Journal of Grid Computing manuscript No.
(will be inserted by the editor)

Standards-Based Job Management in Grid Systems

Paolo Andreetto · Sergio Andreozzi · Antonia Ghiselli · Moreno Marzolla · Valerio Venturi · Luigi Zangrando

Received: date / Accepted: date

Abstract The Grid paradigm for accessing heterogeneous distributed resources proved to be extremely effective, as many organizations are relying on Grid middlewares for their computational needs. Many different middlewares exist, the result being a proliferation of self-contained, non interoperable "grid islands". This means that different Grids, based on different middlewares, cannot share resources, e.g. jobs submitted on one Grid cannot be forwarded for execution on another one. To address this problem, standard interfaces are being proposed for some of the important functionalities provided by most Grids, namely job submission and management, authorization and authentication, resource modeling, and others. In this paper we review some recent standards which address interoperability for three types of services: the BES/JSDL specifications for job submission and management, the SAML notation for authorization and authentication, and the GLUE specification for resource modeling. We describe how standards-enhanced Grid components can be used to create interoperable building blocks for a Grid architecture. Furthermore, we describe how existing components from the gLite middleware have been re-engineered to support BES/JSDL, GLUE and SAML. From this experience we draw some conclusions on the strengths and weaknesses of these specifications, and how they can be improved.

Keywords Job Management · Grid Interoperability · Basic Execution Service (BES) · Job Submission Description Language (JSDL) · GLUE · Security Assertion Markup Language (SAML)

Paolo Andreetto · Luigi Zangrando
Istituto Nazionale di Fisica Nucleare (INFN), via Marzolo 8, I-35131 Padova, Italy
E-mail: {paolo.andreetto|luigi.zangrando}@pd.infn.it

Moreno Marzolla
Dipartimento di Scienze dell'Informazione, Università di Bologna, Mura Anteo Zamboni 7, I-40127 Bologna, Italy
Tel.: +39 051 2094847
Fax: +39 051 2094510
E-mail: marzolla@cs.unibo.it
Part of this work was done while the author was with INFN Padova, Italy

Sergio Andreozzi
CESIA, Università di Bologna, viale Filopanti 3, I-40126 Bologna, Italy
E-mail: sergio.andreozzi@unibo.it
Part of this work was done while the author was with INFN-CNAF, Bologna, Italy

Antonia Ghiselli · Valerio Venturi
INFN-CNAF, viale Berti Pichat 6/2, I-40127 Bologna, Italy
E-mail: {antonia.ghiselli|valerio.venturi}@cnaf.infn.it

1 Introduction

Many large-scale organizations are currently managing their resources using some kind of Grid middleware. The Grid allows seamless access to remote, distributed resources. In particular, job execution and management is one of the capabilities offered by any Grid middleware, as it enables users to harness the power of large CPU pools for computationally intensive applications.

Unfortunately, transparent and uniform access to resources is guaranteed *within* a middleware, but it is not generally available *across* different middlewares. This means, for example, that job management subsystems have different incompatible interfaces, so that jobs originating on a Grid cannot be forwarded to another Grid based on a different middleware, even if the submitter is authorized to access resources on both.

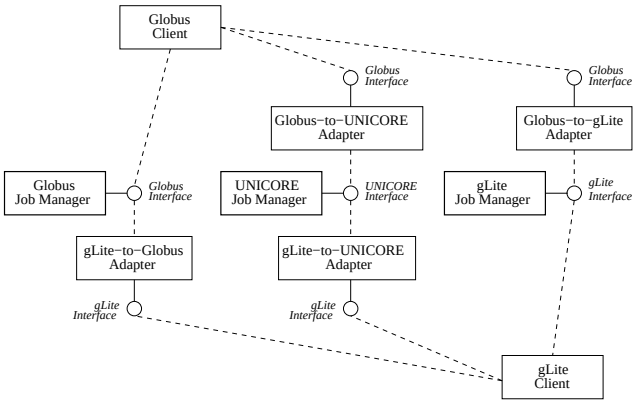
Two specific problems of Grids are *accessibility* and *application portability*. Accessibility includes resource access and portability. A decision to use one particular Grid infrastructure, and hence one Grid distribution, will limit the resources available to the user to those devices running

that Grid distribution. This has implications for all forms of collaborative science and commerce where multiple Grid distributions exist. A related problem is that of application portability. Each Grid specifies its own unique interface to each service. The implication is that user applications are Grid distribution specific: an application written for the gLite middleware [36] will not be compatible with the same application written to use a UNICORE Grid [22].

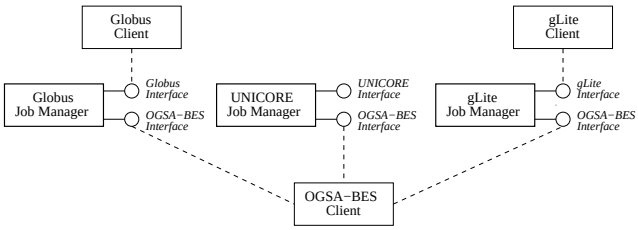
Resource sharing across multiple middlewares is motivated by the increasing demand of scientific applications. The demand is not only for more computational and storage capacities, but also for seamless access to different types of services. As an example, the Wide In Silico Docking On Malaria (WISDOM) Project[1] aims at developing new drugs for Malaria. In silico docking enables researchers to compute the probability that potential drugs will dock with a target protein–in this particular case that potential drugs will dock on the active site of one of the malaria parasite proteins. This step is carried out on resources provided by the gLite middleware and the output of these applications is a list of chemical compounds that may become potentially drugs. This list is not the final compound list, because it must be refined using molecular dynamics. These molecular dynamics computations use the highly scalable assisted model building with energy refinement (AMBER) [14] code that could run on HPC resources within DEISA[2]. Hence, cross-Grid usage significantly accelerates the drug discovery step.

The problem of sharing resources among heterogeneous Grid middlewares has traditionally been addressed by means of ad-hoc components called *adapters*. An adapter connects two specific kind of middlewares, say X and Y. The adapter translates messages originating from X in the format understood by Y; the same is done for all messages originating from Y and directed to X. As such, adapters can be seen as point-to-point solutions enabling the so called *adapters-based interoperability* between two incompatible systems. Solutions based on this approach have limited scalability since among N different kind of middlewares, $O(N^2)$ different adapters are needed.

On the other hand, a more sustainable interoperability can be achieved with the adoption of common, standardized and possibly open (i.e., non proprietary) interfaces and protocols leading to the so called *standards-based* interoperability. Such protocols are usually defined by established organizations and communities, like the Open Grid Forum (OGF)[3] or the Organization for the Advancement of Structured Information Standards (OASIS)[4]. The OGF is a community of users, developers, and vendors leading the global standardization effort for Grid computing. The work of OGF is carried out through community-initiated working groups, which develop standards and specifications in cooperation with other standards organizations, software vendors, and users. The Open Grid Services Architecture (OGSA) [28] describes an architecture for a service-oriented Grid environment for business and scientific use, developed within the OGF.



(a) Adapters-based interoperability



(b) Standards-based interoperability

Fig. 1 Two approaches to interoperability. (a) shows the *adapters-based* scenario where adapters translate specific design aspects from one domain to another. (b) shows the *standards-based* scenario, in which components are enhanced with standards-compliant interfaces

The two interoperability approaches are depicted in Figure 1. As an example, we consider three different job management components, based on Globus [25], UNICORE [22] and gLite [36] respectively. Each component exposes its legacy interface. In 1(a) adapters are used to translate the interfaces and allow clients designed for other middlewares to access each service. In 1(b), each service is enhanced with an additional standards-compliant interface; for job management, the currently defined standard is based on the OGSA–Basic Execution Service (BES) and Job Submission Description Language (JSDL) specifications, which will be described later. Existing platform-specific clients access the services using the legacy interface; on the other hand, every BES compliant client can access the BES interface of any service. Interoperability based on open standards is more scalable than the adapter based one: implementing the same interface on N different middlewares requires effort proportional to $O(N)$.

[1] http://wisdom.eu-egee.fr/
[2] http://www.deisa.org/
[3] http://www.ogf.org/
[4] http://www.oasis-open.org/

The main problem with interoperability is that defining common interfaces for the major Grid services is a difficult and lengthy process. Given that these standards are developed as a cooperation of the major Grid middleware developers, the results are often made of the (very small) common subset of features offered by each Grid.

