index servers (of different index configuration) can be populated with data from a single storage, the data may even cover different time intervals. On the other hand, a single index server may combine data from multiple JP storage servers.

In contrast to the permanent primary storage the index servers are volatile. They may be created, populated with data, and destroyed independently of the primary storage.

The querying interface will be exposed as a web-service. The index server will be also interfaced to the information infrastructure (R-GMA) as an on-demand producer, thus allowing complex distributed queries to be performed over multiple JP index servers.

Analysis support tools Besides the public WS interface to both the primary storage and index server JP provides tools for managing the index servers, e.g. purging and populating them with data coming from the primary storage, as well as manipulating the set of queryable attributes of a particular index server.

Job resubmission support tools JP data contain enough information to re-create the execution environment of a particular job. However, the user may require varying fidelity of the environment reproduction, e.g. preserving the same environment settings but using an upgraded version of a particular library. Therefore support for fine-tuning the specification of the resubmitted job has to be provided. However, this is foreseen as a specific support in existing user interfaces, using the JP service in turn.

8.4.4 SECURITY

Authentication All the communication involved in JP is authenticated using appropriate user or service credentials.

Encryption A lot of the gathered data may be considered sensitive, hence encryption of data transfers should be considered. However, this may generate a considerable overhead in the case of bulk file transfers, therefore fine grain control is required (e.g. encrypt input/output sandboxes but not executables).

Authorization

- *Storing to primary storage* is done by L&B, WM (sandbox management part) and CE eventually. Only credentials of configured trusted services are permitted.
- *Read access* to both primary storage and index servers. ACLs (including both user identities and VOMS groups) are inherited from L&B upon creation of the Job Record.

  If the primary storage back-end (SE service) allows, its access control mechanisms are exploited as well.
- *Modification of Job Records.* To be defined later if required. Also ACL manipulation will be considered.
- *Feeding data into index server.* Mutual trust between primary storage and index server is required. This is based on both primary storage and index servers' configurations.

8.5 PACKAGE MANAGER

A Package Management (PM) Service is a helper service that automates the process of installing, upgrading, configuring, and removing software packages from a shared area (software cache) on a grid site. This service represents an extension of a traditional package management system to a Grid and it should use one of the established package management systems as a back-end. Some well-known examples of such systems include:

- RPM, Red Hat's package manager, used not only by Red Hat Linux but by several other Linux distributions.
- dpkg/APT (used originally by Debian GNU/Linux, now ported to other systems).
- Portage, used by Gentoo Linux and inspired by the BSD ports system.
- The "ports tree" system used by FreeBSD, NetBSD,OpenBSD and the like.
- Pacman, package manager developed at Boston University and used by several Grid projects (International Virtual Data Toolkit - iVDGL, Grid3)

The software is distributed in packages, usually encapsulated into a single file. The file, as well as the software itself, contains metadata that describes the package's details, including its name, checksums, and dependencies on any other packages that it needs to work. It may also include information on how to configure the package for use and how to remove the package cleanly when it is no longer required. The package manager then uses this information to install, configure, and remove packages as requested by the user.

The PM Service is used by other services running in a resource provider environment (Computing Element and Job Wrapper) in order to construct the list of packages that are required to execute a job specified by the JDL. The information returned is an LFN or SURL. Based on that, the application (CE or JW) can retrieve the required files, install them in the package directory and make them available on the WN.

The PM Service operates in the context of a VO and understands and resolves possible dependencies between the package versions provided by the VO administrator. This service is not responsible for the maintenance and deployment of middleware or system software components.

The VO package administrator maintains the package cache at VO level consisting of a tarball and a metadata file per package. The tar file can contain the precompiled binaries for a given platform or the source files that can be compiled to produce executables. The associated package metadata file contains the build instructions for the source files, possible dependencies on other packages as well as the instructions on how to setup the runtime execution environment.

In a typical scenario, the VO package administrator creates the binary package caches for one or more computing platforms, verifies and possibly digitally signs them. These caches are then published and made available for download via the PM Service. On the execution site, a local instance of the PM Service will, on request from a CE or JW, fetch and install binary packages into the local package cache. This local package cache should reside on a file system managed by the PM Service assuring that unused old packages are removed if disk space is needed to install newer versions.

However, should the VO decide to manage only the source caches, the packages can be built by the local PM Service, using the build instructions contained in package metadata and installed in the local package cache. In addition, the existence of binaries can be advertised, thus minimising download of packages from multiple locations. In this way, the PM Service could maintain the hierarchy of package caches to assure scalability and provide a fail-over capability.

8.5.1 Other Services Used

The PM will use the authentication, authorization, and file catalog components.

8.5.2 Security

Access to VO packages should be controlled and possibly restricted and audited. The easiest way to achieve that is to treat the packages as any other File Catalog entry and to apply common Authentication, Authorization and Auditing mechanisms. The integrity of individual packages should be verified by appropriate checksums. The package metadata information (including checksum information) should be retrieved from a trusted and certified VO site, independently from the package itself.

9 Data Services

In this section we describe the EGEE Data Services. For the EGEE data management service stack we make the assumption that the lowest granularity of the data is on the file level. We deal mostly with files if it comes to application data as opposed to e.g. data sets, data objects or tables in a relational database. The reason for this arguably very restrictive assumption is twofold: Most importantly, the initial two application groups to work with EGEE's gLite implementation are the High Energy Physics and Biomedical communities, for whom data are stored mostly in files. 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. For EGEE to solve a generic problem we felt it safe to start with a semantically well-understood system, hence our choice to start solving the Grid data management problem for files, building on the work of previous projects, most notably the EU DataGrid and AliEn.

The data services can be put into three basic categories: storage, catalogs and movement. Figure 11 gives an overview of the service interfaces and some basic internals. All of the details are given in this section. Note that all Figures displaying service components in this section are color- (and shape-) coded: Components that are managed by the VO are displayed as green circles, services managed by the site are displayed as blue rectangles and service managed by both VO and site are coloured in a greenish blue and their shape is rectangular with rounded edges.

9.1 Data Naming

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

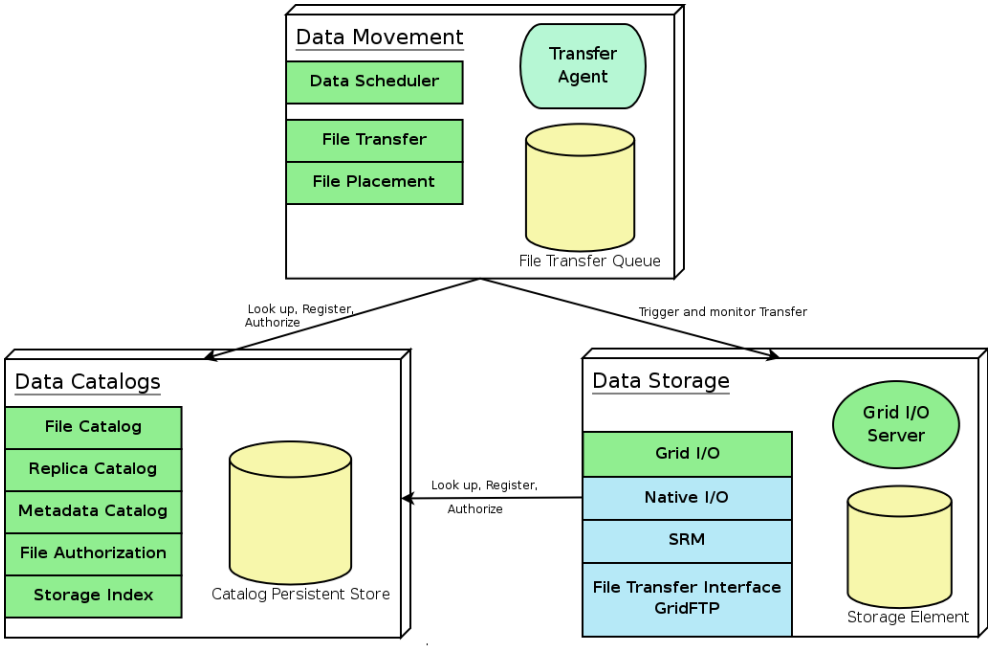

Figure 11: Overview of the EGEE Data Management services.

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 9.3).

In this section we introduce and explain each concept in detail.

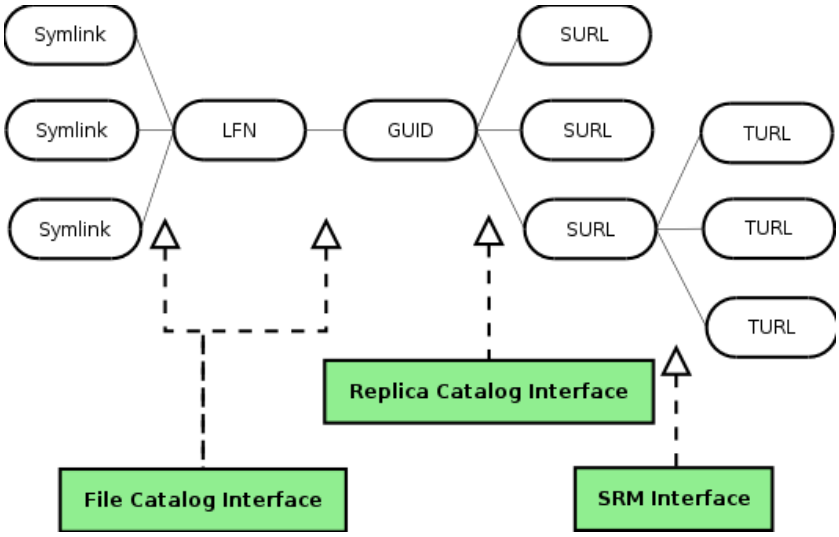

Figure 12: The File Catalog interface manages the 1:1 LFN-GUID mapping and the symbolic links to the LFNs. The Replica Catalog Interface manages the GUID-SURL mapping, and the SRM interface provides the SURL-TURL mappings.

We have the following names identifying data in the Grid (see Figure 12):

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 if everyone agrees to use a convention.

GUID Global Unique Identifier: A logical identifier, which guarantees its uniqueness by construction (based on the UUID mechanism [50]). 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* method).

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 Section 9.2). 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.

9.1.1 Logical File Name

The LFN is the name of a file in the VO-internal logical namespace, organised in a hierarchical directory structure. A valid LFN might be a string like

```
/glite/myVO.org/production/run/07/123456/calibration/cal/cal-table100
```

The LFN has to be unique. In our architecture, the Grid interface that is used to manage the LFNs and their namespace is the File Catalog interface. The gLite Fireman catalog implements this interface. It is responsible for assuring the uniqueness of the LFN.

If the same catalog is to be shared among several Virtual Organisations, then the LFN uniqueness has to be assured across VOs. In order to be able to share the catalog the VOs have to agree on a naming convention for the logical hierarchy. This can be simply achieved by creating two branches in the hierarchical logical namespace for each VO. Then each VO only has to assure LFN uniqueness within its own

branch. The VOs are thus also able to manage their namespaces and administrative domains independently. This kind of logical branching is the same mechanism as is used in other distributed filesystems like AFS where data is always prepended by the AFS cell name [46].

Exposing the VO name so prominently in the LFN hierarchy has as a consequence that the VO names themselves have to be unique, just like AFS cells are. It is therefore probably good practice to have VO names that are actually valid DNS entries, even if the VO does not intend to share its catalogs with other VOs.

9.1.2 DIRECTORIES

We define the concept of a directory in the LFN namespace. The directory may be manipulated just as in a normal file system. It can be listed in the same manner, new entries may be added to it or existing ones renamed and removed. The LFN contains the fully qualified directory hierarchy that can be navigated using the directories. The HEPCAL DataSet concept may be mapped to a directory in the logical namespace. In the previous example LFN, one of the directories is

```
/glite/myVO.org/production/run/07/123456/calibration/
```

A user can also copy logical files from other logical directories into their own directory. **Note:** We do **not** support symbolic links on logical directories, due to the difficulty to traverse symbolic directories across a set of distributed namespaces and catalogs. This is added complexity which we decided not to introduce.

9.1.3 SYMBOLIC LINKS

The File Catalog allows symbolic links to be set to LFNs, their semantics being analogous to those in the Unix file system. Similarly to symbolic links in Unix, LFNs may have a many of symbolic links pointing to it. Symbolic links have to be specified using absolute paths. They have the same semantics as the Unix filesystem symbolic links, i.e. their consistency is weak and they might point to non-existing LFNs. Any operation on the target LFN does nothing to update the link.

As mentioned above, we explicitly do not support symbolic links on logical directories. The reason is that in order to support a distributed logical namespace, it would be very difficult to track directory traversal across many catalogs residing at different locations on the wide area network and to resolve symlink loops. This makes it possible for us to be sure that there are no cycles in the links either. We also restrict symbolic links to point to LFNs only, i.e. not to other symbolic links since there is no reason why a symlink would need to point to another link and not to the LFN directly. This again reduces the complexity of the naming scheme.

9.1.4 GUID

As mentioned earlier, the LFN and GUID are both unique but with the important difference that the LFN is human-readable and mutable, whereas the GUID is neither. Also, the LFNs come with a hierarchical structure whereas the GUIDs expose no structure at all. Some VOs may enforce the policy of immutable LFNs in which case there is no semantic difference except for the logical namespace hierarcy. The use of LFNs is not mandatory, VOs may decide to use only the GUIDs to identify their files.

The internal GUID does not need to be known by the users who will usually only see the human readable LFN. Nevertheless, it is possible that a Grid file has no LFN reference at all, only a GUID. In that case the user has to *know* the GUID when contacting the Grid catalogs to resolve its location.

9.1.5 MOTIVATION

At first glance (looking at Figure 12) all these names and their relation seem overly complex and maybe even unnecessary. The Globus RLS [11] works with a simple N:M mapping model between LFNs and SURLs, why is that not workable for EGEE? What's the reason to have the GUID at all? Or to have the LFNs and Symlinks separated like this? There are several reasons to motivate GUIDs and the fact that there are no hard links, all of which have an influence on the semantics of the operations on the files the Grid end-user sees. These are summarized below.

GUID Motivation The introduction of the GUID is necessary to allow us to distribute the catalog content across the wide area. By keeping the GUID immutable, the distributed catalog can detect accidental duplication of LFNs should they occur. An example where this might occur is when a batch farm producing some output is disconnected from the wide area network and registers a new file (and a new LFN) in its local File Catalog. Upon reconnection, the local File Catalog instance tries to re-sync with the rest of the world, and finds the LFN already registered. The clash can be dealt with by some configurable policy, the easiest of which would be to mail an administrator. The GUID provides the Grid user with a guaranteed unique name that also administrators can use to reference the file while dealing with clash resolution. If the mapping is simply an N:M mapping between LFNs and SURLs, the system would not be able to distinguish between a clash and a regular registration, and we would end up with a situation where different files are registered as replicas to the same set of LFNs. If we do have GUIDs to track clashes, the typical resolution would be to assign a different LFN to the file. In general, the application should take reasonable steps to ensure that the LFN is unique; the process above is only for clash resolution and error recovery purposes.

No Hard link Motivation Having introduced the GUID, one might simply allow more than one LFN to be linked to the same GUID (this was the model in the EU DataGrid). This is like supporting the concept of a hard link on the filesystem, where more than one file is linked to the same inode. However, in a distributed system providing hard links, the delete operation becomes very difficult to manage and its semantics become highly nontrivial. In the local file system the operating system takes care of reference counting; deleting a hard link will not free the inode if it has other hard links pointing to it. In the distributed scenario, every time the delete method is called, the Grid services would need to find all other links to the given file through reverse GUID-LFN lookup over the wide area before allowing the file to be deleted. The user would see long delays, deadlocks etc. In the EU DataGrid project this problem could not be addressed properly.

Therefore we decided to keep the LFN-GUID mapping as a 1:1 mapping and provide symlinks to the LFN. Symbolic link semantics are easy for everyone to understand since they are well-known to the user community, and hard links are rarely used. In addition, if the symlinks were pointing to the GUID and not to a unique LFN, the classical filesystem listing semantics would not be easily supported; the user would see its symlink pointing to a GUID while performing a list operation on a logical directory, thus exposing an *internal* quantity (as if the filesystem `ls` would show the inode number). If there is a unique LFN to which the symlinks can point, the users will see the usual semantics and there is no need to expose the GUIDs.

9.2 STORAGE ELEMENT

Next to computing and networking resources, data storage resources are the basic building blocks of a distributed computing infrastructure. Any kind of data is ultimately stored on a data storage resource. There are many kinds of storage resources today, ranging from a memory stick to a multi-petabyte tape silo. Different storage resources offer different levels of Quality of Service, and have different semantics for data access, both for reading and writing.

In a Grid environment the data and its availability is managed by the Grid middleware. The middleware interfaces to storage need to be rich enough to be able to take into account the possible semantic properties of the underlying storage resource hardware. At the same time its usage should be simple enough so that clients of the storage resource don't need to be experts in the handling of it. The interface to data storage needs to abstract out the commonalities of data stores while allowing the usage of specific features that may be available only on a small subset of storage resources.

Storage is tightly coupled to all other aspects of the Grid architecture. Every data access and data movement needs to take the data resource availabilities into account.

Storage is also tightly coupled to Grid scheduling. Grid Schedulers need to be able to take the data requirements of a job into account. In order to be able to do so, the storage resources need to expose appropriate interfaces as well. If data resources provide means of storage scheduling and reservation, these need to be exposed accordingly as well.

Finally, like any other resource, storage resources also need to be monitored, audited and its usage has to be accountable.

In order to achieve the promise of ubiquitousness, the Grid middleware has to provide the same ease of use as if it was an integrated operating system to the application. The application should be able to access its data independent of the actual location of the CPU it's being executed on. We know that the Grid sites are usually PC farms running a batch system. To set up the same batch system everywhere or to run a world-wide distributed file system everywhere like AFS is not a workable solution for a lightweight middleware solution, i.e. one that is not intrusive. The EGEE middleware has to work with the locally available hardware and software. In order to do so, it needs to interface to every storage system available at a given site. The local storage system may be anything from a file system to a high-capacity mass storage device.

In order to not to have to deal with the peculiarities of each individual storage, we require all Grid-aware storage to implement the Storage Resource Manager (SRM) interface which provides us with most of the functionality we need. We work together with the most important storage providers of our initial two user communities to assure that an SRM interface over their storage solution exists and is well-supported. The SRM interface itself is being standardized through the Global Grid Forum's Grid Storage Management Working Group (GSM-WG) [43].

The set of services that are needed to provide file access and storage makes up what we call a Storage Element (SE):

- Storage back-end with all the associated hardware and drivers
- SRM service implementation on the given storage
- Transfer service for a (set of) transfer protocol(s) but at least GridFTP
- Native file access service exposing a POSIX-like I/O interface
- Auxiliary security, optional logging and accounting services

In essence, the Storage Element is the Grid service responsible for saving/retrieving files to/from some data store which can provide a wide range of quality of service to the user.

The SE provides the following set of capabilities which will all be explained in detail in this section:

Storage space for files. In an SE there is a certain amount of space available to store file-based data.

Storage Resource Management interface. The SE has a standardised interface to manage the underlying storage resource. Currently this is mainly for staging files from and to disk from other media residing in a Hierarchical Storage Management system (HSM).

Storage space management. The SE may provide a means to manage the available space through mechanisms such as quotas, file lifetime management, pinning, space reservation, etc. We do not oblige each SE to provide such functionality, but if the functionality is available the middleware should be able to make use of it. The SRM interface is being extended to provide this capability in a standardised manner. SRM interface version 1 does not provide such capabilities, but interface version 2 does. We expect everyone to provide at least version 1. The EGEE middleware will have to be able to use the capabilities of later versions as well.

POSIX-like I/O access to files. The SE provides an interface that can be used to access the files directly through some supported protocol.

File Transfer requests. Each SE has to support a file transfer protocol. For EGEE we expect every storage resource to support GridFTP [29].

On top of each SE, the middleware provides a Grid I/O and a Grid File Transfer Service (see Figure 13). The end-user application only needs to use the Grid I/O API in order to access its data - the storage resource, the SRM and security services are contacted by this service on the user's behalf. The detailed semantics of file access will be different depending on what kind of storage back-end is being used beneath an SE; there may be substantial latencies for reads and many more failure modes for write. The list of errors and messages is longer than for a conventional file system.

The Grid I/O server component of the EGEE Grid middleware (of which gLite has an implementation: gLite I/O) is making use of the catalog services to authorize access, resolve GUIDs and LFNs to SURLs and to check authorization for the files. The services needed to do this resolution may be deployed next to the SE or at another site. If the deployment is such that these services are *close* to the SE, this will minimise the probability of the files being inaccessible due to catalog unavailability.

9.2.1 Storage Space Types

Storage Elements represent a resource that may have very different quality of service (QoS) characteristics between different instances, depending on the actual storage technology being used. Storage Element middleware running on a tape archiver is considerably more complex than a thin layer of logic on top of a local hard-disk in user-space. In-between these two extremes there is anything from disk-arrays to DVD racks.

The "QoS" in terms of data storage refers to how strong the commitment is on the side of the SE to actually keep the user's data safe. Some SEs may be unable to give any guarantee to the user that the data will be still available the next minute because of the properties of the underlying technology. Other SEs have a Mass Storage System (MSS) backend, with proper backups and tape migration policies, being able to guarantee the storage and availability of the data. Of course the semantics of such systems need to be exposed to the user to some extent. We differentiate between the following SE semantics:

Volatile or Temporary Space. Data stores that provide temporary space semantics make little guarantee for the data in storage. These are also sometimes called opportunistic stores or scratch spaces. Data stored in temporary space may disappear anytime. Such spaces are usually much easier to provide and also to use than permanent storage spaces. The security considerations of volatile space may also be much more relaxed, although the data stored in volatile space may also be well secured through proper authorization and authentication. Obviously, data stored in temporary space should not include master or primary copies of the data or important pieces of data that must not be lost. Such data stores may be used also to provide data caches. The concept of volatile space is not to be confused with data lifetime management; this is a property of the storage space, not of the data. Some stores may however combine the two concepts and provide volatile space based on data lifetime policies.

Permanent Space. Data stores may provide storage space where the data can be kept permanently or if in addition data lifetime management is provided, up to the guaranteed lifetime of the data. Permanent storage does make guarantees when accepting the storage of data: it takes the necessary precautions that the data is not lost for the lifetime of the data. Some storage may provide 'infinite' lifetime semantics, in which case the storage takes care of migration to new technologies, etc. Of course the more guarantees are taken, the more expensive will the management of the data become, so the usage of permanent stores is usually more expensive than the usage of volatile space. For important data and for primary and master copies, permanent storage should be used.

Durable Space. The concept of durable space is introduced to describe a very specific use case: When important, not to be lost data is being produced at a site where no permanent storage is available, the data must not be lost before it is transferred off-site to its final, permanent location. Durable Space provides these semantics. So it will guarantee the safeguarding of a file until it has been moved to another 'safe' place. Of course this guarantee has its limits, but it is better than pure temporary space. Users may have no choice to take the risk and use temporary space, but durable space, if available, is much preferred. The durable concept is similar to the concept of a safe write-ahead cache. Durable space is not as 'cheap' as temporary space since important data that has not been migrated yet must not be lost. Durable data semantics supersede lifetime management semantics: durable data may not be removed even if its lifetime has expired if it has not been migrated to permanent store yet.

The three types of storage space described above need to be taken into account for all operations and their semantics need to be available to all users of data storage resources.

9.2.2 STORAGE RESOURCE MANAGEMENT INTERFACE

The SRM interface that we adopt is described in great detail in the documents available through the GGF Grid Storage Management Working Group (GSM-WG) webpages [43]. We foresee the evolution of the SRM implementations according to these specifications.

This management API is intended to be used mostly by administrators, the job submission system, and as an internal API between the Grid Services and the SE itself, as shown in the picture 13. Of course the users may access the SRM directly if the authorization scheme supports it (details in Section 9.5).

The EGEE SE middleware relies on the SRM to abstract the peculiarities of the underlying SE implementation. gLite does not provide its own SRM implementation or its own MSS backend. The actual middleware implementations for the SE will come from other projects, such as dCache[6], Castor[7], SRB[8], ADS[9], Condor NeST, LCG Disk Pool Manager[10], etc.

9.2.3 SERVICES

We expect that a Storage Element (SE) has at least three interfaces (see Figure 13). The Storage Resource Management (SRM) interface allows the client to manage the storage space - to allocate space for jobs, to prepare data to be retrieved through a certain protocol, etc. The second interface that is expected to be available on a SE is a native POSIX-like file I/O API. And finally, the SE is expected to provide a transfer protocol interface, where we expect at least GridFTP to be supported.

[6] Developed at Deutsches Elektronen Synchrotron DESY
[7] Developed at CERN
[8] The San Diego Supercomputing Center's Storage Resource Broker
[9] Rutherford Appleton Laboratory's Atlas Data Store
[10] Also developed at CERN in the context of the LHC Computing Grid project

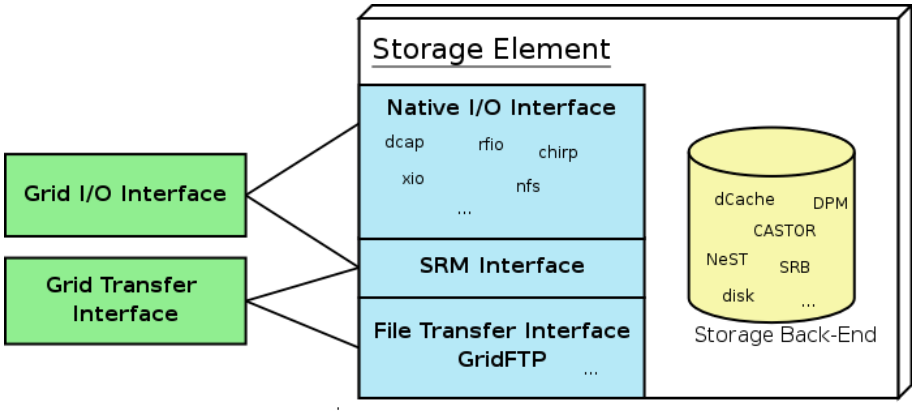

Figure 13: The Storage Element. The SE has three external interfaces: GridFTP, SRM and a native POSIX-like file I/O interface. Depending on the underlying storage system, the SE may support more than one protocol to access the files. The list of examples for storage and I/O in the picture is non-exhaustive. For EGEE we expect each storage to support at least GridFTP as a common transfer protocol, of course also other protocols may be supported. The two interfaces on the left are the Grid interfaces to storage: a Grid I/O and a Grid Transfer interface.

The Grid interfaces will interact with the available storage and protocol components. It exposes a uniform interface to the Grid clients and abstracts away local peculiarities (like the locally available native I/O protocol). The two Grid interfaces are a Grid I/O and a Grid File Transfer interface. In EGEE these are provided by the gLite I/O (see Section 9.2.4) and the File Transfer Service components (see Section 9.4).

9.2.4 GRID FILE I/O

The interaction with the storage element should be transparent to the user through a POSIX-like file I/O API. The EGEE implementation, gLite I/O, provides a minimal subset of truly POSIX-compliant methods as well as a set of custom methods which may be better suitable for some operations in the Grid.

Figure 14 shows how the I/O works in detail. The I/O client library accepts either LFN or GUID as an input to the API (see section 9.3 for an explanation of what the catalogs contain and what GUIDs and LFNs are). The LFN or GUID is presented to the I/O server. Then the actual operation is authorized through the File Authorization Service, i.e. whether the user is allowed to access the file in the given way. Resolving the GUID or LFN into the SURL is the next step, which is then used to access the file through the local SRM. Since the SRM is only a management interface and not an actual protocol, there is a step when the I/O server invokes the native I/O protocol of the underlying storage. (The gLite implementation makes it possible to combine steps 2 and 3 since the FAS and Catalog interfaces are implemented by the same service.) After having received requests through the SRM or native I/O, the SE checks with the Local Authentication and Mapping services whether the given user is allowed access at the local site at all (see section 9.5).

The service implementing the FAS and Catalog interfaces may be co-located with the I/O server and the SE on the same site in order to assure locality of reference (this being the main reason to provide distributed catalogs, see 9.3.2. Of course this is a deployment choice as the catalog may be on a remote site as well. However, if the FAS is on a remote site which is not accessible, none of the local files can be read due to authorization failure. So this situation should be avoided if possible.

There is a lot of room for customisation since all components should be modular on the Grid I/O server

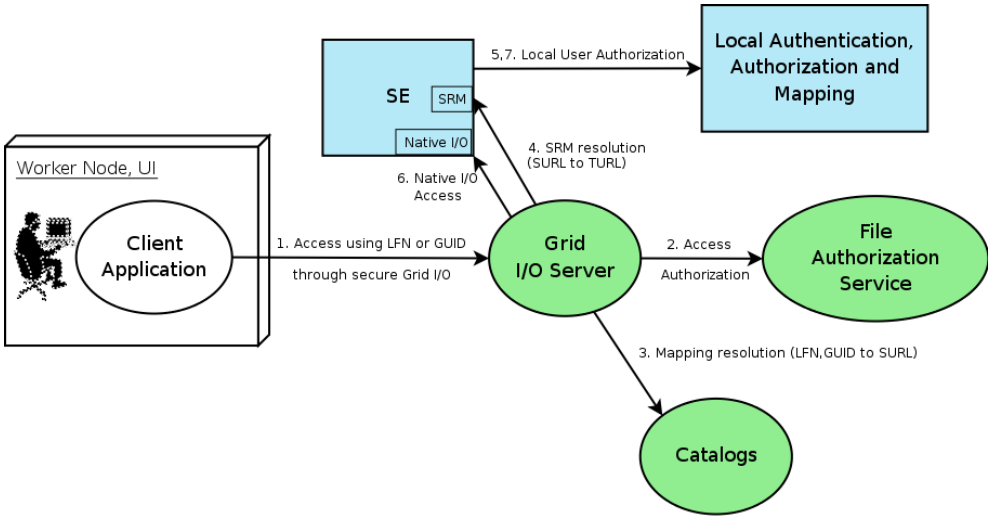

Figure 14: The Grid I/O server interactions. The client's POSIX I/O request is received in step 1. Step 2 is the file access authorization (Grid ACL enforcement) and step 3 returns the actual SURL. Steps 4 and 6 are the SE interactions through the SRM and native I/O protocols. The SE makes sure the client is authenticated and authorized locally in steps 5 and 7.

side following the SOA paradigm. The EGEE implementation allows pluggable components, i.e. in order to support another SRM with another native protocol, it is enough to provide a plugin to the gLite I/O server.

File Creation Creating a new file in the SE is mostly done using the Data Scheduler and File Placement Service (see Section 9.4), i.e. an existing file is copied into an SE and registered in the catalog (or if the file was already on the SE, only the registration and security domain update are necessary).

If a new file is created through the gLite I/O `open()` command, it has to be declared what kind of file the system is supposed to create (see discussion about file access patterns below). A new GUID will be allocated by the system (in order to avoid inconsistencies). Associated file metadata may have to be added as well. Such a creation is not as efficient as the first method. However, it is also possible to create the file locally outside of the Grid efficiently, and to move it into the Grid through the gLite I/O CLI. There is a `glite-put` and `glite-get` utility available in order to do so; this way gLite also supports applications that cannot simply link with the gLite I/O libraries.