Contribution of this paper The contribution of this paper is twofold. First, we consider three recently defined standards for Grid middlewares:

- the BES [24] and JSDL [8] specifications, which are OGF standards related to job submission and management. We implemented BES and JSDL in the gLite Computing Resource Execution And Management (CREAM) Computing Element [1], as will be discussed in Section 3.
- GLUE [5], a conceptual model and reference realizations to concrete data models for describing advertisable capabilities of Grid services and resources. We implemented the GLUE 2.0 specification in the GLUEMan service, as will be discussed in Section 4.
- the Security Assertion Markup Language (SAML) [13], an OASIS-standardized XML language for releasing assertions regarding authentication, attributes and authorization. We implemented the SAML specification within the gLite Virtual Organization Membership Service (VOMS) service, as will be discussed in Section 5.

Drawing from the experience gained implementing the above specifications, we then analyze the relevant standards by taking into consideration the requirements of actual production Grids, in particular by focusing on the gLite middleware [36] from which CREAM-BES and VOMS-SAML originate. This allows us to validate the standards with respect to an actual production environment, and identify limitations and area for further improvements.

BES/JSDL, SAML and GLUE 2.0 have been selected because they represent a minimal set of specifications from which a functional job execution service can be realized. Furthermore they are, at the time of writing, quite mature. Support for BES/JSDL is available in different execution services from multiple Grid middlewares (e.g., ARC [21], UNICORE [22], and GridSAM[5] just to name a few). Interoperability between different BES implementations have been assessed during live demonstrations during the Supercomputing 2006 and 2007 conferences [6].

Organization of this paper This paper is organized as follows. In Section 2 we revise related works in the area of standardization of components for Grid middlewares. Section 3 describes the BES/JSDL specifications and their implementation into the gLite CREAM service. Section 4 describes the GLUE 2.0 specification and its implementation into the GLUEMan component. Section 5 describes the SAML specification and its implementation into the gLite VOMS service. We illustrate in Section 6 the complete architecture for job management based on the standards-compliant components previously described. Finally, we discuss in Section 7 the lessons we learned implementing these standards.

2 Related Works

In this section we briefly review some relevant standards related to job management, information modeling and authentication/authorization in Grids.

2.1 Job Management

For the job management service, some relevant standard specifications are Distributed Resource Management Application API (DRMAA) [45] and Simple API for Grid Applications (SAGA) [30]. DRMAA is an API for distributed resource management systems. The scope of DRMAA is limited to job submission, job monitoring and control, and retrieval of the finished job status. DRMAA defines a programming model that enables the development of distributed applications tightly coupled to an underlying Distributed Resource Management System. SAGA is a high level, application-oriented API for grid application development; SAGA includes support for job submission and management, resource discovery, data management, data access and replication and asynchronous notification. DRMAA and SAGA are programming APIs, and not service interfaces. Thus, user applications need to be linked against libraries implementing a DRMAA or SAGA API. On the other hand, any BES client can connect to a BES service without the need to rebuild the code, as BES clients and services are separate components which communicate through message exchanges.

2.2 Information Modeling

In the area of resource modeling, it is important to consider the Distributed Management Task Force (DMTF) [7] and Web-Based Enterprise Management (WBEM) [47] standard suite. The DMTF is an industry-supported organization which develops management standards and promotes interoperability for enterprise and Internet environments. The main standard set that is interesting for this work is WBEM, a set of management and

[5] http://gridsam.sourceforge.net/
[6] http://sc07.supercomputing.org/
[7] http://www.dmtf.org/

Internet standard technologies developed to unify the management of distributed computing environments. The standards included in WBEM are: the Common Information Model (CIM) [15], that is a common definition of management information for systems, networks, applications and services; CIM-XML [17], a protocol that uses XML over HTTP to exchange CIM information; the CIM Query Language [16], a specialized query language for CIM inspired by SQL and XQuery; WBEM Discovery using Service Location Protocol; and WBEM Universal Resource Identifier mapping, defining how CIM elements are identified in the Web.

A typical question that is raised in the area of information modeling in Grids is why the CIM model is not adopted. There are two main reasons why this did not happen so far. The first motivation is a difference in the scope: information modeling in Grids is targeted at capturing the capabilities of Grid services that should be advertised to any potential consumer in order to support activities such as resource selection, inventory and high-level monitoring. On the other hand, CIM focuses on the management aspect of resources and the consumer of this information is the system administrator which needs to inspect and act on the resource. The difference in scope implies that the CIM model does not cover the use cases present in the Grid context.

One could argue that CIM is also extensible and therefore Grid-specific extensions could be defined. The second motivation should be considered here. The CIM model presents a steep learning curve and is mainly driven by industries. Extending it starting from its meta-model requires specialized expertise which is better to involve when a mature model is consolidated.

In the current state of art, the information model of Grid resources is maturing as an independent model from CIM for the given motivations. On the other side, the Grid middlewares lack a management layer, and the WBEM technologies are a mature solution even though they have a difficult adoption path. The short-term vision for convergence could be that the GLUE information model and providers are handled in a WBEM framework with the following constraints: a realization of the GLUE information model in terms of the Managed Object Format (MOF) is provided (this does not imply to derive GLUE from the CIM meta-model, since MOF is a textual representation of UML class diagrams and can be used to represent any diagram); the writing of providers for GLUE and the access to the information from a consumer should be isolated from WBEM-specific details in order to maintain simplicity at both sides of the information chain (producing and consuming the information).

In Section 4 we will consider the short-term vision by proposing a framework (GLUEMan) which enables to manage providers for the GLUE information model using a WBEM implementation and by isolating both producers and consumers of information from WBEM-specific details.

2.3 Authentication and Authorization

The concept of Virtual Organizations (VOs), defined as a dynamic collection of individuals, institutions (resource owners), and resources, emerged as a central concept within world-wide Grid and e-Science infrastructures that deal with the so-called “Grid problem”: flexible, secure, coordinated resource sharing across dynamic, multi-institutional collaborations [27]. This includes highly flexible sharing relationships for sophisticated and precise levels of control over how shared resources are used. In this context, a VO typically makes agreements on sharing resources with resource owners that want to have control on how the sharing is done raising the demand for fine-grained and multi-stakeholder access control. These agreements are a crucial part of VO management that provides instruments to facilitate the enforcement of such agreements using VO-level attributes that are used for authorization. One of these instruments is the VOMS [2] that represents an Attribute Authority (AA), which allows users to be assigned with attributes holding their position in a VO. This includes group or project membership as well as role possession. These attributes are used at the resource level to enable fine-grained authorization. The other main AA within Grids is GridShib/Shibboleth [9], which provides a federated single sign-on and attribute exchange framework. GridShib is a software product that allows interoperability between the Globus Toolkit and Shibboleth, thus making the latter available for Grid authorization. GridShib project members are working on an OASIS standard for a deployment profile for X.509 subject to use with SAML 2.0 [43]. This profile complements the SAML specifications [13] with indications on how to use SAML with X.509 certificates. It is expected that GridShib and VOMS follow the same standard interface for message exchanges as well as assertions and thus what described in Section 5 should work not only with VOMS but also with GridShib, except the different policy definitions that are dependent from the correspondent attribute formats.

Relationship with OGSA The Open Grid Services Architecture (OGSA) [28] describes the requirements and the capabilities which are needed to support Grid systems and applications in both e-science and e-business. The following capabilities are considered in the OGSA architecture: Execution Management, Data, Resource Management, Security, Self-Management, and Information. The capabilities are mostly considered in isolation, and there is no requirement that they all be present in an OGSA-compliant system.

The OGSA Execution Management Service (EMS) is the service dealing with job submission and management. Its requirements are defined as follows:

- Support for various job types, including simple jobs or workflows;
- Ability to manage jobs during their entire lifetimes;
- Ability to schedule and execute jobs based on such information as specified priority and current allocation of resources;
- It must be possible to deploy the required applications and data to resources and configure them automatically.

These requirements are partially fulfilled by the BES and JSDL specifications. JSDL currently supports "simple" jobs only (workflows and structured job collections are currently out of scope from JSDL); BES allows the management of jobs during their lifetime, but no specific feature for handling priorities or automatically configure resources is provided.