File Access Patterns We differentiate between several different file access patterns:

- **Write once, read many.** These files are read-only, they may not be written, so write methods will not work. They can be replicated safely.
- **Rare append only updates, only one owner.** These files are updateable for writes (appends), and we do not expect concurrency issues, so also such files are replicateable, although detection of conflicts has to be built into the system.
- **Frequently updated, one source.** These files are like parameter files, that may change often, but change only at a single location from where all other replicas are updated. The replicas may be checking whether there is a new version available and pull in the latest version if needed.

- **Frequent updates, many users, many sites.** Such a pattern is too difficult to be managed in general by the grid and is not supported. It is also a rare pattern according to the user requirements so far. Databases provide a much better technology for this usage pattern anyway, so we suggest to store such data in databases, not in files. Even so, multi-master distributed database technology is expensive and error-prone and should be used only if it cannot be avoided.

The I/O methods will exhibit different semantics for each supported pattern. For example the write() method will return an error for write-once read many files.

9.2.5 GRID FILE TRANSFER

The second Grid interface to storage is a File Transfer interface (see Figure 13). The transfer services and their relation to storage are explained in section 9.4.

9.2.6 OTHER SERVICES USED

As mentioned above, the Storage Element is making use of the catalog services to resolve GUIDs and LFNs to SURLs and to check authorization for the files. The services needed to do this resolution should be deployed next to the SE. This minimises the probability of the SE files being inaccessible due to unavailability of the catalog.

The SE also makes use of the monitoring services to publish its state information, and of the security services for the purpose of authorization and accounting.

The client of the Grid I/O (on the left-hand side of Figure 14) will have to have a Grid certificate, signed by his VOMS server. In addition, the client will have to find the proper I/O server to talk to through the Service Discovery services.

9.2.7 SECURITY

Security throughout the data management subsystem is explained in detail in section 9.5.

9.3 CATALOGS

In the EGEE architecture, the data catalogs store information about the data and metadata that is being operated on in the Grid. The Grid Catalogs are used to manage the Grid file namespaces and the location of the files (see also Section 9.1 and Figure 12), to store and retrieve metadata and to keep data authorization information.

We decompose the catalogs into catalog feature sets which are represented by catalog interfaces (see Figure 15). These interfaces expose a well-defined set of operations to the client:

Authorization Base The basic authorization interface offers methods to set and get permissions on one or more catalog entries. The permissions are represented as a set of ACLs. The ACLs that the gLite implementation provides are read, write, execute, remove, list, set permission, set metadata and get metadata.

Metadata Base The methods of the base metadata interface deal with setting, querying, listing and removing metadata attributes to one or more catalog entries.

Metadata Schema The schema interface allows the definition attributes and group them into schemas. Attributes may be added and removed from existing schemata. These attributes are then available to be filled with values through the Metadata Base interface.

Replica Catalog The Replica Catalog exposes operations concerning the replication aspect of the grid files. Operations are for example to list, add and remove replicas to a file identified by its GUID.

File Catalog The File Catalog allows for operations on the logical file namespace that it manages. The File Catalog operations are for example making directories, renaming logical file entries and managing symbolic links.

File Authorization The methods needed to implement a standalone authorization service are in this interface, which are basically the ability to add and remove permission entries.

Metadata Catalog In order to implement a standalone metadata catalog, methods are needed to add and remove entries.

Combined Catalog It usually makes sense to provide catalogs that implement the File and Replica interfaces. The combined catalog interface defines a set of convenience methods which in principle could be implemented on the client side by calling the File and Replica Catalog interfaces in sequence. If this interface is implemented by a catalog service, it either also implements File, Replica and Base Authorization interfaces or calls upon an external catalog in order to do so. If the implementation is not in the same place, the Combined Catalog implementation needs to maintain a persistent state of all operations it performs across catalogs in order to make sure that the operations only occur in a synchronised manner and that the method semantics are preserved.

StorageIndex The Storage Index interface is tightly coupled to the File and Replica catalog functionality. It offers methods to return the list of Storage Elements where a given file (identified by its LFN or GUID) has a stored replica.

All catalog interfaces expose bulk operations as part of the interface. They increase performance and optimise interaction with the Grid services.

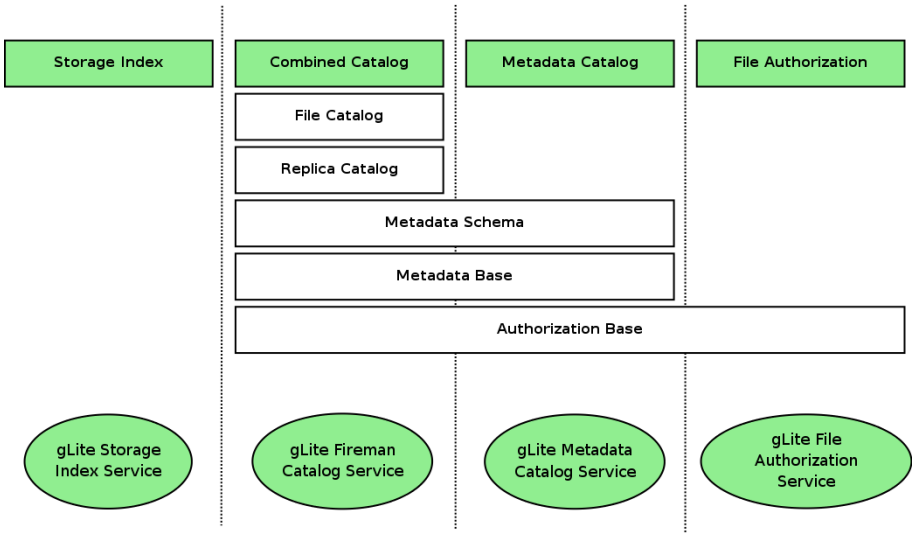

Figure 15: Catalog Interfaces with the gLite services implementing them.

The reason for this interface decomposition is to have service interfaces with well-defined semantics which may be implemented by many parties. In this model, a possible scenario is that more than one interface is implemented by the same service. In Figure 15 the interfaces are grouped such as to show which gLite service provides what implementation. These are

gLite Fireman The gLite FIle and REplica MANager called the Fireman catalog implements all file management interfaces.

gLite StorageIndex The Storage Index service is provided also by the Fireman implementation, but is offered as a separate porttype.

gLite Metadata The standalone gLite Metadata Catalog implementation offers a full metadata catalog solution.

gLite FAS The File Authorization Service can be deployed as a simple authorization enforcement service for file access.

Of course another implementation may choose to implement the interfaces in a different manner, to even further extend the possibilities for deployment of the services. In the following we discuss some details of the catalog interfaces and semantics.

9.3.1 METADATA

All metadata is application specific. Therefore a generic middleware layer can only provide generic, non-specialized metadata capabilities which can only be used optimally to a limited extent.

In EGEE we also provide an implementation of the metadata interface through the Fireman catalog in addition to a standalone metadata catalog implementation. In the Fireman catalog we consider metadata only in the context of file-based metadata, i.e. metadata that is related to the files stored in the Grid, with the LFN or the GUID as the key binding the metadata and the File and Replica Catalog contexts together. File-based metadata has therefore more specific semantics than generic application metadata.

In the Fireman catalog we restrict the set of attributes, i.e. the **schema** of the metadata to be the same for the logical directories, i.e. all files within a directory have to have the same set of attributes. Of course many directories may have an identical metadata schema, in which case metadata searches may be performed over many directories (see Figure 16).

Actually the Fireman metadata catalog can be also interpreted as a metadata catalog where the entries have a hierarchical structure.

The standalone metadata catalog has no such concept as a hierarchical namespace for the items that it associates metadata catalog with, so for any arbitrary existing metadata catalog it should be possible to interface it to the Grid by implementing the gLite metadata catalog interface.

Generic metadata service interface can be offered through a generic Grid Database service that makes data accessible through the Grid. Such services have been tested in previous projects, like project Spitfire as part of the EU DataGrid project and the OGSA-DAI project of UK e-Science.

9.3.2 SCALABILITY AND CONSISTENCY

The File Catalogs that have been deployed to date are all deployed centrally and therefore are a single point of failure. The central catalog model has obviously excellent consistency properties (concurrent writes are always managed at the same place) but it does not scale to many dozens of sites. There are three possibilities to solve this issue:

- Database Partitioning. The data in the database is kept at different sites. For some applications where datasets are unlikely to move, partitioning on data location (and mapping this implicitly into the logical namespace) may be a good approach, solving the scalability problem.

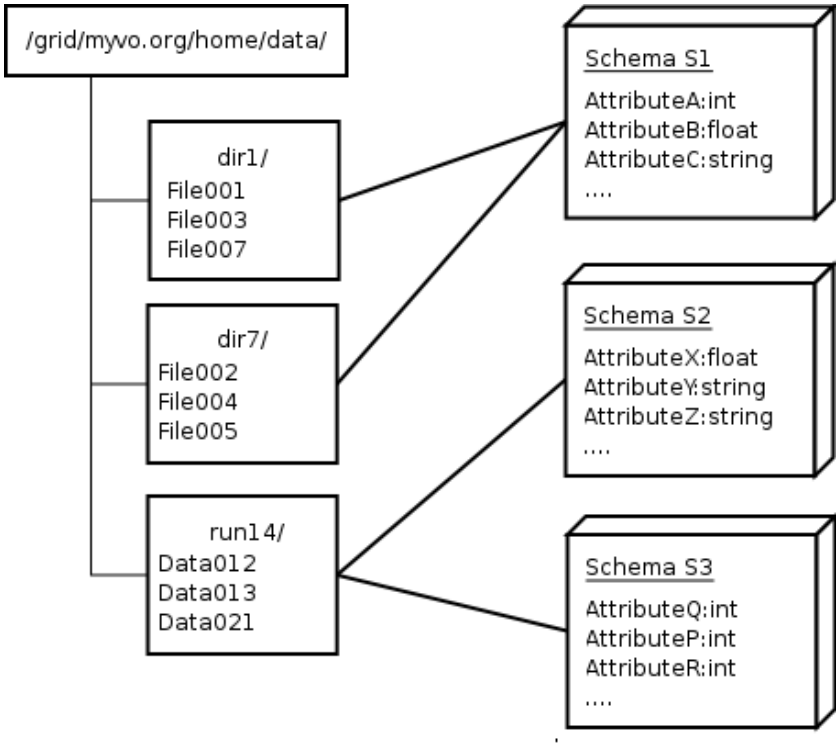

Figure 16: In the file hierarchy, each file in a directory shares the same metadata schema. In the picture, `/grid/MyVO/home/data/set1` and `/grid/MyVO/home/data/set2` both have the schema S1, ie. all files in these two directories may have AttributeA, AttributeB and AttributeC set. Queries on metadata may be scoped on either the schema or the directory, returning files from either only a set of directories or for all files sharing the same schema. A directory may have more than one schema table.

- Database Replication. The underlying database is replicated using native database replication techniques. This may mean a lock-in to a vendor-specific solution. Currently commercial database vendors like Oracle provide multi-master database replication options which may be exploited for such a purpose.
- Lazy database synchronisation exploiting the specific semantics of the catalogs using messaging to propagate the updates. Reliable messaging technologies are available commercially (just like replication for database technologies) and there are some solutions also in the open source domain. The File and Replica Catalog semantics are rather simple and very specific for catalog write operations, so that every time a local write operation occurs, it can be distributed through a message queue to all known and available remote catalogs.

The EGEE middleware will be able to accommodate all solutions by design. Consistency might be broken in the second and third model i.e. it is possible to register the same LFN in two remote catalogs at the same time such that a conflict will occur. The reconciliation techniques apply in both cases. In the third case we can be specific to the semantics of the system and exploit the uniqueness of the GUIDs to detect consistency problems and to notify the users asynchronously.

9.3.3 Bulk Operations

The whole data management design is optimised for bulk operations as opposed to single-shot operations. The reason is that in the stateless web service model in the SOA of gLite, there is a considerable

performance hit when the connection between client and server is established and the security context is built. If this connection has to be rebuilt for each operation, the client-server interaction will be very inefficient.

For this reason, all catalog operations have bulk operations as part of the interface wherever it is reasonable. Using them to bundle similar tasks as a single operation increases performance considerably, optimising the interaction with the Grid services. So whenever possible, the bulk interface should be used instead of sending single requests to the catalogs.

9.3.4 OTHER SERVICES USED

The catalogs rely on being passed in VOMS proxies to be able to perform the full fine-grained authorization based on ACLs.

9.3.5 SECURITY

By imposing ACLs on the filesystem the security semantics are straightforward. This should also help in avoiding concurrency issues when writing into the catalog since each user will have only limited access rights in the LFN namespace and there should be only a finite set of administrators per VO who have full access rights for all of their LFN namespace. The probability of two users with the same access rights to write into the catalog in the same directory in a distributed system is therefore low.

The Replica Catalog interface exposes the operations on the file ACLs. There are two possibilities of how ACLs may be implemented.

- **POSIX-like ACL** The POSIX semantics follow the Unix filesystem semantics. In order to check whether the user is eligible to perform the requested operation, all of the parent ACLs need also be parsed.
- **NTFS-like ACL** The Microsoft Windows semantics are simpler, i.e. the ACLs are stored with the file and the branch has no effect on the ACL. These are “leaf” ACLs, only operating on the file itself.

In a distributed environment, the NTFS-like semantics are simpler to track and are probably more efficient. The Authorization Base interface exposes all operations that deal with querying and setting of the file ACLs. It acts as the authorization authority for file access and is called by other services such as the File Placement Service to enforce ACL security. See Section 9.5 for more details.

9.3.6 ADDITIONAL CONCEPTS

The Master Replica Currently we do not expect the files to be updated. In a distributed system to keep track of all replicas in a consistent manner is a nontrivial task that the middleware should not need to deal with from the start. However, a placeholder is needed to enable such functionality. The master replica flag for a SURL as present in the File Catalog may be used to flag a SURL as the only replica where update operations are allowed. This may then also be the only source for replications. If the master replica is lost, it might be recovered from other replicas or not, based on VO policies. A master replica should always be kept on a reliable SE providing high QoS (permanent space semantics).

Datasets The functionality of having a dataset as described in the high energy physics requirement document [9] is not explicitly supported by this set of services since the catalogs described here are purely file-based. However, for applications having files as the data granularity most of the functionality

is available by using the logical namespace mechanisms for directory and virtual directory, covering a large fraction of the Use Cases. The added metadata support for the file hierarchy may also be used to provide the semantics needed by the applications. These are the possible mechanisms:

- A directory is an explicit dataset by inclusion. Files may be added, removed by the usual mechanisms.
- Datasets by reference may be a directory containing symbolic links to other files.
- A directory may have "real" files as well as symbolic links.
- A virtual directory is a dataset which is created from a metadata query (see below).

Datasets that are more complex, containing object references and metadata as well as references to files are usually application specific and we expect them to be managed by a higher-level service or through the metadata associated with the directories. VOs may provide a specialised metadata service to manage complex datasets and nontrivial operations on datasets or they may use the metadata capabilities of the EGEE implementation. VO dataset tools and dataset catalogs may also make use of the EGEE catalog interfaces internally if they choose to do so, hiding the complexity from the user.

Virtual Directories A virtual directory is a special directory created from a metadata query. A user defines a metadata query whose output is a set of logical files that are then made accessible through the virtual directory as symbolic links. The virtual directory contains a list of files, all of which are symbolic links to the files returned by the user's query. Only a limited set of operations are available in such directories. The result of a query may be stored back in the File Catalog, so whenever the user issues the list command, the same stored results are retrieved. In order to refresh the contents of the directory, a virtual directory refresh would have to be issued explicitly. The advantage is that the user has to explicitly invoke remote catalog calls, they are not invoked "accidentally". Also, since the actual data may change, the user has the possibility to see how a query evolves over time.

Of course only files sharing the same metadata schemas can be used to create virtual directories, since otherwise there is no way to formulate the metadata query.

9.4 DATA MOVEMENT

The data movement services provide scalable and robust managed data transfer between Grid sites, to and from Grid storage. Files are scheduled to be moved between sites reliably. Scalability, manageability and extensibility have driven the service decomposition described in this section (see also Section 4).

The logical service decomposition for data services is shown in Figure 17. The services involved in data movement are the following:

Data Scheduler From the VO's point of view the Data Scheduler is a single central service. It may actually be distributed and there may be several of them, but that depends on the implementation. It accepts high-level transfer requests, where the user does not specify the source replica, or even the destination of any given transfer.

File Transfer/Placement Service The transfer and placement services may get transfer requests from the user directly or through the Data Scheduler. The difference between a transfer and placement service is only their connection to the catalog – the transfer service does not provide LFN and GUID resolution into SURL/TURLs. The deployment of FTS and FPS services depends on what the needs of the VO are. It is possible to have a service at each site.

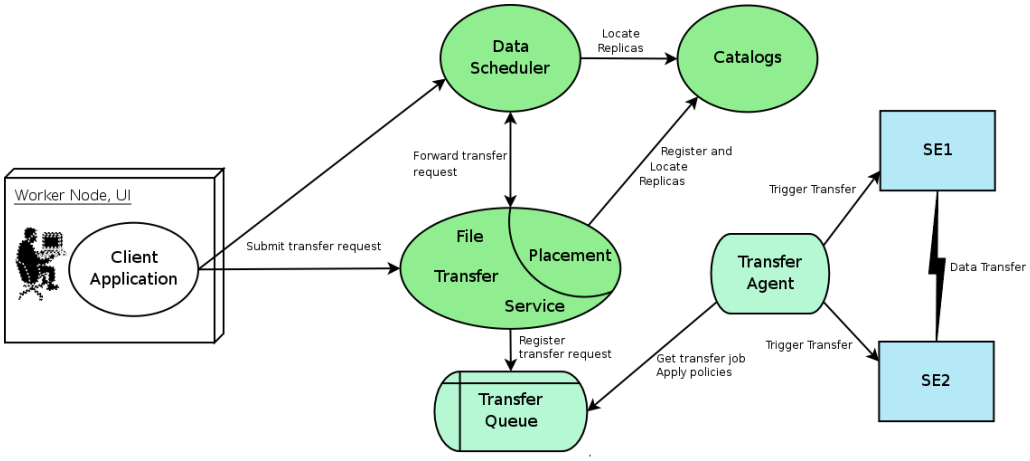

Figure 17: Architecture overview of the Data Movement service components. The Data Scheduler is a high-level component. There may be many File Placement Services around, acting as interfaces to put requests into the Transfer Queues.

Transfer Queue The transfer queue is persistent and is not tied to a site, but rather to a (set of) connections, wide area network channels (links). It keeps track of all ongoing transfers and may also be used to keep a transfer history, for logging and accounting purposes.

Transfer Agent The transfer agent again is tied to both the network channels and the VOs (see details below). The Agent may not only simply get the next transfer job out of the queue, but may also reorder the queue according to site and VO policies.

In the following, we describe each service in more detail and explain their control flow, look at their components and discuss actual use cases.

9.4.1 DATA SCHEDULER

The Data Scheduler (DS) is the high-level service that keeps track of most ongoing WAN transfers inside a VO. The DS schedules data movement based on user requests. At a later stage, the DS might be extended to accept more dynamic kinds of requests by virtue of a data job description language, enabling it to do data job scheduling coordinated together with the WMS.

Users and the job scheduling system will contact the DS in order to move data in a scalable, coordinated and controlled fashion between two SEs. The DS has the following components:

VO Policies Each VO can apply policies with respect to data scheduling. These may include priority information, preferred sites, recovery modes and security considerations. There may be also be a global policy which the VO, by its policy, may choose to apply. Global policy is usually fetched from some configurable place. The same global policy may apply for many VOs.

Data Scheduler Interface The actual service interface that the clients connect to, exposing the service logic. All operations that the clients can use are exposed through this service.

Task Queue The queue holding the list of transfers to be done. The queue is persistent and holds the complete state of the DS, so that operations can be recovered if the DS is shut down and restarted. This queue is the heart of the DS, all services and service processes operate on this persistent queue.

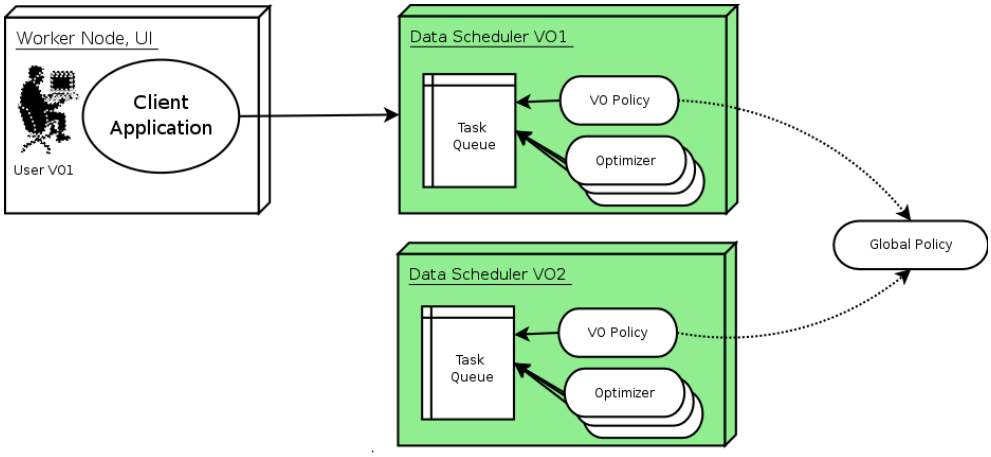

Figure 18: The Data Scheduler

Optimisers A set of modules to fill in missing information (like the source to copy from), or modifying the details of a request based on some algorithm (such as the protocol or the source replica to be used).

The DS is shown in Figure 18 as a single instance. However, there may be several Data Schedulers for the same VO deployed in the Grid. The DS may also be contacted by the File Placement Service, not only from the user directly (see Figure 17).

9.4.2 File Transfer and Placement Service

There is no difference between the FTS and FPS service in terms of functionality, except that the FPS accepts LFNs and GUIDs in its interface and resolves them through the Grid Catalogs whereas the FTS does not. The FTS and FPS receive requests either through their Web Service interface directly from the clients or indirectly through the Data Scheduler. If the request cannot be managed by the local service, it may be forwarded to a known Data Scheduler for further processing.

Once a request is accepted, it is simply put into the associated File Transfer Queue. If the request came through the FPS interface, the LFNs and GUIDs are first resolved through the catalog, and only the resolved names (SURLs) are put into the persistent queue. However, it is made sure that FPS requests also are updating the proper replica catalogs after successful transfer.

The transfers managed by the FTS/FPS are all asynchronous, i.e. the client submits a transfer 'job', which may contain a list of files (with proper source and destination qualifiers) to be transferred. The FTS/FPS assigns a unique string identifier to the job if it is accepted. The states of the whole job and the states of the individual files (that can be tracked using this ID) are not the same: See Figure 19 for the job states and Figure 20 for the individual file states. The reason for the difference is that there may be many files to be transferred in a single job.

The clients see only the FTS/FPS Web Service interface and its associated API and convenience command line tools. In gLite we also provide a web browser interface to the FPS/FTS.

9.4.3 File Transfer Queue

The transfer queue is just a table, usually kept in a relational database. However, any other persistency mechanism is suitable. The queue in itself does not provide an interface or a service. In the gLite

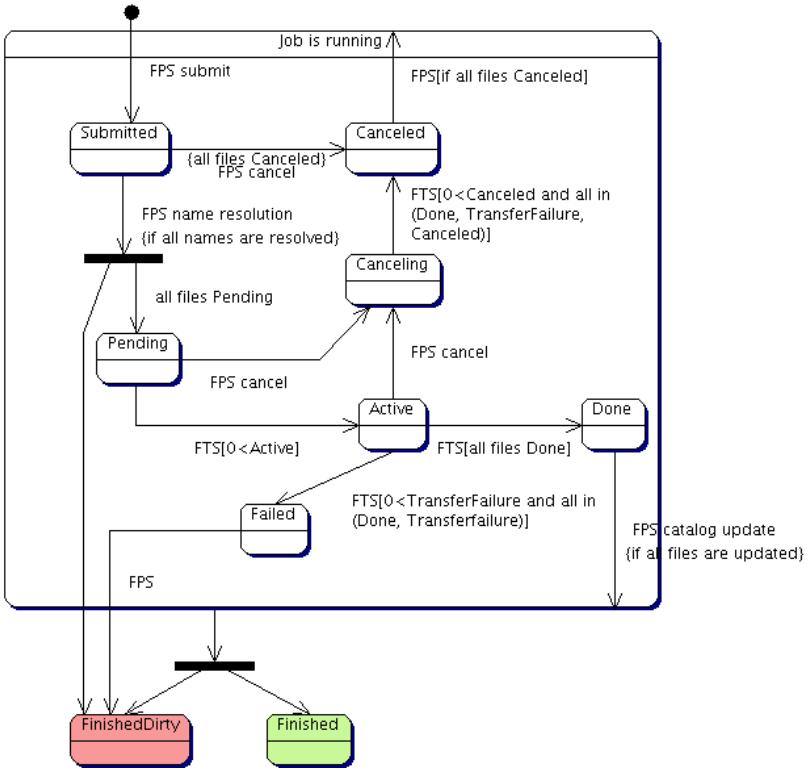

Figure 19: The states that are possible for a file transfer job. A job may have many files.

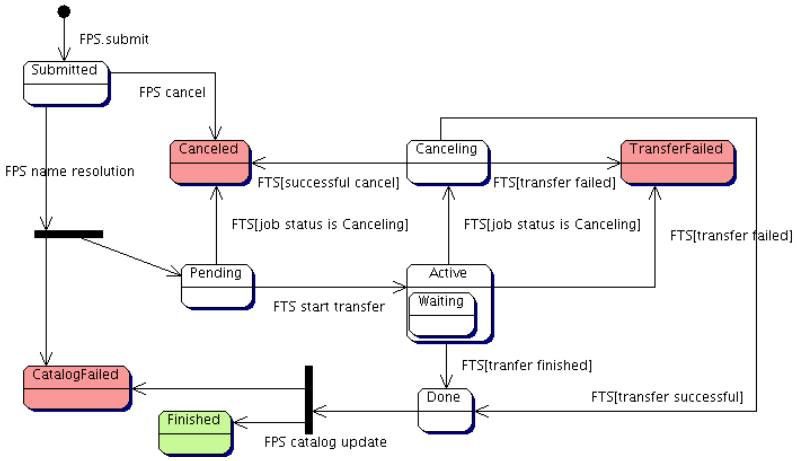

Figure 20: The states that are possible for a file that is being transferred.

implementation it is simply a set of tables in a relational database. The queue is being manipulated directly by the Transfer Agents and the FPS/FTS.

9.4.4 FILE TRANSFER AGENT

The FTA is a very modular component built as a container for many *Actors* that can be executed periodically. Each Actor will act on the Transfer Queue in a very well-defined manner.

The Transfer Queue provides different views for the FTA agents to operate on. There are two kinds of views: channel and VO views. The channel view will expose all transfers relevant for a given channel, the VO view will show all transfers associated with a certain VO.

Standard FTA Actors will resolve names, apply VO policy, channel policy, monitor the state, trigger and monitor the actual file transfer and change the state in the queue if the state of the transfer has changed. For the discussion below, it is important to define at least two Actors: The *Channel Allocator Actor* will actually assign a file to be transferred through a certain channel. The *Transfer Trigger Actor* will actually work its way through triggering the transfers based on the parameters set for the channel configuration (number of parallel transfers, etc).

9.4.5 TRANSFER CHANNELS

The Data Movement services need to interoperate and manage two basic resources: Storage and networking. For the network side, we use the concept of network channels (links) that may be dedicated or shared resources. Each channel has its own queue in a File Transfer Queue, which is managed by at least one File Transfer Agent.

The following channel states are defined (explained by referring to the two FTA Actors defined above):

Active The Allocator is looking for work to assign to a channel *and* the Trigger is looking for work in that channel to be put on the wire.

Drain The Allocator will *not* add anything new to a channel *but* the Trigger will continue to serve jobs that have been assigned to its channel. The effect is to drain all pending transfers from the channel.

Inactive The Allocator will assign work to a channel *but* the Trigger will *not* put any more jobs on the wire. This is used by a sysadmin to empty the network. Note that jobs currently active on the wire will complete.

Stopped Neither the Trigger or Allocator will do any work. Nothing will be assigned to the channel and no work will be put on the wire. Existing jobs on the wire will complete.

Halted A serious error has been detected on the channel (e.g. there have been a certain number of 'sequential' failures on the channel in the last few minutes) and that the channel has been automatically stopped by some monitoring process. When a channel is halted an alarm should be raised to alert an operator for manual intervention. This state is designed to prevent the transfer queue draining in case of problems.

9.4.6 ADDITIONAL HIGHER LEVEL SERVICES

Having the basic set of data management services in place, it is straightforward to put additional convenience services in place, usually in the application layer. An example is a component that automatically schedules data transfers based on some trigger or event. The event may be application specific or it may be linked through the catalogs (new entries) or from the SE (new data) or other monitoring services. The auto-scheduling service would place new requests into either the local FPS or the global DS periodically whenever the event is triggered. It may use the WS interface to achieve the scheduling task and can focus on implementing its triggering mechanism. It is also possible to extend the FTS/FPS and the File Transfer Agent by implementing custom actions directly into the FTA, which are executed periodically and may be used to enforce some special policy (reordering the queue) or to add additional tasks (register data and metadata in application specific catalogs), etc.

9.5 SECURITY IN DATA MANAGEMENT

As described in the executive summary Section 2, it is important to distinguish services owned and managed by the VO from those owned and managed by the site administrators (see Table 1). Throughout this Section, all Figures (11 - 18) colour-code the components that are managed by the VO (in green circles) and by the site (in blue rectangles). Some components of the Data Movement services are managed by both, these are coloured in a greenish blue and their shape is rectangular with rounded edges (e.g. in Figure 17).

The importance of this taxonomy is two-fold:

- It allows the assignment of clear responsibilities for every Grid service, which is important for accounting, auditing and liability, which are all important aspects of the security management and design.
- It allows us to identify service locality, since all site-managed services are running at a well-defined site. VO-managed services are usually 'free-roaming' services, i.e. the VO is free to choose on which site to deploy them, and vice versa each site, knowing what service is VO managed and what service is site-managed, is free to choose which VO service to deploy.

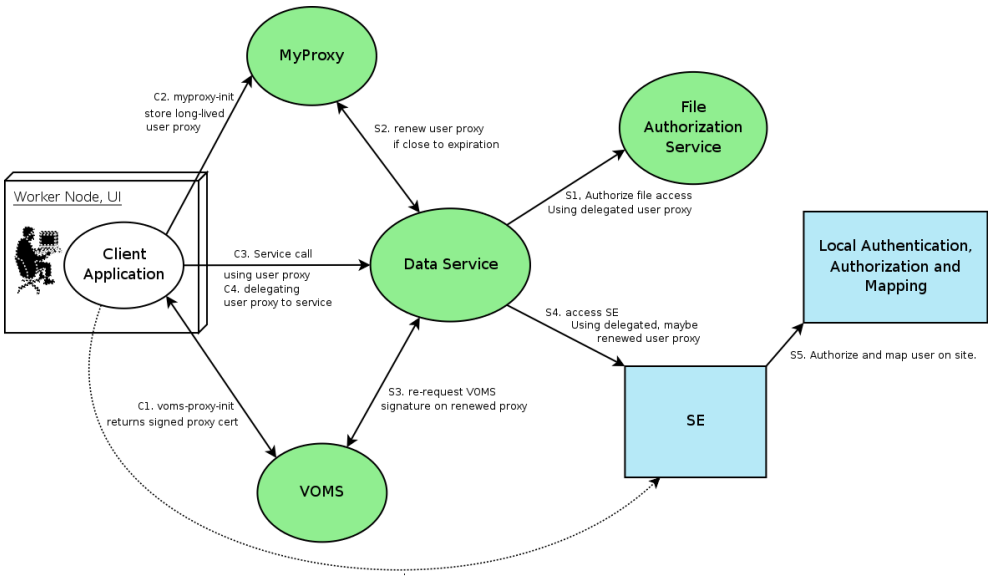

Figure 21: Security interactions of the Grid Data Services: Grid I/O and data transfer. The steps are explained in the text. The dotted line represents direct, non-Grid access.