The OGSA security requirements include the following:

- Authentication mechanisms are required so that the identity of individuals and services can be established;
- OGSA needs to integrate and interoperate with existing security architectures and models;
- Resources may have to be accessed across organizational boundaries. OGSA requires standard and secure mechanisms that can be deployed to protect organizations while also enabling cross-domain interaction;
- Various kinds of isolation must be ensured, such as isolation of users, performance isolation, and isolation between content offerings within the same Grid system;
- Mechanisms that allow for delegation of access rights from service requestors to service providers are required;
- Service requestors and providers should be able to exchange dynamically security policy information;
- Strong monitoring is required for intrusion detection and identification of misuses, malicious or otherwise.

The SAML specification mainly addresses the *authentication and authorization* requirements stated above. It is an XML-based framework which allows entities to make assertions regarding the identity, attributes, and entitlements of a user. SAML assertions are packages of information that supply one or more statements by a SAML authority. It also defines protocols to request assertions from SAML authorities.

Finally, the OGSA defines the requirements for Grid information services. The term information refers to dynamic data or events used for status monitoring; relatively static data used for discovery; and any data that is logged. An information service needs to support a variety of QoS requirements for reliability, security, and performance. The GLUE specification defines an information model and reference realizations for describing Grid resources. It should be observed that GLUE does *not* address the equally important problem of how to access and query an information system. Thus, we partially solved this problem by using one operation of the BES interface (specifically, the *GetFactoryAttributesDocument* operation) to expose an XML rendering of a GLUE 2.0 representation of the capabilities of the job execution service. More details are given in Sections 3 and 4.

3 Job Management

One of the most important functionalities offered by any Grid system is the possibility of submitting and managing jobs, which are executed on suitable computational resources. While the exact notion of *job* varies from Grid to Grid, there are many common features which can be identified. For example, a job usually consists of running some executable program on a given processor; the program may operate on one or more input data files, and produce one or more output data files. Moreover, job requirements (minimum available memory, disk space, CPU speed) may be part of the job description. The existence of those common features across different Grids was the motivation for the development of standards for job descriptions and management.

3.1 The JSDL Specification

The JSDL [8] is an XML-based notation for describing the requirements of computational jobs. The JSDL notation is defined by a normative XML Schema that facilitates the expression of those requirements as a set of XML elements.

JSDL provides a notation for describing the structure and requirements of individual jobs. Other, equally important, aspects are outside the scope of JSDL. For example, many Grid systems provide the notion of *structured job collections*; in the gLite framework, a Directed Acyclic Graph (DAG) can be used to represent workflows where multiple, independent jobs can be scheduled according to a set of user-defined inter-job dependencies. Most Grid systems have similar features; however, these are outside the scope of the JSDL specification.

A JSDL document has this general structure:

```
<JobDefinition>
  <JobDescription>
     <JobIdentification ... />?
     <Application ... />?
     <Resources ... />?
```

```
    <DataStaging ... />*
  </JobDescription>
  <xsd:any##other/>
</JobDefinition>
```

where:

`<JobIdentification>` contains a human-readable description of the job. It is a complex element which contains sub-elements for specific informations such as the job name, a (textual) job description, the name of the project the job belongs to, and so on.

`<Application>` contains a machine-readable description of the job. It includes the executable name and any parameter needed to run the job. Note that the Application element is optional; if missing, the JSDL document describes a null job.

`<Resources>` contains a description of the resource requirements for the job. Additional sub-elements can be used to specify bounds on, e.g., the required number of CPUs, free disk space, specific filesystem layout, available system memory and so on. Furthermore, the CPU architecture, operations system name and version for the execution host can be specified.

`<DataStaging>` defines the files that should be moved to the execution host (stage in) and from the execution host (stage out). Files are staged in before the job starts executing, and are staged out after the job terminates. The files which are staged out usually are meant to contain the result of the job execution.

While the JSDL specification is general enough to encompass the basic features of most Grids, there are many other features which are not present. The JSDL has an extension mechanism by means of which it is possible to add additional information:. The JSDL extension mechanism is implemented by allowing arbitrary XML elements (`<xsd:any##other/>`) to be added in specific position of the JSDL XML data structures, provided that the new XML elements have a different namespace than the JSDL ones. This extension mechanism has some drawbacks, as will be discussed in Section 7.

3.2 The BES Specification

The BES specification [24] describes a Web Service interface for creation, monitoring and control of computational jobs. In the BES terminology, jobs are called *activities*, and are described using the JSDL notation. While JSDL is used to describe the static structure of an activity, BES specifies a set of operations which can be executed on activities: creation, termination, obtaining the current status of an activity or a set of activities and so on.

In general, a BES service acts as a frontend to one or more *BES resources*, where a BES resource is a generic term used to denote anything from a supercomputer, to a pool of workstations managed through a batch system such as LSF, PBS or Torque, or individual computers. Multiple resources can be managed by one BES service; the submitted JSDL may contain a `<Resources>` element describing the requirements of the job. Those requirements are matched against the capabilities provided by the available resources, and one of those matching the requirements is selected for the execution of the activity. Note that the current BES specification does not include any specific way for accessing and managing individual (contained) BES resources, so different implementations provide different access mechanisms.

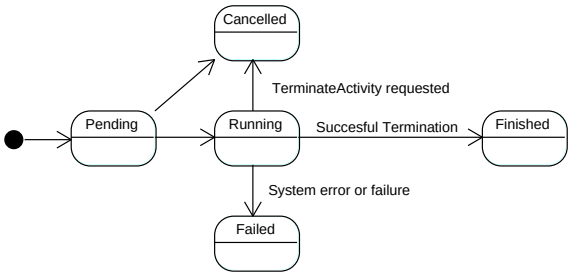


Fig. 2 BES basic state model

Technically speaking, the BES Web Services Description Language (WSDL) document defines two Web Service (WS) port-types, which are shown in Table 1 along with their corresponding operations.

The BES-Management port-type is used to control the BES service itself. This port-type contains two operations which are used to start and stop the service, respectively. This port-type should normally be used by the system administrators.

The BES specification mandates that the activities must be described using the JSDL specification. Activities are uniquely identified using WS-Addressing End Point Reference (EPR) [32]. The BES *CreateActivity* operation returns an EPR, which can be used by clients to refer to this activity. During execution, activities traverse a number of states. The basic state model comprises the following states (see Figure 2): (1) *pending*, the service has created the activity, but the latter is not yet running: (2) *running*, the activity is being executed on some computational resource; (3) *finished*, the activity successfully completed execution; this is a terminal state, (4) *terminated*, the activity has been terminated by calling the *TerminateActivities* operation; (5) *failed*, the activity has failed due to some error or failure (both *terminated* and *failed* are terminal states).

The state model can be extended, subject to some restrictions. It is possible to define sub-states of any valid BES state. but transitions between sub-states S_1 and S_2 are only possible if, either: (i) S_1 and S_2 are both sub-states of the same BES state, or (ii) S_1 is a sub-state of BES state X and

Table 1 BES Port-Types and Operations

BES-Management Port-type	
StartAcceptingNewActivities	Administrative operation: Requests that the BES service starts accepting new activities
StopAcceptingNewActivities	Administrative operation: Requests that the BES service stops accepting new activities
BES-Factory Port-type	
CreateActivity	Requests the creation of a new activity; in general, this operation performs the submission of a new computational job, which is immediately started
GetActivityStatuses	Requests the status of a set of activities
TerminateActivities	Requests that a set of activities be terminated
GetActivityDocuments	Requests the JSDL document for a set of activities
GetFactoryAttributesDocument	Requests the XML document containing the properties of this BES service

S_2 is a sub-state of BES state Y and $X \rightarrow Y$ is a valid transition according to the BES state model of Figure 2.

The BES-Factory port-type defines operations for creating and manipulating activities and set of activities. Moreover, it contains an operation (*GetFactoryAttributesDocument*) for retrieving informations about the BES service itself. Such information contains, among others, the human-readable service name, the total number and EPRs of all the activities in the service, and the number of contained BES resources. The *GetFactoryAttributesDocument* operation only returns a very simple description of the capabilities of the BES service. However, the BES specification (similarly to JSDL) provides standard extension mechanisms so that additional XML elements can be inserted in the normative BES data structures, provided that the new elements have a different XML namespace than the normative ones. Using this extension mechanism it is possible to encode a more complete description of the BES endpoint, using an XML rendering of the GLUE2 specification (we will discuss GLUE2 in Section 4). More specifically, the *GetFactoryAttributesDocument* returns a `<FactoryResourceAttributesDocument>` structure, which represents the current status of the resource represented by the BES endpoint itself (name, endpoint, operating system name, number of CPUs and so on):

```
<FactoryResourceAttributesDocument>
   <BasicResourceAttributesDocument ... />*
   <IsAcceptingNewActivities ... />
   <CommonName ... />?
   <LongDescription ... />?
   <TotalNumberOfActivities ... />
   <ActivityReference ... />*
   <TotalNumberOfContainedResources ... />
   <ContainedResource ... />*
   <NamingProfile ... />+
   <BESExtension ... />*
   <LocalResourceManagerType ... />
   <xsd:any namespace="##other" ... />*
</FactoryResourceAttributesDocument>
```

The `<ContainedResources>` might contain other `<FactoryResourceAttributesDocument>` or `<BasicResourceAttributesDocument>` elements, so that it is possible to recursively describe each contained BES resource.

As can be seen, a very limited set of informations can be exposed through the `<FactoryResourceAttributesDocument>` element. Using the BES extension mechanism, it is possible to enhance the resource description by using the XML rendering of the GLUE 2.0 information model [4]. The resulting XML block can be inserted in the placeholder `<xsd:any##other>` element.

3.3 Implementing BES/JSDL in CREAM

We implemented the BES specification in the CREAM Service. CREAM [1] is a Web Service based job submission and management service being developed for the gLite middleware [36]. We show in Figure 3 a high-level view of the main architectural components of CREAM.

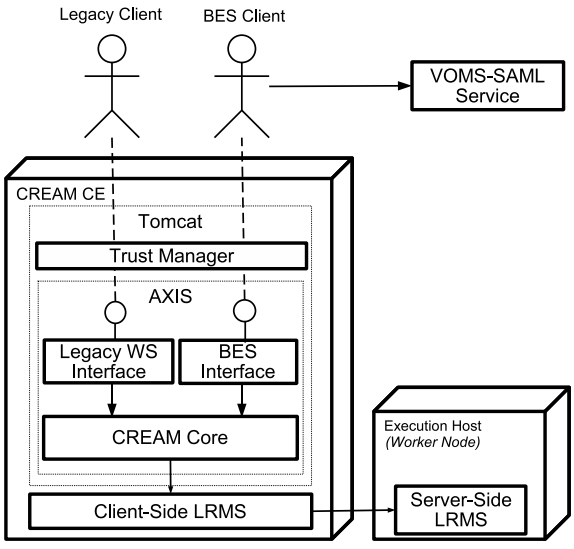


Fig. 3 High level structure of the CREAM service enhanced with BES interface

CREAM runs as a Java-Axis servlet [8] in the Tomcat application server [9]. Requests to CREAM traverse a pipeline of additional components, collectively called Trust Manager. The Trust Manager is responsible for carrying out authentication operations. It is external to CREAM, and is an implementation of the J2EE security specifications. CREAM can be logically split in two main parts: the interface(s) and the CREAM core. Note that CREAM was developed before standards such as BES/JSDL were available. For this reason, its legacy interface, while based on Web services, is not BES compliant. However, the service interface is decoupled from the core, do it was possible to support both BES and the legacy one interface, allowing BES and non-BES clients to access the service at the same time. The CREAM core is responsible for the actual processing of the requests, and for keeping the internal state up-to-date. The core interacts with the client-side Local Resource Management System (LRMS), which in turn interacts with the server-side LRMS. Thus, it is possible to have the CREAM service running on one host, and the batch system head node running on a separate host.

In general the CREAM service can be connected to one or more batch systems such as Torque of LSF. For this reason, it is easy to identify a BES resource with a specific queue of the available batch systems. More complex configuration are possible, for example defining some *shares* [5] over a LRMS, where a share is an abstraction of the resource partitioning among user. CREAM handles this kind of internal resource structure as a BES contained resource. Each contained resource publishes its own set of information and access policies; this information is a subset of the data stored into the information system of the Computing Element (CE). CREAM may allow the user to submit activities to the CE as a whole; in this case an internal mechanism, or selector, is used to forward the request to one suitable queue or share. Users can also access individual resources (batch queues) as follows. In CREAM a batch queue is uniquely identified by a pair of strings $(\text{batch system}, \text{queue name})$, hence, if the service URL for the BES interface is `https://my.host/path`, then each batch system/queue name will be made available as standard BES services with URL `https://my.host/path?b=<batch>&q=<queue>` (administrators can restrict or forbid access to individual queues). Other approaches for accessing contained resources exist, for example using the WS-Resource specification [31]. However, the one adopted in CREAM has the advantage of being easier to implement, and allows access to contained resources also to non-WSRF clients. In should be observed that the BES specification only mentions that compliant services *may* expose resource properties according to WS-Resource, but does not specify the details on how this is to be done.

Authentication in CREAM is based on a Public Key Infrastructure (PKI). Each user (and Grid service) wishing to access CREAM is required to present an X.509 certificate [33]. These certificates are issued by trusted entities, the Certificate Authorities (CA). The role of a CA is to guarantee the identity of a user. This is achieved by issuing an electronic document (the certificate) that contains the information about the user and is digitally signed by the CA with its private key. An authentication manager, such as the Trust Manager, can verify the user identity by decrypting the hash of the certificate with the CA public key. This ensures that the certificate was issued by that specific CA. The Trust Manager can then access the user data contained in the certificate and verify the user identity.

Note that the security mechanism used by the BES interface is slightly different than the one used by the legacy interface. Specifically, a BES client needs to insert a SAML assertion inside each request. To do so, for each request the client must contact a service capable of releasing SAML (signed) assertions. One of such components is VOMS-SAML, which will be described in Section 5.

4 Information Modeling

An important aspect to realize the discoverability of resources is sharing a common characterization of what the resources are, their properties and relationships. The characterization of these aspects are typically captured in information models, that are abstractions of real world entities into constructs that can be represented in computer systems. Different Grid middlewares are provided with their own information models to describe the properties of the exposed resources that should be advertised in order to enable resource selection (e.g., GLUE 2.0 [5], NorduGrid schema [35]). As specified in the OGSA [28], this information is made available via a Grid information service [19] and is therefore accessible by potential consumers. Accessing resource descriptions via the information service enables to achieve the resource awareness, that is, a state whereby one party has knowledge of the existence of the other part.

The plurality of information models are a barrier to interoperable Grid systems. In particular, we refer to the information interoperability, that is the ability to meaningfully exchange information among separately developed systems, including the understanding of the information format, meaning, and quality. For this reason, within the OGF, a Working Group started to work on

[8] Apache Software Foundation, Axis SOAP Container: http://ws.apache.org/axis/

[9] Apache Software Foundation Jakarta Tomcat Servlet Container, http://tomcat.apache.org/

the unification of the various information models into a community standard specification called GLUE 2.0. One document from this group addressing the conceptual model was published as proposed recommendation [5] while the reference realizations [4] to concrete data models are in advanced definition.

The GLUE 2.0 conceptual model is described in terms of UML class diagrams enriched with descriptive tables providing extra information about the UML elements (e.g., class name definition, properties definition and unit of measure). Three sub-models are present: the main entities sub-model which captures concepts such as AdminDomain, UserDomain, Service, Manager, Resource, and Endpoint; the computing entities sub-model which is a specialization of the main entities in the context of computing resources (typically batch systems or super computers); the storage entities sub-model which is a specialization of the main entities in the context of storage systems, ranging from simple disk servers to complex hierarchical storage systems.