On the Grid it is very difficult to provide uniform security semantics for data since every Storage Element may be built on different local services with conflicting semantics. It is possible, however, to provide uniform file access semantics using the PKI infrastructure. Figure 21 shows the steps that are needed to authorize Grid file access (either through I/O or through the data movement services). The details of Figure 21 are explained below.

Client

C1 The Client first needs to acquire a VOMS proxy by issuing `voms-proxy-init`. The VOMS server returns a signed VOMS proxy, with some additional fields in the proxy. The ones relevant to the data services are:

DN The DN provided in the user certificate identifies the user. Its uniqueness within its VOMS domain is guaranteed by the VOMS server.

VO The VO name is also provided as part of the VOMS proxy. This is an extension.

Groups The user may be member of one or several VOMS groups. These groups are also provided as part of the VOMS proxy extension.

C2 The user can also put a long-lived certificate into a MyProxy server. 'Normal' certificates last for a few hours (12 is the default), which may be insufficient for some tasks (like very long lasting data transfers). Therefore a longer proxy can be acquired by the user upon request. However, a long lasting proxy should not be used directly (otherwise the proxy concept would be useless) so it is kept in a MyProxy server, which uses it to extend the lifetime of short-lived proxy certificates. The lifetime of the short proxies may of course be only extended up to the total lifetime of the long-lived proxy.

C3 After having acquired a proxy and optionally having placed a long-lived one into MyProxy, the user can contact the service (either the Grid I/O or File Transfer/Placement Service), which will successfully authenticate and authorize the user if the credential is valid.

C4 Although the user may not be aware of this process, the user also delegates his credential to the server so that it can further process the request on the user's behalf.

Server

S1 After the server receives the client's request (C3), it not only checks the DN and the VO to be valid for the given server, but it also contacts the File Authorization Service (FAS) using the user's delegated credentials (from C4). It is as if the user would contact the FAS himself. The FAS will authorize the operation, i.e. whether the given file may be accessed, read or written by the user. The FAS takes the DN and the Groups of the user into account.

S2 Upon successful authorization with the FAS, the user's request may need some babysitting. In the File Transfer Service, it is common that requests sit on the Transfer Queue for hours. If the user's delegated proxy nears expiration, a Transfer Agent Actor will contact MyProxy and ask for a renewed proxy.

S3 VOMS signatures and attributes may need to be renewed as well, since they expire also periodically; so the same Actor also has to call VOMS to fill in the necessary attributes in order to get a fully renewed VOMS proxy. These two steps are not necessary in the synchronous Grid I/O case.

S4 Finally, if the user's certificate is valid and not expired, the server will contact the SE (either its native I/O or GridFTP or SRM interface) using the user's delegated proxy and will try to access the data.

S5 On the SE server side, it is foreseen that for each access the SE also authorizes and maps the user locally (based on the certificate it's being handed with the request) by contacting the site-local (*blue*) authorization and mapping service.

9.5.1 File Ownership and Authorization Details

In this section we discuss some of the implications of the model presented above in detail.

The data stored on some site local storage is ultimately owned by a site local owner. This owner has to be identifiable by the site for accounting and auditing purposes locally, but it does not have to be the storage system itself that specifies who corresponds to a given local user. If we take the SE as given in

Figure 13 a user can access all its interfaces using a non-grid certificate or some local account directly (*direct local access*), as if there was no Grid context (if the user is known to the system). This access is represented by the dotted line in Figure 21. However, as mentioned above, the semantics of access may be different for this direct local access as opposed to the Grid access using the mechanism described before (see Figure 22 for the difference between Grid and native I/O access and mapping). For example, if direct local access is used, the SE will not check authorization with the FAS since it has no way of knowing which FAS for what VO to contact if the user simply comes with a (valid) local account name and local group (see also Figure 22). In order to assure the same security semantics everywhere, there are two options:

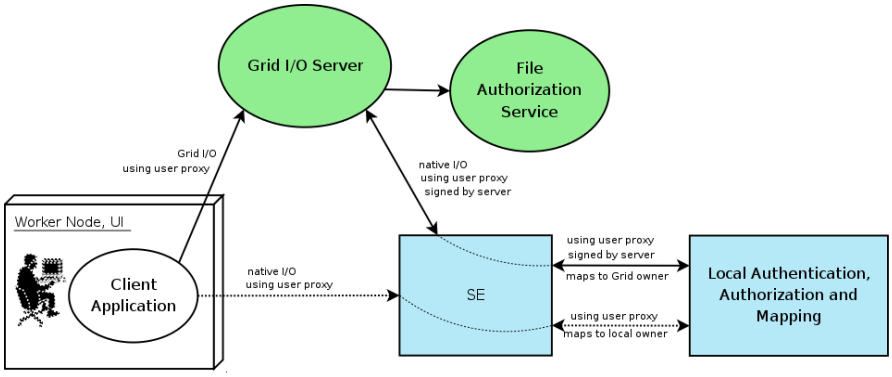

Figure 22: Certificates being used to give a user access to data either through the Grid I/O or through native I/O directly. The Grid services need to sign the user's certificate so that the local mapping is done correctly, and for the right operations (which is what is checked with the FAS).

1. If the local access mechanism provides ACL semantics which can be mapped on the ACLs provided by the FAS, the Grid services may simply map all users to local users (through the local authorization and mapping service). However, some service will need to keep the ACLs in sync between the FAS and the local SE.

2. The data in the SE will need to have full access through the Grid services. In conventional storage facilities where no ACL support is available, this means that the Grid services **own** the files. If ACL support is there, it simply means that the Grid services have full rights on all files exposed through the Grid.

If the local storage DOES NOT support ACLs but the VO wants to secure Grid access to it with full Grid ACL support, this can only be done using option 2, with the Grid services owning the data. In this particular case however, *direct local access using a site local user id* will **not necessarily be possible anymore**, unless for example the Grid group is the same as the user's group when it is mapped to the local SE security context. However, if such mappings exist, it is a serious breach of the security offered by the Grid ACL and enforced by the Grid Services, which both the users and the service providers need to be aware of. Site local protocols that do *not* need a local ID but can authorize the user with its Grid certificate, are slightly better, since the users that should be able to access the data can be mapped to the same account as the Grid service if the SE does not support native ACLs. However, in this case the given user will be able to access *all* files of a given VO. The bottom line is that storage services that do not offer ACLs can only be trusted from the Grid point of view if all of the data in such storages are owned by the Grid I/O and transfer services and no other user is mapped to this user. This is no surprise, as this is the mechanism which is used in all comparable systems to enforce incompatible security semantics

(e.g. a relational database management system (RDBMS) owns all its partitions and files and therefore all data has to be accessed through the RDBMS interfaces, not the native file interfaces the operating system provides).

If the local storage DOES support ACLs, we can analyze the issues we face for the two options in detail:

Option 1 Semantics and Issues. Option 1 is only workable if the FAS ACL semantics are identical to the local SE semantics. This means that the storage has to provide the same semantics as the FAS (i.e. for gLite, NTFS-like ACLs and not POSIX – see Section 9.3.5), with the same set of access control elements as supported by the FAS. There is also the need for a consistency service which has to be provided for each storage service type separately, keeping the ACLs on storage and in FAS in sync. In EGEE we do not foresee to provide any such service, since we do not know of any storage that provides the same ACL semantics. So option 1 again does not seem workable.

Option 2 Semantics and Issues. If the local storage supports ACLs, it is easy for the local administrator to add the Grid service user to the list of users having authorized access to the files that are to be 'in the Grid'. This also works with storage systems with incompatible ACL semantics, since Grid access authorization would still be enforced through the FAS.

There are three issues with this model:

1. If the local owner of the data manages the ACLs locally outside of the Grid, i.e. adding new users to the 'read' list for example, the Grid owner may not be aware that his data is suddenly readable by another person at a given site. However, since usually the Grid owner (who owns a given file according to the FAS) and the local owner are one and the same person, this is probably a minor point.
2. If the Grid owner manipulates the ACLs, for example by giving additional people 'read' rights on the data, these people will not automatically get *local access* read rights on the storage through the native interfaces, i.e. they would need to get local accounts and be added locally to the storage ACL lists to be granted the given access control capability. Since the Grid and local owners are usually the same, this may be a minor issue again.
3. The biggest problem is if a user wants to have *Grid access* to data stored in an SE using his own certificate while he is not the local owner of the data. This means that somehow the certificate that the user presents locally would need to have extra information embedded in it so that he is mapped to the *local Grid user*. This would mean that if a user needs native I/O like shown in Figure 22, but has to be mapped into the proper Grid user, the client has to request a service-signed proxy from the Grid I/O server before using native I/O (see Figure 23). This is a possibility that we foresee to explore further in EGEE (it is not possible in gLite yet). This is the model that has been proposed by the Globus project in their Community Authorization Service (CAS) [66].

In summary, option 2 is the only way to secure data without explicitly compromising its access through 'back-door' access, i.e. the Grid service needs to have ownership rights on the data managed by it. The issues that arise from this model are not easy to reconcile but also not impossible, while option 1 leaves the door open for unwanted access with no clear solution of how to avoid it.

9.5.2 User and Service Certificate Usage

In this section we discuss the usage of certificates by the data management services.

In EGEE we have decided to support the Grid Security Infrastructure (GSI) based on PKI certificates as introduced by the Globus Project [44], and as described in Section 5.1.

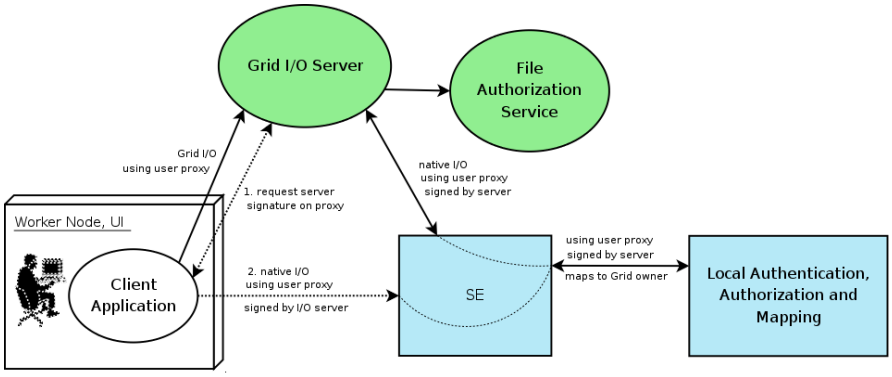

Figure 23: Certificates being used to give a user access to data both through the Grid I/O and through native I/O, mapping it always to the Grid user and authorizing through the FAS. For native I/O, the Grid I/O server needs to be contacted to provide a server-signed proxy.

According to this mechanism, the user identifies himself by presenting a proxy certificate (step C3 in Figure 21) and delegating it to the service (step C4). As we have discussed above, the service would need to *sign* the delegated user proxy so that it can contact the site local service with a certificate that carries both the user and the service information. (Actually it does not matter whether it's the service certificate containing the user proxy or the user proxy signed by the service.) As shown in Figure 22, the Grid I/O server contacts the SE with such a *dual* certificate. The SE again uses this certificate to contact the local mapper so that the proper local account can be selected. This mechanism allows to

- Authenticate the user locally at the site. Since the user identity is transferred with the proxy, the site has full control over who has access to its resources.
- Map the user into the proper local account. Since the service certificate or signature is part of the certificate, the mapping can be done into the proper Grid owner. If the service certificate is used with an embedded user cert, it is not a problem trusting the service and doing the mapping, otherwise the local service would still need to keep a table of trusted IP addresses in order to avoid man-in-the middle attacks (someone may fake a service signature), assuring that the message comes from a trusted machine.

10 HELPER SERVICES

10.1 BANDWIDTH ALLOCATION AND RESERVATION

The European Research Area is currently served by a set of NRENs linked via a high-speed pan-European backbone, the GEANT network, built and operated by DANTE[11]. The EGEE Grid will use these networks to connect the providers of computing, storage, instrumentation and applications resources with user virtual organizations. Grid demonstration projects have shown that Grid applications can generate very high volumes of network traffic that can exceed the current aggregate flows from non-Grid usage, and will therefore demand new and innovative features of GEANT and the NRENs over and above the current best-efforts IP service.

JRA4 is putting in place a web service to implement bandwidth allocation and reservation. This will allow the usage of the network to be controlled and balanced and to categorise and prioritise traffic flows so that users and the layers of Grid middleware receive the required level of service from the network.

[11] http://www.dante.net

For EGEE-1 the service available from the networks is IP-Premium, an IP-level implementation of Differentiated Services [73]. It should be noted that the European Network Survey carried out by SA2 showed that, while the service is available on the backbone (GEANT), it is still far from being widely deployed on the NRENs [39]. We will install a pilot service on a limited number of NRENs who have expressed in principle an interest to install the GN2 software under development.

In the following we give an overview on the BAR architecture. More details can be found in [22].

10.1.1 BAR Architecture Overview

Figure 24 shows how a BAR service interacts with High Level Middleware (HLM) and network services. An example of a HLM service could be the Data Scheduler (see Section 9.4).

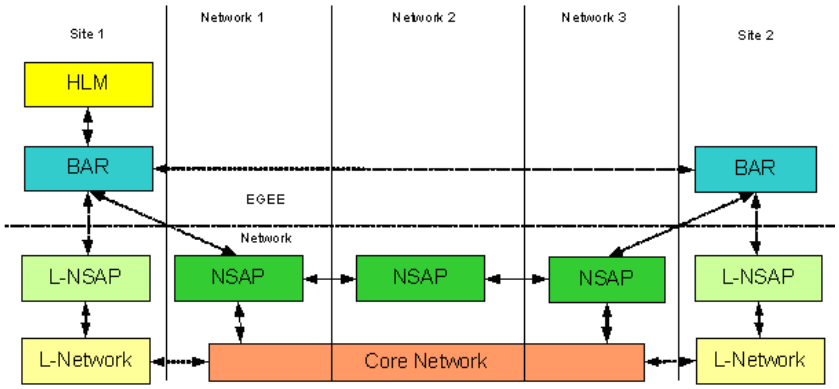

Figure 24: BAR service and how it fits into the bigger picture

There are three webservices involved in Bandwidth Allocation and Reservation: BAR, Network Service Access Point (NSAP) and Local Network Service Access Point (L-NSAP).

BAR: Receives the HLM request in a network-neutral language. It passes the request on to its designated NSAP and L-NSAP for the configuration of the backbone and local network respectively, and also to the BAR at the target destination. When interacting with NSAPs and L-NSAPs, BAR translates an HLM request into a network-oriented request.

NSAP: Present in the backbone (GEANT and NRENs). They are concerned with the configuration of network equipment on the backbone. Their functionality involves sending notification to BAR of the success of the request. As explained in [24], the NSAP abstracts the network-specific services but still speaks a language that is network-oriented.