4.1 The Main Entities

The main entities that are central to the GLUE Information model are the concepts of *UserDomain*, *AdminDomain*, *Service*, *Endpoint*, *Resource*, *Manager* and *Activity* (see Figure 4). A *UserDomain* is defined as a collection of actors that can be assigned with user roles and privileges to services or shares via policies. It is the concept used to model groups of users or entire virtual organizations having access to Grid services. On the other side, an *AdminDomain* is used to identify atomic management units responsible for a set of services. Services part of an administrative domain can span different physical locations. A *Service* is defined as an abstracted, logical view of actual software components that participate in the creation of an entity providing one or more functionalities useful in a Grid environment. The service is a concept introduced to identify the whole set of entities providing the functionality with a persistent name. A service aggregates the following entities: *Endpoint*, that is a network location having a well-defined interface and exposing the service functionalities, *Manager*, that is a software component locally managing one or more resources, *Resource* that is an entity providing a capability or capacity and managed by a local software component (i.e., the manager) and *Share* that is a utilization target for a set of resources managed by a local manager and offered via related endpoints. Finally, an *Activity* is a unit of work managed by a service. An activity can have relationships to other activities being managed by different services, therefore it shares a common context.

4.2 The Computing Entities

An important type of resources available in a Grid environment is related to the provision of a storage functionality (see Figure 5). The following entities have been identified as useful to be described: computing service, computing manager, computing share, execution environment, application environment and computing activity. The *Computing Service* is a specialization of a service focused on the computing functionality. The *Computing Endpoint* is a specialized endpoint for creating, monitoring, and controlling computational activities called jobs or computing activities. The *Execution Environment* provides a description of hardware and software characteristics that define a type of environment available to and requestable by a Grid job when submitted to a Computing Service via the Computing Endpoint. Such a description also includes information about the total/available/used instances of the execution environment. An Execution Environment may also contain one or more Application Environments. The *Computing Manager* is a grouping concept for a set of different types of execution environments; the aggregation is defined by the common management scope (e.g., a local resource management system like a batch system defines an aggregation scope). An important concept for a Computing Service is the *Computing Share*, which is a utilization target for a set of computing resources defined by policies and characterized by status information.

4.3 The Storage Entities

Another important type of resources available in a Grid environment is related to the provision of storage functionality (see Figure 6). The following entities have been identified as useful to be described: storage service, storage manager, storage share, storage resource and storage access protocol. The *Storage Service* is a specialization of a service focused on the storage functionality. The *Storage Endpoint* is a specialized endpoint for managing storage shares or for accessing them. The *Data Store* is a description of sufficiently homogeneous storage device providing a storage capacity. The *Storage Manager* is the primary software component locally managing one or more storage resources. The *Storage Share* is a specialization of a share used to describe utilization target on storage resources. The *Storage Access Protocol* is useful to describe the type of protocol available to access the available storage capacities.

4.4 GLUE Realization

The GLUE realizations document [4] describes the mapping of the conceptual model into three different concrete

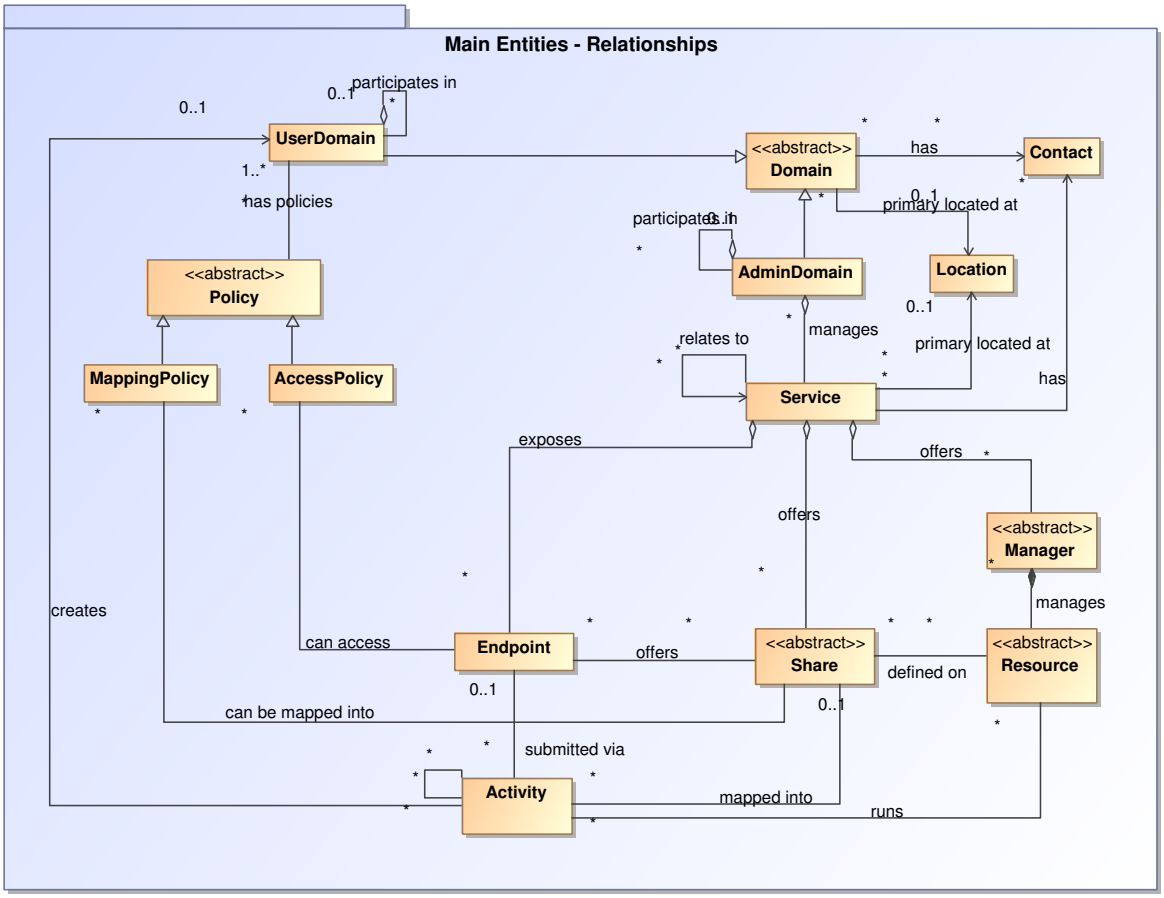


Fig. 4 GLUE 2.0 Information Model–Main Entities

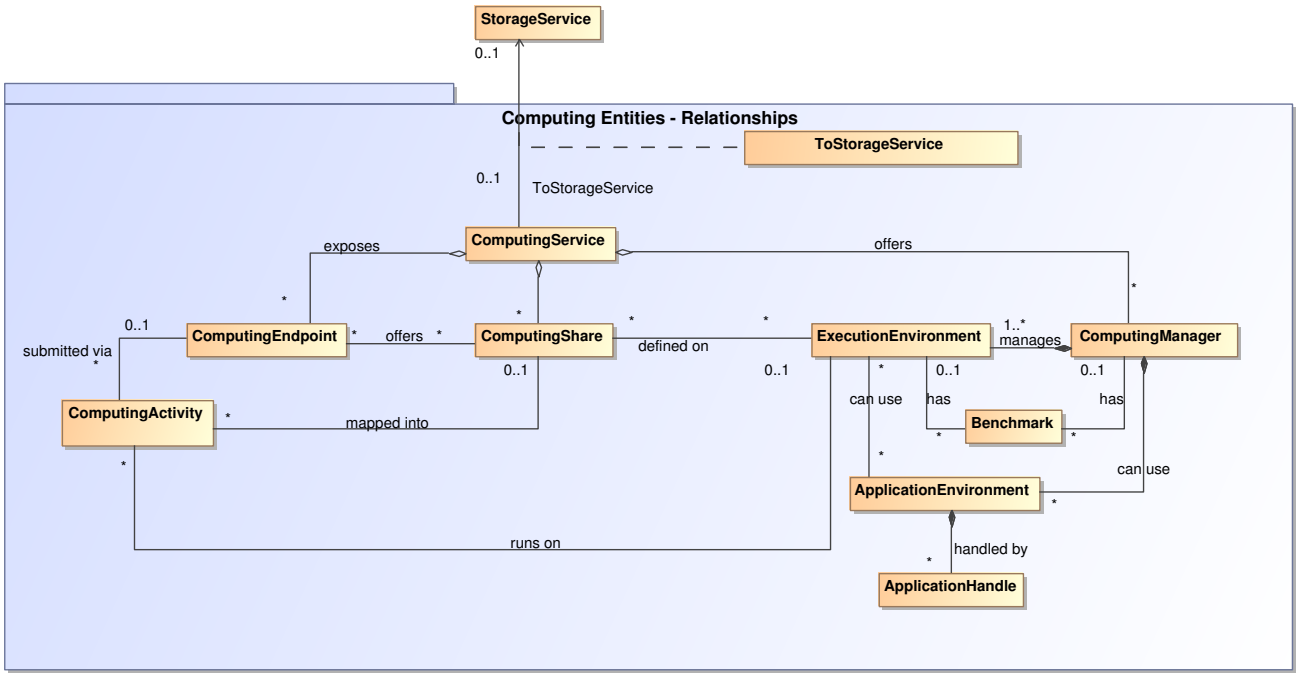


Fig. 5 GLUE 2.0 Information Model–Computing Entities

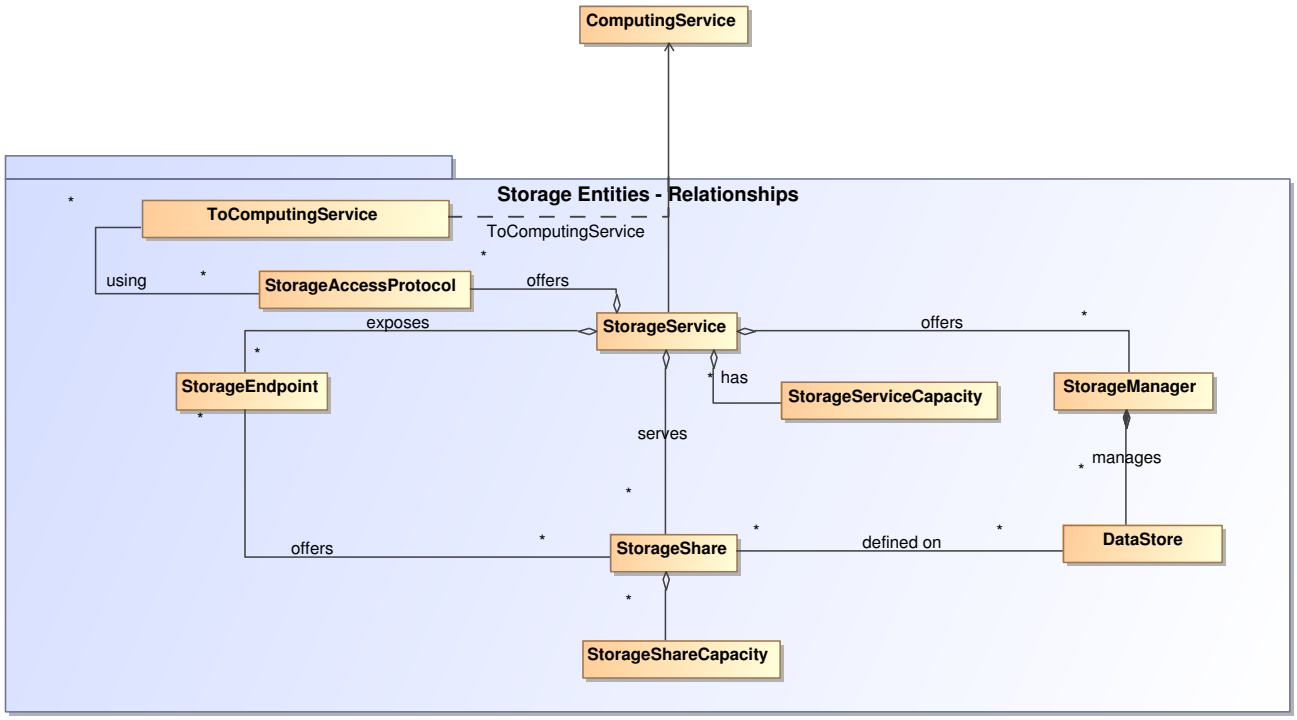


Fig. 6 GLUE 2.0 Information Model–Storage Entities

data models: XML Schema, LDAP and SQL. Such concrete data models are selected based on community requirements. The main motivation for the various mapping is that Grid information services relying on different concrete data models exist (e.g., the gLite information service is based on LDAP, while the Globus MDS information service is based on XML).

With the availability of the final GLUE specification and the related realizations, the various middleware developers aiming at its adoption need to instrument their software components with information providers that measure the properties defined in the schema and present the measured information according to the conceptual model definition and the related concrete data models realizations. The information from various sources need also to be aggregated. Information providers perform the measurement either automatically by interacting with other software components or by accessing configuration data which was written by system administrators.

As an example, this is a very simplified fragment of GLUEL2 representation of a BES computing endpoint:

```
<glue:Domains>
 <AdminDomain>
  <ID>urn:admindomain:infn:t1</ID>
  <Name>INFN-T1</Name>
  <Description>
   This is the Italian T1
   of EGEE Grid
  </Description>
  <WWW>http://www.cnaf.infn.it</WWW>
  <Owner>INFN</Owner>
  <Location>
    ...
  </Location>
  <Contact>
   <LocalID>
    mailto:t1-admin@cnaf.infn.it
   </LocalID>
   <URL>
    mailto:t1-admin@cnaf.infn.it
   </URL>
   <Type>general</Type>
   <OtherInfo>
    working hours: 8-18
   </OtherInfo>
  </Contact>
  <Services>
   <ComputingService>
    <ID>urn:infn:cnaf:bes</ID>
     <Name>CNAF Computing BES Endpoint</Name>
     <Capability>
       executionmanagement.jobexecution
     </Capability>
     <Type>org.glite.cream.bes</Type>
     <QualityLevel>testing</QualityLevel>
     ...
     <ComputingEndpoint>
       <ID>urn:infn:cnaf:cs:bes</ID>
       <Name>CREAM-BES</Name>
       <URL>
https://creambes.cnaf.infn.it:8443/ce-cream/services
       </URL>
       <Technology>webservice</Technology>
       <InterfaceName>OGSA-BES</InterfaceName>
```

```
        <InterfaceVersion>1.0</InterfaceVersion>
        <WSDL>http://someurl/ogsa-bes.wsdl</WSDL>
        <SupportedProfile>WS-I 1.0</SupportedProfile>
        <Semantics>
         http://www.ogf.org/documents/GFD.108.pdf
        </Semantics>
        ...
        <AccessPolicy>
         <LocalID>
          urn:infn:cream:bes:policy
         </LocalID>
         <Scheme>basic</Scheme>
         <Rule>VO:CMS</Rule>
         <Rule>VO:ATLAS</Rule>
        </AccessPolicy>
        ...
      </ComputingEndpoint>
    </ComputingService>
  </Services>
 </AdminDomain>
</glue:Domains>
```

4.5 GLUEMan

We now describe the design and implementation of GLUEMan [10][6], a framework based on WBEM[11] technologies aimed at simplifying the adoption process of the GLUE information model in existing Grid middlewares. GLUEMan enables the middleware developers to concentrate only on their essential role in the process, that is producing the information according to the schema in a simple and unique format. The framework takes care of aggregating the information, validating it against the normative specification, generating the realization in the various concrete data models and exposing it via a network accessible endpoint using WBEM standards.

In Figure 7 we see a simplified view of the GLUEMan components: a proxy provider and a client. The proxy provider is a component added to decouple the interaction of Open Pegasus[12] from the real provider while the client is added to decouple the interaction of the information consumer from the Open Pegasus server. Open Pegasus is an open-source implementation of the Distributed Management Task Force CIM [15] and WBEM standard designed to be portable and highly modular. It is coded in C++ so that it effectively translates the object concepts of the CIM objects into a programming model but still retains the speed and efficiency of a compiled language.

4.6 Proxy Providers Module

In Open Pegasus, each provider is related to a class or association, therefore it is responsible for generating a number of instances for a certain class/association definition. Open Pegasus offers a native API in C++, nevertheless it also supports the CMPI standard binary interface (Common Manageability Programming Interface) [18]. Writing a provider for a single class or association requires implementing several methods.