L-NSAP: Of equivalent functionality to the NSAPs but concerned with the configuration of equipment on the local network of the source and destination sites (*last-mile problem*).

The HLM and BAR entities belong to the EGEE administrative domain, and as such they are developed by EGEE. Network service providers are free to implement NSAP (and L-NSAP) however they see fit. The only constraint is that the standard BAR-NSAP and BAR-L-NSAP interfaces defined by EGEE and GN2 must be honoured.

JRA4 will deploy the GN2 NSAP solution on the pilot service. L-NSAPs belong to the administrative domain of the end-sites and the resulting diversity of local equipment and policies complicates the last-mile problem. JRA4 will likely only investigate reference implementations of L-NSAP.

10.1.2 WEB SERVICES AND THEIR INTERFACES

The HLM-BAR and BAR-NSAP interfaces are represented as web service interfaces, each being composed of set of Port Types (PTs) showing the operations that can be performed.

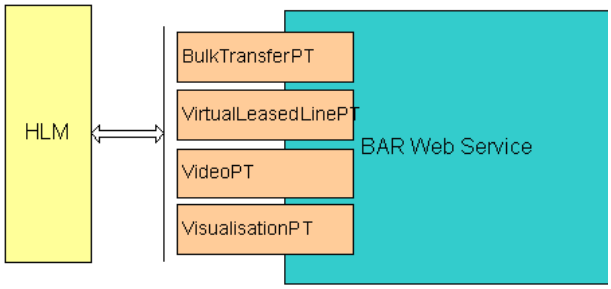

Figure 25: BAR Web Service

BAR Web Service Figure 25 shows the BAR service interface that is exposed to the HLM. Based on four Service Types defined in [24], the BAR web service provides four Port Types as its interface. Bulk Transfer (BT), Virtual Leased Line (VLL), Video and Visualisation Port Types corresponding to the BT, VLL, Video and Visualisation Service Types defined in [24]. The framework is extensible as extra Port Types can be added to support additional Service Types. It is also flexible since each service type is free to define, if necessary, a completely different interface. That is, operations and their signatures can be completely different from one service type to another.

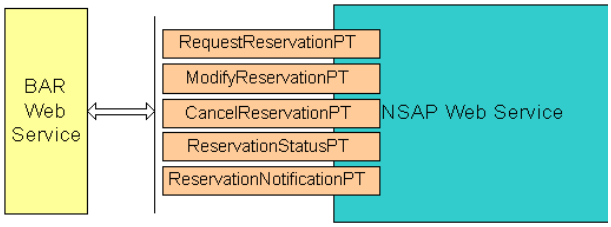

Figure 26: NSAP Web Service

BAR-NSAP interface and NSAP Web Service Figure 26 shows the interface exposed by the NSAP Web Service. Its interface is used by a BAR service to access network-oriented services in order to fulfil HLM requests. The parts of the NSAP interface and the parameters they accept are described in [24]. The port types RequestReservationPT, ModifyReservationPT, CancelReservationPT, ReservationStatusPT and ReservationNotificationPT correspond to the Request Network Service, Modify Network Service, Cancel Network Service, Query Network Service Status and Network Service Notification parts of the interface respectively and accepts the parameters described in [24].

Note that translation is necessary between the operations in the BAR service and the ones in the NSAP service.

Scope for EGEE 1 Out of the four BAR Port Types shown in Figure 25, only Bulk Transfer and Virtual Leased Line can be supported. This is because only two service types, BT and VLL, are widely available on the underlying networks presently.

10.1.3 SECURITY

Figure 27 shows the overall security architecture for the BAR. The entities involved, their certificates, and the information stored pertaining to authorisation are shown.

Security is required at the following levels:

- Between the Higher Level Middleware (HLM) or other Clients and the Bandwidth Allocation and Reservation (BAR) service, to ensure that no-one can alter or intercept the requests and to ensure that only authorised users can successfully make requests to the BAR service via either their Client or the HLM.
- Between the BAR service and the Local Network Service Access Points (L-NSAPs)/Network Service Access Points (NSAPs), to ensure that no-one can alter or intercept the requests and to ensure only authorised BAR services can successfully make bandwidth reservations.
- Between BAR services communicating with each other (to set up the L-NSAPs at each end-site involved in a bandwidth reservation), to ensure that no-one can alter or intercept the requests and to ensure only BAR services that have been delegated a Proxy by an authorised user can successfully make requests to another BAR service.
- Between NSAP services; this is outside the scope of JRA4 and will not be considered further.

The security architecture is defined in [23].

10.2 AGREEMENT SERVICE

The Agreement Service implements the communication protocol used to exchange information about Service Level Agreements (SLAs) and defines the SLA structure. Signalling requires the exchange of one reservation and allocation request between the *Agreement Initiator* and the *Agreement Service*. The Agreement Service is responsible for ensuring that the SLA guarantees are enforced by a suitable *service provider*. In addition, the Agreement layer defines the mechanisms to: 1. expose information about types of service and the related agreement offered (the *Agreement templates*); 2. handle the submission of service requests (the so-called *Agreement offers*).

The gLite architecture for resource reservation is based on three notions: the agreement initiator, the agreement service and the service provider, as illustrated in 28, according to the approach followed by the GRAAP Working Group of the GGF [38].

An agreement initiator uses the agreement service to obtain appropriate agreements with reservation and allocation service providers, which are typically co-located with physical or logical resources. In the gLite architecture, agreement initiators would include the workload management system (WMS), the data scheduler (DS), and the user; while reservation and allocation service providers (RASPs) would be associated with the logical representation of physical resources: the computing element (CE), the storage element (SE), and the network service access point (NSAP). The agreement initiator forwards a list of potential resources and an agreement offer to the agreement service. The agreement service will contact the relevant RASPs and the result of the negotiation is communicated back to the initiator. If an agreement can be reached with one of the RASPs, the agreement service returns information about the successful agreement, while in case of failure, an error code is sent back.

10.2.1 RESERVATION AND ALLOCATION SERVICE PROVIDER

The reservation and allocation service provider is responsible for performing admission control, i.e. of checking if the requested service can be actually guaranteed to the user, and of applying the configuration

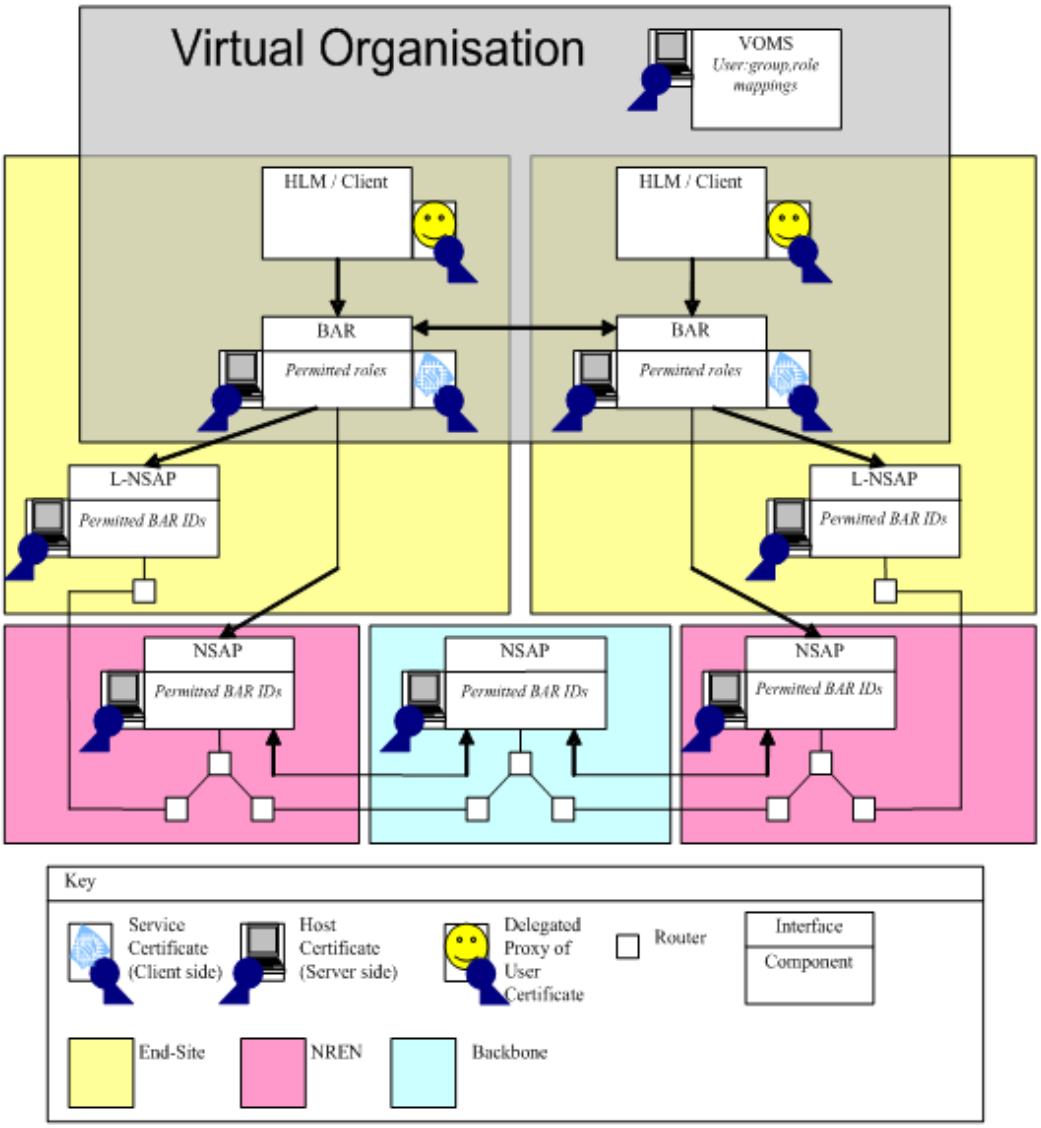

Figure 27: BAR Security Architecture

techniques needed to enforce the guarantees when agreed reservations start. Admission control depends on the user's identity, on the policies applicable to her/him and to the availability of a sufficient amount of resources to satisfy the request. Enforcement of service guarantees requires actions on the physical resources. An example of action is the configuration of network devices in case of bandwidth reservation. For brevity reasons, in what follows the reservation and allocation service provider is simply named service provider. The service provider is characterized by the following list of properties.

10.2.2 AGREEMENT SERVICE

The service provider is invoked by one agreement service, a collective-layer service characterized by the following functionality and set of properties.

- The agreement service interacts with the agreement initiator and with the service providers that can potentially satisfy one user's request. In particular, the agreement service accepts reservation

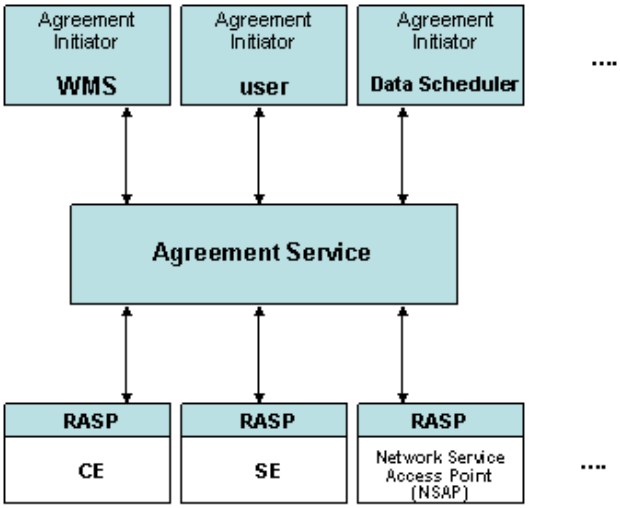

Figure 28: gLite Advance Reservation Architecture

requests (the so-called agreement offers) from the initiators, it checks the compliance of the offer, and in case of success it hands it to one or more service providers.

- The agreement service advertises the service provider capabilities through Agreement Templates. The template is a skeleton defining the structure of the contract. It includes a list of creation constraints, which define the rules that need to be satisfied by the service description terms in the offer. The template is an XML document. The templates advertised by one agreement service explicitly define the types of service the agreement service can handle, and consequently depend on the service providers the agreement service interacts with. In addition to this, different templates for the same service may be exposed in order to offer various types of agreement to different user groups, where each agreement template expresses one specific service abstraction. Membership of users to groups and information about roles and capabilities can, for instance, be handled through the VO Membership Service (VOMS) [3].

- The agreement service can invoke one or more service providers. In order to do so, it needs to support the service provider interface. Different invocation strategies can be supported depending on the Grid environment; this is transparent to the initiator.

- The agreement service can perform negotiation, i.e. agreement offer attributes from the initiator can be tuned during the negotiation phase according to the response received from one service provider.

- The service description terms specified in the agreement offer may be different from the parameters required by the service provider interfaces. In this case, the agreement service translates high-level service description terms (from the agreement initiator) to low level service specific terms requested by the service providers, depending on the case.

- The agreement service needs to provide information about both the status of agreements under negotiation and the attributes of the agreements established. These information elements need to be appropriately logged, for instance in the gLite Logging and Bookkeeping component.

- The agreement service rejects offers from non-authorized initiators.

10.2.3 AGREEMENT INITIATOR

The agreement initiator is responsible of triggering the negotiation of a new agreement by interacting with one agreement service. According to the gLite architecture (Figure 28), the agreement initiator will for instance be the WMS, the Data Scheduler or the user.

The input information sent to the agreement service includes: 1. the quantitative description of the service profile requested; 2. one or more identifiers of the reservation and allocation service providers that the initiator wishes to be contacted by the agreement service.

The initiator can be the consumer of the resource reserved, or a proxy. The former case can apply when the consumer has a-priori knowledge of the resources to be reserved. If this is not the case, the consumer can decide to submit a service profile request to the WMS (the so-called agreement offer), which performs resource discovery according to the resource requirements specified by the invoker, and contacts the agreement service on behalf of it.

10.3 CONFIGURATION AND INSTRUMENTATION

10.3.1 OVERVIEW

The Configuration and Instrumentation Services are a first attempt at introducing common, standard-based configuration and instrumentation functionality in the gLite grid middleware.

The term configuration is here used to represent the set of information required to transition a service (or service instance) from its initial installed state to a working state or from two different working states according to given functional and environmental condition. The term instrumentation is used to represent the set of operations that a service (or service instance) must implement in order to store, query or change its configuration, state and management information.

Since instrumentation can be used to query a service state, it can also be used to implement monitoring functionality in the service for example by periodically querying the values of specific service attributes either in real-time or via some queuing mechanism. Although the configuration and instrumentation services do not implement themselves this functionality, they provide the basic building blocks to do so.

The gLite configuration and instrumentation architecture is represented in Figure 29. It is based on the web service paradigm as are the other services composing the gLite middleware stack. A basic principle that has been followed is that the system should work even in the absence of the configuration and instrumentation service, although the resilience, robustness and ease-of-use of the middleware may decrease.

The main components are the Configuration Service and Clients, the Instrumentation Interface implemented by all services and the Configuration and Instrumentation Proxies. The component responsibilities are explained in more details in the following sections.

10.3.2 CONFIGURATION SERVICE

The Configuration Service is responsible to provide a standard interface to configuration information stored in a number of back-end repositories. Consumers of configuration information should not depend on the type of storage (files, RDBMS, LDAP repositories, grid File Catalogs, etc), but only on a set of agreed interfaces to store and retrieve information and appropriate schemas.