The main design pattern used to address the provider requirements is the proxy design pattern. For each provider related to the GLUE information model, we define a proxy provider decoupling the interaction between the Open Pegasus server and the real provider. The proxy provider offers three main extra-functionalities: 1) invocation and interaction with the real provider written in any language; 2) caching of the result (the expiration time can be configured for each individual provider); 3) conformance check to the GLUE 2.0 specification.

The communication between the real provider and the proxy provider is performed via the standard output. The selected format for exchanging data is the INI format [34]. The motivations for this choice are: (1) the INI format is simple; (2) there are many parsers in all relevant languages to handle this format (if no existing parser can be reused, it is simple to write a new one); (3) the complexity of the output is suitable for the INI format (list of instances of classes/associations); (4) the validation of the data is performed by the proxy provider.

All the proxy providers are packaged in a single software component written in C++ and compiled as a shared library which exposes a CMPI (Common Manageability Programming Interface) interface to the Open Pegasus server. This library contains as many providers as there are classes and associations defined in GLUE 2.0.

When a query for a certain class is sent to the Open Pegasus server, this invokes the related proxy provider. The proxy provider reads a configuration file where the relevant parameters describing the real provider are stored. Among these parameters, the full path name of the real provider and the expiration time for the cache are available. If a previous output was created in the cache validity timeframe, then the real provider is not invoked and the previous output is used to create the instances of GLUE 2.0 information within the Open Pegasus server. On the other side, if no valid output was present, the real provider is invoked and the output is consumed. A configurable time out per each provider is available in order to deal with real providers who get stuck.

4.7 Client

The client design is based on two main design patterns: the Model-View-Controller pattern and the Strategy pattern. As regards the Model-View-Controller: the Model component is represented by CIM Classes and Instances rendered in the

[10] http://glueman.sf.net/
[11] http://www.dmtf.org/standards/wbem/
[12] http://openpegasus.org/

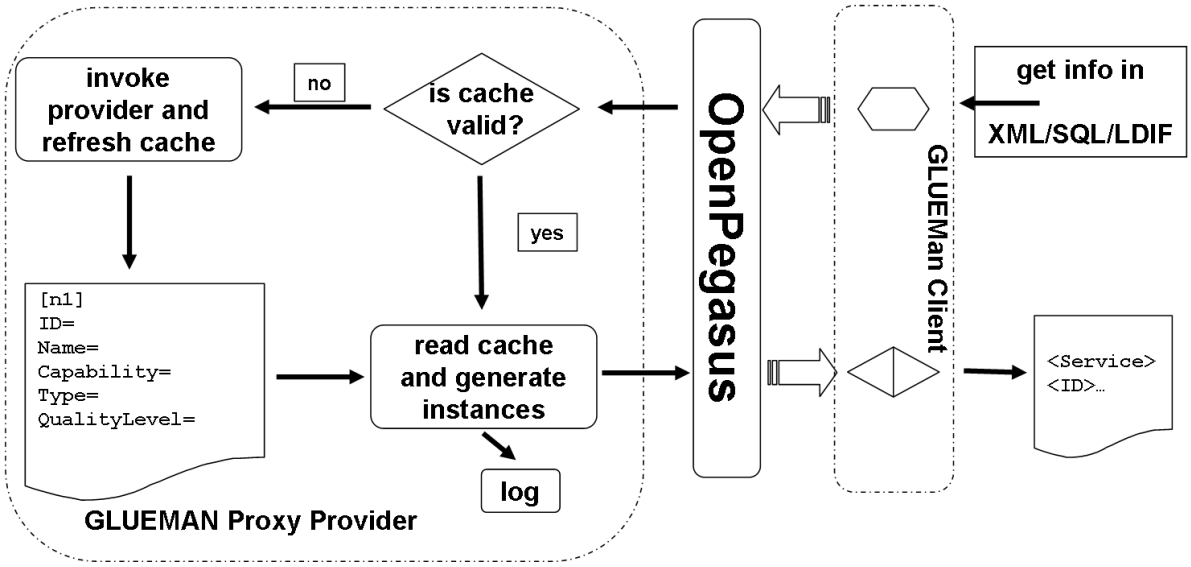


Fig. 7 GLUEMan: Simplified Functional View

Managed Object Format (MOF) (a textual representation of UML class diagrams), the View component is represented by the GLUE 2.0 realizations (XML Schema, LDAP and SQL), and the Controller component is represented by the navigation strategy through the GLUE instances. The Strategy design pattern is used in order to handle the different realizations of the data acquired by the Open Pegasus server. This pattern enables to isolate them and to easy the addition of new realizations.

The client is provided as a command line tool and not in the form of an API library for specific programming language. Programs which want to use it, have to invoke the client via a system call and, depending on the command line arguments, can direct the information in a file or to the standard output.

The main goal of the client is to provide a simple way to access the GLUE-based representation of the services provided by a certain computing environment without requiring knowledge of the CIM over HTTP protocol. With this solution, we can satisfy two categories of information consumers: those interested in the GLUE-based information of Grid resources regardless how this is produced and aggregated and the management tools based on WBEM standards which want to access the same information using the CIM over HTTP protocol.

4.8 Integration of GLUEMan in Grid Middleware

GLUEMan can be adopted as a back-end for managing information providers and expose the measured data in different formats to the interested consumers in a certain execution environment. In Figure 8, we exemplify a general deployment strategy where a number of information providers (IP) have been written to interact either with the Grid middleware or with the underlying resource in order to extract the properties captured in the GLUE information model. The proxy provider (PP) decouples the interaction of the Open Pegasus server with the real providers.

The GLUEMan client can be used by different consumers to gather the information in different formats in order to be exposed to the higher level services. In the given figure, three different consumers use the client: a WS interface which exposes the Grid functionality prefers the XML realization since this is the ubiquitous data format in the WS-* technology stack; the LDAP-based information service requires the information in the LDAP Interchange Data Format (LDIF) [29], finally, the R-GMA service (Relational Grid Monitoring Architecture) [23] requires the information as SQL statements. To be noticed that by this solution, we add a native management interface based on standard to access the GLUE-based information by management clients.

5 Authentication and Authorization

The core problem in Grids is enabling coordinated resource sharing among dynamic collections of individuals, institutions, and resources, what are referred to as VOs [27]. In a VO, a varying number of participants with various degrees of prior relationships, join in order to share resources. Resource sharing is conditional: Resource Providers (RPs) make resources available subject to a number of constraints on who can use them, when, and for what reasons. Such constraints are agreed between RPs and VOs. Authorization plays thus a central role in enabling Virtual Organizations.

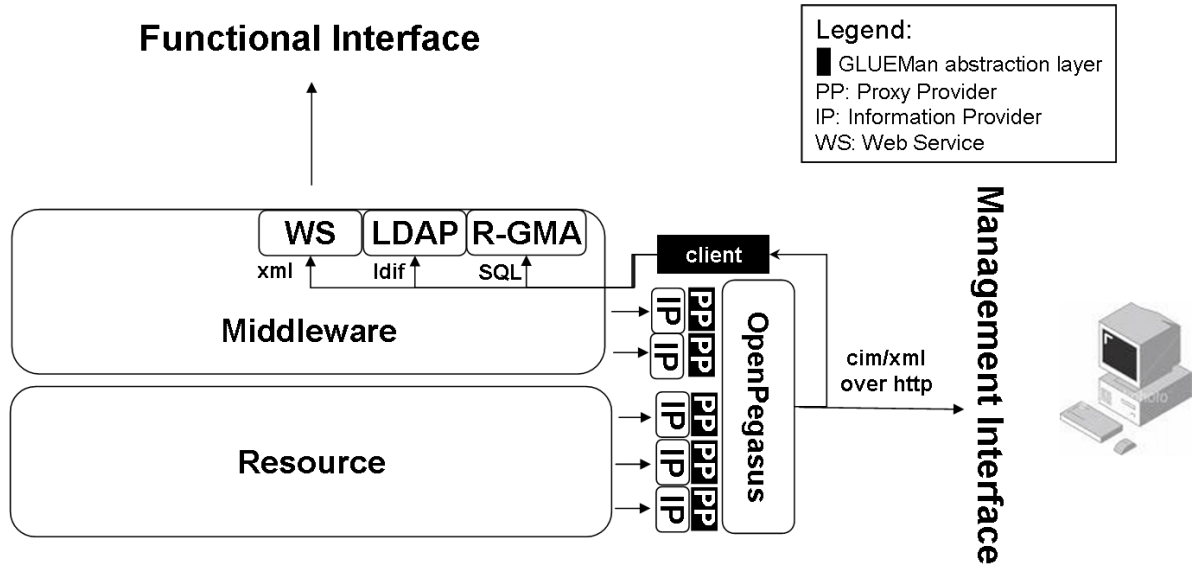


Fig. 8 Integration of GLUEMan in Grid Middleware

Essential in VOs is the ability to establish sharing relationships among *any* potential participants, independent of the nature of the resources and middlewares. Ensuring integrity, confidentiality and interoperability between heterogeneous systems can be achieved using a Web Service Architecture, which is an incarnation of a Service Oriented Architecture (SOA) in the context of the World Wide Web. SOA is the leading architectural style of the newly developed Grid technologies. Therefore, in order to achieve cross-Grid interoperability, the scientific community defines and implements standard interfaces for common services in the light of a SOA context.