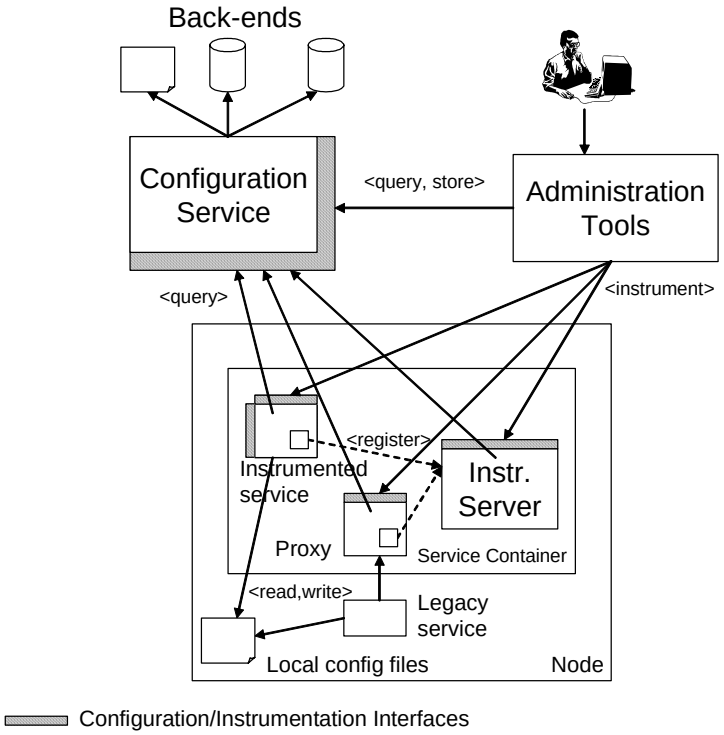

Figure 29: The gLite Configuration and Instrumentation Architecture

The Configuration Service receives queries from clients (services or administration tools), retrieves the information from the existing back-end, formats and verifies the data (according to some validation and consistency rules) and pass the information to the requestor.

In the same way, the Configuration Service can receive request to store, modify or delete configuration information using properly formatted messages and updates the back-end as needed.

The service interface will be described in a separate design document using the appropriate notation.

10.3.3 CONFIGURATION CLIENTS

The Configuration Clients are standard information consumers implemented by:

- Services: to get configuration information during the service initialization phase and possibly to put modified values changed by administrators or other services via the service instrumentation interface
- Administration Tools: to get, put and delete configuration information from the back-end repositories

The architecture also provides the possibility for the services to read configuration information from local files without using the configuration service. This is done in order to avoid strong coupling between services and the configuration service.

10.3.4 INSTRUMENTATION INTERFACES

The instrumentation interfaces are an agreed set of operations that all gLite instrumented services must implement. The details of the interfaces will be described in a separate design document using the

appropriate notation, but they should comply with existing instrumentation standards or guidelines (such as the DMTF WBEM [17] or the OASIS WSDM [60]). The interfaces include the operations required to get and set configuration, state and management information from and to the instrumented services.

10.3.5 Service Containers

A service container is implemented in each node (host) where services are deployed. The service container is normally provided by existing third-party applications inside which the web services are running, like Tomcat for example. In order to support the functionality required by the instrumented services, such as service registration and discovery, the service container is extended with instrumentation functionality implemented by standard tools like JMX [48] that provides such instrumentation capabilities for Java web services via an MBeans Server and local service MBeans (MBeans are management beans analogous to the EJBs in the SUN J2EE architecture).

10.3.6 Configuration and Instrumentation Proxies

The gLite middleware stack is composed of service implementations of various nature, provenance and history. They are developed in different languages and are more or less suitable for being instrumented using existing standard instrumentation technologies. In order to ease the transition from the current services to a set of fully instrumented services, the use of configuration and instrumentation proxies is foreseen. A proxy is a service fully compliant with the web service paradigm suggested by the gLite Architecture and implementing the configuration and instrumentation interfaces recommended in this section. However, its role is to act on behalf of a "legacy service" implementing the configuration client that interacts with the configuration service and receiving/transmitting instrumentation messages using any appropriate protocol. Internally, the proxy communicates with the legacy service and executes on it the requested instrumentation operations using any existing feature provided by the legacy service. As the legacy services are properly instrumented and implement the standard interfaces and functionality, the proxies can be progressively removed without changes in the external interaction with other actors.

10.3.7 Implementation Considerations

Data Modelling and Encoding The modelling of the configuration information and the objects involved in the configuration and instrumentation operations must be done according to a standard model. The preferred choice to describe information and objects is the DMTF Common Information Model (CIM) [12]. A subset of the CIM model is used. However additional providers using different modelling schemas can be foreseen if the need arises.

The data encoding language at the transport level is XML from which appropriate formats can be derived and objects instantiated within the services and clients using functionality of the local object managers and specific language support.

Communication Protocol The recommended communication protocol is SOAP as described in the general introduction to this architecture document. If CIM is used as suggested, the standard set of CIM Operations over HTTP(S) [12] or the CIM-SOAP protocol being discussed by DMTF/OASIS within the WSDM context can be used.

Authentication All the communication between the configuration service or the instrumented services/proxies and the clients is authenticated using appropriate user or service credentials.

Authorization The execution of configuration or instrumentation operations on the configuration service or the instrumented services/proxies is authorized using ACLs based on user DNs or VOMS roles.

10.3.8 Other Services Used

VOMS and the gLite security infrastructure.

11 Issues

In this section we briefly discuss some of the most urgent issues we believe work is needed on in future. In particular, we discuss emerging standards, database access, virtualization, and reliable message passing.

11.1 Standards

11.1.1 Service Coordination

In service-oriented architectures (see Section 4), services may expose a well-defined set of operations to the clients where the operations are atomic. Atomicity means that an operation either succeeds or fails completely; there are no partial failures or partial successes. As an example, consider the File Placement Service described in Section 9.4.2. It has to coordinate the File Transfer Service and Catalog Service operations in order to expose an atomic behaviour.

So in service-oriented architectures we often build new services by aggregating other services. In a distributed environment, however, there is no guarantee that a service may be contacted at all times. This means that the service which is coordinating operations of other services has to have built-in policies on how to deal with failures.

The policies depend on the semantics of the service. In our example of the FPS, it makes sense to re-try to contact the File Transfer and Catalog Services at certain intervals (until a timeout is reached) to try to complete the operation. However, if we try to copy a file first and fail with the catalog operation, should the copy be removed in the process? (Our answer is yes.)

There are several proposed specifications to deal with the issue of service coordination [83]:

- **WS-Coordination** describes an extensible framework for providing protocols that coordinate the actions of distributed applications or services.
- **WS-Transactions** describes coordination types that are used with the extensible coordination framework described in WS-Coordination. It defines two coordination types: WS-AtomicTransactions and WS-BusinessActivity. Either of these can be used when building applications requiring consistent agreement on the outcome of distributed activities.
- **WS-ReliableMessaging** describes a protocol that allows messages to be delivered reliably between distributed applications in the presence of failures. WS-Reliability from OASIS is another specification that has the same goals. Both specifications might merge into one in the future.

It is an open question how these specifications can be used to achieve the desired functionality of our services. Such an evaluation will have to be made on a case-by-case basis. Most of the WS-* specifications are still in draft stage and subject to change in the near future and have yet to be submitted to any standards body. The extent of their implementation in various tools varies.

11.1.2 ADDRESSING

WS-Addressing is a fundamental specification that defines transport neutral mechanisms for sending messages and addressing web service endpoints. For example both WS-Coordination and WS-ReliableMessaging make use of WS-Addressing. Because of its importance, this specification is nearing recommendation status as a result of a fast track approach to its standardization within W3C. This makes it a safe candidate for adoption by the project in the near term.

11.1.3 WSRF RESOURCES

Very recently, the Grid community has proposed the notion of Resources in the Web Service context by proposing a Web Service Resource Framework (WSRF) as a standard framework for building a rich set of services managing stateful resources [27]. The Global Grid Forum (GGF) bases its Open Grid Service Architecture (OGSA V1.0) on the WSRF model [32].

The motivation to introduce WSRF is as follows [77]: Web services must often provide their users with the ability to access and manipulate state, i.e., data values that persist across, and evolve as a result of, Web Service interactions. And while Web Services successfully implement applications that manage state today, we need to define conventions for managing state so that applications discover, inspect, and interact with stateful resources in standard and interoperable ways. The WS-Resource Framework defines these conventions and does so within the context of established Web Services standards.

The WS-Resource Framework (WSRF) is a set of Web Services specifications that define what is termed the WS-Resource approach to modelling and managing state in a Web Services context.

The WS-Resource approach has been introduced in order to declare and implement the association between a Web Service and one or more named, typed state components [82]. In this approach, state is modelled as stateful resources and codify the relationship between Web services and stateful resources in terms of the implied resource pattern, a set of conventions on Web Services technologies, in particular WS-Addressing. When a stateful resource participates in the implied resource pattern, it is referred to as a WS-Resource.

EGEE cannot adopt the WSRF approach currently because of its immature state. The progress of this important standardisation effort has to be tracked though and the architecture should be such that a future migration to WSRF standards is straightforward.

11.1.4 WEB SERVICE INTEROPERABILITY WS-I

The Web Services Interoperability Organisation is an open industry effort chartered to promote Web Services interoperability across platforms and programming languages [63]. The first basic WS-I document was released in April 2004, and contains simple guidelines on how to use WSDL 1.1 and SOAP 1.1 to be interoperable within the Web Services domain.

These specifications should be adapted by EGEE middleware as soon as the appropriate tooling is available. For legacy and early WS implementations effort should be planned to implement the WS-I guidelines.

11.1.5 NOTIFICATIONS

Many of the gLite services described in this document need to notify the client or other services on changes in their internal state. It is important to adopt a common mechanism for notifications. Unfortunately, there is currently no commonly agreed upon standard for notifications in the WS community: Microsoft is proposing WS-Eventing [84] while IBM and others specified WS-Notification [85].

11.1.6 JOB SUBMISSION RELATED STANDARDS

Several useful systems have been constructed in the past years in many Grid projects for enabling remote access to compute resources. Unfortunately, because of the lack of clear and consolidated standards in this area, these systems are often incompatible.

The specification and adoption of standards in the areas of compute resource management interfaces, and of the language used to specify job characteristics and requirements are therefore needed to allow different implementations to co-exist and interoperate.

Initiatives where these issues are currently tracked and which hopefully will make progress towards the definitions of these standards include:

- the GGF Job Submission Description Language (JSDL) Working Group [79], which aims to specify an abstract standard job submission description language independent of language bindings;
- the newly formed GGF OGSA Basic Execution Service Working Group, which has the objective to develop a a specification for a minimal set of execution management services;
- the CRM (Compute Resource Management) initiative [62], where the goal is to make progress towards the definition of standard compute resource management interfaces.

11.2 DATABASE ACCESS

The current architecture as described in this document is based on the assumption that data is stored in files and that the finest level of granularity for a data item is a file. The File Catalog with its filesystem-like semantics is the manifestation of this assumption. However, many applications store their data not in files but in databases. Such applications should also profit from being able to run in a distributed Grid environment. We believe that the current architecture is flexible enough to accommodate such applications. For EGEE, we define a Metadata catalog interface for which we also provide an implementation. In principle, any database can easily implement this interface on top of its existing interfaces, fully integrating it with the gLite services.

It has been shown that data stored in databases can be made accessible to the applications in a distributed Grid environment. Existing efforts include the EU DataGrid Spitfire project [5], the OGSA-DAI project [68] (which provides an implementation of the GGF Data Access and Integration Services working group's (DAIS-WG) [1] specifications for data access) and efforts from the industry (Oracle, IBM). Data in an existing (usually central) database may be made available to Grid jobs through such mechanisms. We will make sure that gLite is interoperable with OGSA-DAI and applications can use this mechanism to access data in databases. Still, distributing the data in a scalable, secure, controlled manner, co-locating it with the running jobs, making it writable at many sites (i.e. multi-master database replication) is a nontrivial problem that has kept the database community busy for the last two decades. Some commercial solutions exist, but none provide interoperability and platform-independence, which is necessary for Grid deployment.

We are actively involved in the DAIS-WG of GGF and will also make sure that the proper binding layers are provided to existing databases in our user communities. However, some problems might be too difficult to solve within the lifetime of this project.

11.3 VIRTUALIZATION

One of the primary obstacles users face in Grid computing today is that while Grids offer them access to many powerful resources, they offer very little to ensure that this access fulfils the user's expectations

of the resource and the requirements of their code. This is because most Grid platforms today do not support performance isolation: memory demands or scheduling priority of one job will affect another executing on the same platform in an uncontrolled way. One way to address the problem is to build support for performance isolation on a per-process basis into the operating system. However, we are typically interested in controlling resource usage on the granularity of groups of processes, in particular a group of processes belonging to a specific user or virtual organization (VO).

Another manifestation of the lack of control is that while Grids typically offer access to a group of resources of diverse software environments a user's application requires a very specific, customized environment. Between computing clusters, variations in operating system, middleware versions, library environments, and file system layouts all pose barriers to the portability of applications. Applications that work on a developer's desktop may only function "out of the box" on a small fraction of the total number of compute resources potentially available to the scientist.

Virtual machines (VMs), integrated into existing Grid security and resource management infrastructure, provide a compelling solution to these problems. The concept of machine virtualization allows a client to create a custom execution environment configured with a required operating system and software stack on almost any physical computer. Moreover, sharing between such environments can be orchestrated based on scheduling resources between virtual machines themselves assuring controlled resource usage. In addition, VMs allow the client to suspend the operation of a "guest" virtual machine by writing the VM's state to disk; such saved VM "images" can be subsequently resumed allowing the migration of the VM's frozen state across different resources.

A virtual machine provides an isolated virtualization of the underlying physical host machine. Software running on the host, called virtual machine monitor (VMM) or "hypervisor", is responsible for supporting the perception of multiple isolated physical machines by intercepting and emulating privileged instructions issued by the guest virtual machines. A VMM typically provides an interface allowing a client to start, pause or stop multiple guests. A VM representation contains a full image of the VM's RAM, disk, and other devices, allowing its state to be fully serialized, preserved, and restored at a later date. Recent exploration of "paravirtualization", specifically the development of the Xen hypervisor has led to substantial performance improvements in virtualization technologies that make VMs a very cost-effective solution typically resulting in only very small performance degradation.

11.4 RELIABLE MESSAGE PASSING

Messaging systems were conceived in order to solve the problem of reliable message passing across wide area networks. This is especially important for businesses where messages have to be delivered with zero loss like for banks using Automated Teller Machines. Such systems employ industrial-strength products like IBM's MQSeries [55] and Microsoft's MSMQ [45].

These messaging systems operate with a store-and-forward message queue. The sender of the message expects someone to handle the message asynchronously. There are several standards in the domain of messaging, a very widely used standard is the Message Passing Interface (MPI) [31]. MPI is designed for high performance on both massively parallel machines and workstation clusters. Messaging frameworks based on the classical remote procedure calls include CORBA [61] from OMG, DCOM [18] from Microsoft and Java RMI [54] from Sun Microsystems.

We speak of message oriented middleware (MOM) if the publish/subscribe model is applied, where the implementation takes care of routing the right content of the messages from the right publisher to the right subscriber. Industrial strength solutions in the publish/subscribe domain include products like Rendezvous [14] and SmartSockets [15] from TIBCO. Other related efforts in the research community include NaradaBroker [76] and Elvin [6]. Sun Microsystems have pushed Java to include publish subscribe features into its messaging middleware through JINI [49] and JMS [51]. JMS aims to offer a unified API across publish subscribe implementations. Various JMS implementations include solutions

like SonicMQ [74] from Progress, Java Message System Queue [71] from Sun and FioranoMQ [13] from Fiorano. In the open source community there is Joram [53], JBOSS [52], ActiveMQ [2] and OpenJMS [69]. Also MQSeries and Microsoft MSMQ as well as some of the above mentioned publish/subscribe MOM systems implement JMS.

Reliable message passing may also be used in a Grid architecture to coordinate the state of services residing at different sides in the wide area network. By using a standard like JMS it is possible to choose from a wide range of existing solutions, both commercial and open source to deliver the necessary Quality of Service to the distributed Grid services. Message passing will increasingly become one of the driving factors in distributed service provisioning for Grids, as is being demonstrated by NaradaBrokering for example.

For EGEE, we intend to use JMS to keep the StorageIndex up to date with the local site File and Replica Catalogs in a distributed deployment scenario.

12 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. 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), Grid3 (http://www.ivdgl.org/grid2003/), OSG (http://www.opensciencegrid.org/) or NorduGrid (http://www.nordugrid.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, etc.).

Figure 30 shows a typical deployment scenario for a site providing computing and storage resources. Figure 31 shows the additional services a VO might use. Note that we focus on job submission, data management, and information and monitoring in these figures and do not cover additional services like accounting, bandwidth allocation, agreement, and configuration. In depicting the services we try to separate resource and virtualized services. Naturally, all the services in Figure 31 are virtual ones.

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; in reality, this goal might not always be achievable and at least stubs or dummy services might be needed, however, the service design and implementation should proceed in that direction. The inverse of this argument needs to be supported as well: in certain circumstances there might be the need to include additional services into the Grid system. While a service oriented architecture in general foresees this dynamic extension, the service implementations also need to be compliant with this goal, by using dynamic discovery mechanisms, for instance.

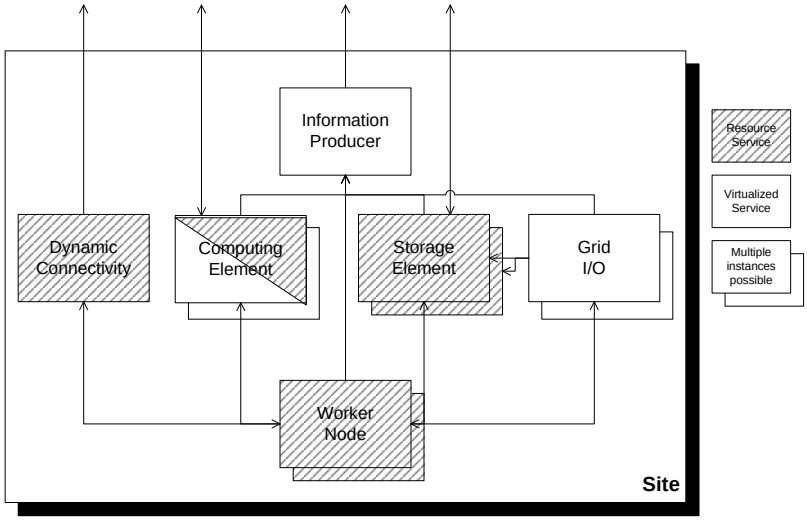

Figure 30: Typical Site Deployment Scenario

13 CONCLUSIONS

This document presented the revised architecture of the EGEE Grid middleware, called *gLite* using a service oriented architecture approach. Five main logical service groups have been identified which themselves contain a set of services. The architecture of these services has been described and we discussed the differences between resource and virtualized services.

The services have been put in context to requirements and a number of issues, in particular with respect to ongoing standardization efforts have been discussed.

The detailed specification of these services will be described in a separate design document (Deliverable DJRA1.5) which will be a revision of the original document DJRA1.2 (https://edms.cern.ch/document/487871/).

It is also worth noting that the architecture described in this document is subject to a continual evolution, based on experiences with deployment and usage of the implementations of the services described, user feedback, and the evolution of application requirements.

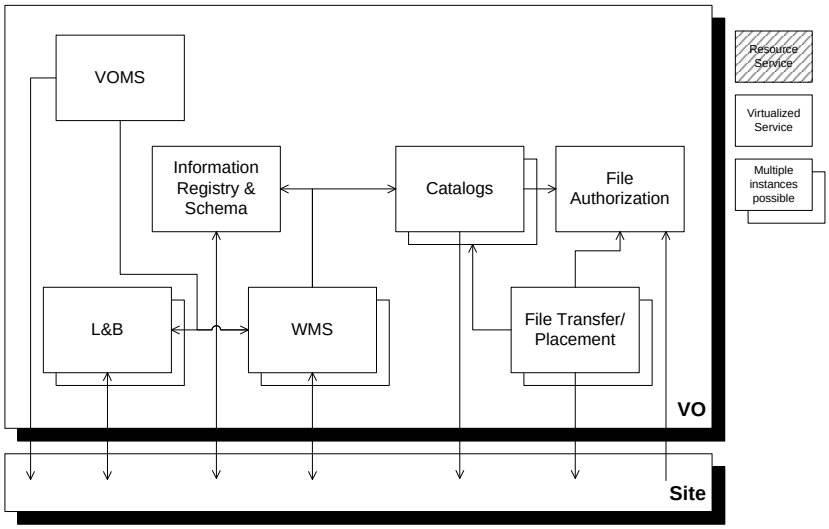

Figure 31: Typical VO Deployment Scenario

REFERENCES

[1] GGF Data Access and Integration Services Working Group. Data Acces and Integration Services. http://forge.gridforum.org/projects/dais-rg/document/.

[2] ActiveMQ. Webpage. http://activemq.codehaus.org/.

[3] Alfieri R. et al. VOMS, an Authorization System for Virtual Organizations. In *Grid Computing, First European Across Grids Conference*, 2004.

[4] Alexander Barmouta and Rajkumar Buyya. Gridbank: A grid accounting services architecture (gasa) for distributed system sharing and integration. In *Proceedings of the 17th Annual International Parallel & Distributed Processing Symposium (IPDPS 2003) Workshop on Internet Computing and E-Commerce*, 2003.

[5] William Bell, Diana Bosio, Wolfgang Hoschek, Peter Kunszt, Gavin McCance, and Mika Silander. Project spitfire - towards grid web service databases. In *Global Grid Forum Informational Document (GGF5)*, Edinburgh, Scotland, July 2002.

[6] Bill Segall and David Arnold. Elvin has left the building: A publish/subscribe notification service with quenching. In *AUUG97*, September 1997.

[7] P. Buncic, F. Rademakers, R. Jones, R. Gardner, L.A.T. Bauerdick, L. Silvestris, P. Charpentier, A. Tsaregorodtsev, D. Foster, T. Wenaus, and F. Carminati. Architectural Roadmap towards Distributed Analysis. Technical report, LHC Computing Grid Project, October 2003.

[8] Steve Burbeck. The Tao of e-business services. http://www-106.ibm.com/developerworks/webservices/library/ws-tao/.

[9] F. Carminati, P. Cerello, C. Grandi, E. Van Herwijnen, O. Smirnova, and J. Templon. Common Use Cases for a HEP Common Application Layer – HEPCAL. Technical report, LHC Computing Grid Project, 2002. http://project-lcg-gag.web.cern.ch/project-lcg-gag/LCG_GAG_Docs/HEPCAL-prime.pdf.

[10] F. Carminati and J. Templon (Editors). Common Use Cases for a HEP Common Application Layer for Analysis – HEPCAL II. http://lcg.web.cern.ch/LCG/SC2/GAG/HEPCAL-II.doc.

[11] A. Chervenak et al. Giggle: A Framework for Constructing Sclable Replica Location Services. In *Proceedings of Supercomputing 2002 (SC2002)*, 2002.

[12] Common Information Model. http://www.dmtf.org/standards/cim.

[13] Fiorano Corporation. A Guide to Understanding the Pluggable, Scalable Connection Management (SCM) Architecture. http://www.fiorano.com/products/fmq5_scm_wp.htm.

[14] TIBCO Corporation. Rendezvous. http://www.tibco.com/software/enterprise_backbone/rendezvous.jsp/.

[15] TIBCO Corporation. SmartSockets. http://www.tibco.com/software/enterprise_backbone/smartsockets.jsp/.

[16] DANTE. Multi-Domain Monitoring *Perfmonit*. http://www.dante.net/server/show/nav.00100q003002.

[17] Distributed Management Task Force. Web-Based Enterprise Management (DMTF WBEM). http://www.dmtf.org/standards/wbem.

[18] Guy Eddon and Henry Eddon. Understanding the DCOM Wire Protocol by Analyzing Network Data Packets. *Microsoft Systems Journal*, March 1998.

[19] EDG. WP7 Network Services. http://ccwp7.in2p3.fr.

[20] EGEE JRA3. Global Security Architecture. https://edms.cern.ch/document/487004/.

[21] EGEE JRA3. Site Access Control Architecture. https://edms.cern.ch/document/523948/.

[22] EGEE JRA4. Architecture For Bandwidth Allocation And Reservation. https://edms.cern.ch/file/533751/1/EGEE-JRA4-TEC-533751-BAR-arch-v0-5.doc.

[23] EGEE JRA4. Security Architecture For Bandwidth Allocation and Reservation. https://edms.cern.ch/file/571891/1/.

[24] EGEE JRA4. Specification of Interfaces for Bandwidth Reservation Service. http://edms.cern.ch/document/501154/1.

[25] Andy Hanushevsky et al. The SLAC Virtual Smart Card project.

[26] C. Rigney et al. RFC2865: Remote Authentication Dial In User Service (RADIUS). http://www.ietf.org/rfc/rfc2865.txt.

[27] Karl Cajkowski et. al. The WS-Resource Framework, 2004. http://www-106.ibm.com/developerworks/library/ws-resource/ws-wsrf.pdf.

[28] M. Myers et al. RFC2560: X.509 Internet Public Key Infrastructure Online Certificate Status Protocol (OCSP). http://www.ietf.org/rfc/rfc2560.txt.

[29] W. Allcock et al. GridFTP Protocol Specification. Global Grid Forum Recommendation GFD.20, March 2003.

[30] European Grid Authentication Policy Management Authority for e Science. http://www.eugridpma.org/.

[31] Message Passing Interface Forum. A Message Passing Interface Standard, May 1994.

[32] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. In *Open Grid Service Infrastructure WG, Global Grid Forum*, June 2002.

[33] Apache Software Foundation. Chainsaw. http://logging.apache.org/log4j/docs/chainsaw.html.

[34] Apache Software Foundation. log4j. http://logging.apache.org/log4j/.

[35] Apache Software Foundation. Logging Services. http://logging.apache.org/.

[36] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[37] The GEANT2 Project. http://www.genat2.net.

[38] GGF. The grid resource allocation and agreement protocol working group (graap). https://forge.gridforum.org/projects/graap-wg.

[39] M. Goutelle. European Network Overview. http://egee-sa2.web.cern.ch/egee-sa2/EurNetOverview/index.html.

[40] EDG Application Working Group. Joint List of Use Cases. https://edms.cern.ch/document/386184.

[41] EGEE Application Working Group. Biomedical Application Requirements. https://edms.cern.ch/file/474424.

[42] GGF Site AAA Research Group. Grid Authentication Authorization and Accounting Requirements. http://forge.gridforum.org/projects/saaa-rg/document/.

[43] The GGF Grid Storage Resource Manager Working Group.

[44] GSI: Grid Security Infrastructure. http://www.globus.org/security/overview.html.

[45] P. Houston. Building Distributed Applications with Message Queuing Middleware. Microsoft White Paper.

[46] J. Howard, M. Kazar, S. Menees, D. Nichols, and M. West. Scale and performance in a distributed file system. In *Proceedings of the eleventh ACM Symposium on Operating systems principles*, pages 1–2. ACM Press, 1987.

[47] 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.

[48] Java Management Extensions. http://java.sun.com/products/JavaManagement/.

[49] Ken Arnold, Bryan O'Sullivan, Robert Scheifler, Jim Waldo and Ann Wollrath. *The Jini Specification.* Addison-Wesley, June 1999.

[50] Paul J. Leach and Rich Salz. UUIDs and GUIDs, February 1998.

[51] Rich Burridge Mark Happner and Rahul Sharma. Java Message Service Specification., 2000. http://java.sun.com/products/jms.

[52] JBoss Messaging. Webpage. http://wiki.jboss.org/wiki/Wiki.jsp?page=JBossMessaging/.

[53] JORAM: Java Open Reliable Asynchronous Messaging. Webpage. http://joram.objectweb.org/.

[54] Sun Microsystems. Java Remote Method Invocation (Java RMI) - Distributed Computing for Java. White Paper. http://java.sun.com/marketing/collateral/javarmi.html.

[55] IBM MQSeries. Website. http://www.ibm.com/software/mqseries.

[56] O. Mulmo and V. Welch. Using the Globus Toolkit(R) with Firewalls. *Clusterworld magazine*, March 2004.

[57] Network Measurments Working Group. Schema-related work. http://www-didc.lbl.gov/NMWG/#schema.

[58] OASIS. eXtensible Access Control Markup Language (XACML). http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml.

[59] OASIS. Security Assertion Markup Language (SAML). http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security.

[60] OASIS. Web Services Distributed Management (WSDM). http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsdm.

[61] Object Management Group (OMG). CORBA Services, June 2000. http://www.omg.org/technology/documents/.

[62] Workshop on Compute Resource Management. Website. http://www.pd.infn.it/grid/crm/.

[63] The Web Services Interoperability Organization. WS-I Documents. http://www.ws-i.org/Documents.aspx.

[64] E. O'Tuathail and M. Rose. RFC3288: Using the Simple Object Access Protocol (SOAP) in Blocks Extensible Exchange Protocol (BEEP). http://www.ietf.org/rfc/rfc3288.txt.

[65] F. Pacini. JDL Attributes. DataGrid-01-TEN-0142, 2003. http://www.infn.it/workload-grid/documents.html.

[66] L. Pearlman, V. Welch, I. Foster, C. Kesselman, and S. Tuecke. A Community Authorization Service for Group Collaboration. In *Proceedings of the IEEE 3rd International Workshop on Policies for Distributed Systems and Networks*, 2002.

[67] R. Piro, A. Guarise, and A. Werbrouck. An economy-based accounting infrastructure for the DataGrid. In *Proc. of the 4th International Workshop on Grid Computing (Grid2003)*, Phoenix, Arizona, USA, November 2003. SC2003.

[68] OGSA-DAI project. Website. http://www.ogsadai.org.uk/index.php.

[69] The OpenJMS Project. Webpage. http://openjms.sourceforge.net/.

[70] UMich Kerberos PKI Project. http://www.citi.umich.edu/projects/kerb_pki/.

[71] Sun Java Systems Message Queue. Webpage. http://www.sun.com/software/products/message_queue/index.xml/.

[72] IRTF AAAarch RG. RFC2904: AAA Authorization Framework. http://www.ietf.org/rfc/rfc2904.txt.

[73] The SEQUIN Project. http://archive.dante.net/sequin/.

[74] SonicMQ JMS Server. Website. http://www.sonicsoftware.com/.

[75] David Sprott and Lawrence Wilkes. Understanding Service-Oriented Architecture. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnmaj/html/aj1soa.asp.

[76] The Narada Event Brokering System. Website. http://grids.ucs.indiana.edu/ptliupages/projects/narada/.

[77] Globus Alliance Website. http://www.globus.org/wsrf/.

[78] V. Welch, I. Foster, C. Kesselman, O. Mulmo, L. Pearlman, S. Tuecke, J. Gawor, S. Meder, and F. Siebenlist. X.509 Proxy Certificates for Dynamic Delegation. In *3rd Annual PKI R&D Workshop*, 2004.

[79] GGF Job Submission Description Language WG. Website. https://forge.gridforum.org/projects/jsdl-wg/.

[80] GGF OGSA WG. Open Grid Services Architecture – Glossary of Terms. https://forge.gridforum.org/projects/ogsa-wg.

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

[82] IBM Developer Works. Modeling stateful resource with Web services. http://www-106.ibm.com/developerworks/library/ws-resource/ws-modelingresources.html.

[83] IBM Developer Works. Web Services Standards. http://www-106.ibm.com/developerworks/views/webservices/standards.jsp.

[84] WS-Eventing. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/WS-Eventing.asp.

[85] WS-Notification. http://www-106.ibm.com/developerworks/library/specification/ws-notification.