Using an established set of standard interfaces, RPs should be able to share their resources between different VOs, even when running different middlewares from different vendors. For that to be possible, however, it is necessary that resources, besides common interfaces, implements common authorization mechanisms. In fig. 9 we provide an overview of the theoretical relationships that take place inside a VO between users, RPs, resources and middlewares.

In order to achieve interoperability among different middlewares, we must use a VO management tool, capable of arranging users in a VO and therefore simplifying the authorization procedures. The tool we are envisioning is the VOMS [2].

On one side, VOMS has been in use for several years in production Grids and is well integrated with gLite [36] and Globus[13]. We consider VOMS as a de-facto standard service for Grid authorization systems, but in order to reach

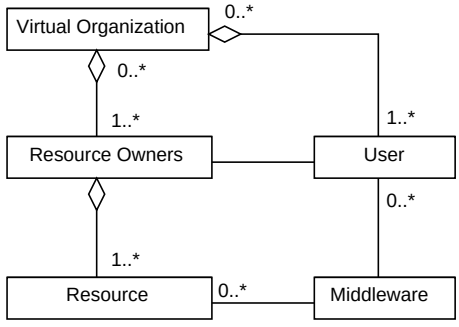


Fig. 9 Relationships in a Virtual Organization

complete interoperability it must expose a standardized WS interface.

On the other side, the SAML [13] is an OASIS-standardized XML language capable to release assertions regarding authentication, attributes and authorization. The emphasis is therefore on the re-engineering of VOMS, in order for it to be capable to expose a SAML interface. To reach our scope, we adapted the way SAML releases assertions to comply with the VOMS way of releasing attributes to Grid users.

5.1 The OASIS Security Assertion Markup Language

The aim of the OGSA authorization working group[14] is to define the specifications needed to allow for interoperability and pluggability of authorization components from multiple authorization domains in the OGSA framework. The

[13] http://dev.globus.org/wiki/Incubator/VOMS

[14] https://forge.gridforum.org/sf/projects/ogsa-authz

group leverages authorization work that is ongoing in the WS community (e.g. SAML, XACML [37], and the WS-Security [40] set of specifications) and defines profiles on how these should be used by Grid services. The group has identified three functional components to be used in Grid authorization and is currently working on defining profiles for them:

- the **Attribute Authority** is a service that releases assertions regarding user attributes. The protocol being defined uses SAML;
- the **Authorization Service** releases authorization decisions based on user and resource attributes. The protocol being defined uses XACML and the SAML profile for eXtensible Access Control Markup Language (XACML);
- the **Credential Validation Service** validates user credentials (digitally signed attributes assertions) to be used by an authorization service. The protocol being defined uses SAML and WS-Trust [38].

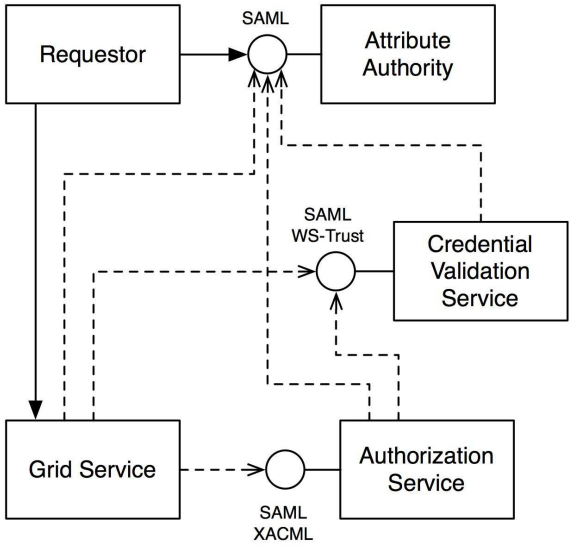


Fig. 10 Functional components for a Grid authorization service

The interactions between the functional components of OGSA Authorization are shown in Figure 10. In such a vision, an AA (like VOMS) should be able to release SAML assertions. Such assertions could be validated by an external Credential Validation Service, and then used as input by an Authorization Service.

The SAML is developed by the Security Services Technical Committee of OASIS. It is an XML-based framework that allows business entities to make assertions regarding the identity, attributes, and entitlements of a subject (an entity that is often a human user) to other entities, such as a partner company or another enterprise application.

SAML defines *Assertions*, packages of information that supply one or more statements by a SAML authority, among which are *Attribute Assertions*. It also defines protocols to request *Assertions* from SAML authorities [13], and bindings into standards messaging or communication protocols [12].

5.2 The OASIS Web Service Security

The Web Services Security (WSS) specification [40] defines a set of SOAP extensions that can be used when building secure WSs to implement message content integrity and confidentiality. It also provides a mechanism to send security tokens as part of a SOAP message. Additional profiles define the usage of this mechanism with different security tokens, including SAML [39].

The gLite VOMS service is an AA focused on VO Management. It releases signed assertions containing attributes expressing a user membership and position in a VO. Such assertions are used by Grid Services to drive authorization decisions, thus enabling the fine grained access control needed in Grids. VOMS has been re-engineered to support authorization standards emerging from the OGF. VOMS is widely used in the Grid community, thus the aim of our effort was to retain the functionalities of the current service, and extend it with a standard WS interface that uses SAML. Besides the protocol, the new service uses SAML Assertions to contain the subjects' attributes. The service is not meant to be a replacement of the legacy one, but aims at making the VOMS framework supporting the wider possible range of use patterns. A driving use case has been those Grid middlewares not using proxy certificates [46]. The main interactions between a client and the re-engineered VOMS service are shown in Figure 11.

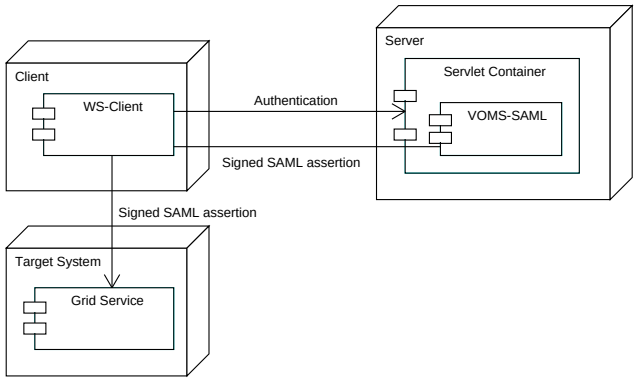


Fig. 11 Interactions between a client, a re-engineered VOMS service and a target system

5.3 Service Interface

The VOMS SAML service exposes an interface according to SAML protocols [13] and bindings [12]. The service supports a single operation, whose input is a `<samlp:AttributeQuery>` element and the output is a `<samlp:Response>` element. The `<samlp:AttributeQuery>` element contains the subject for which the requestor wants to retrieve attributes, and eventually which attributes she is interested in. A successful `<samlp:Response>` contains a `<saml:Assertion>` element with the requested attributes. The elements `<samlp:AttributeQuery>` and `<saml:Assertion>` are used according to the SAML profile for X.509 subjects [43].

When the service authorizes a request, a `<samlp:Response>` is used to return a `<saml:Assertion>` containing a `<saml:AttributeStatement>` with the subject's attributes. In the following, we sketch an example of a SAML assertion (some XML tags and attributes are omitted for brevity). Following the VOMS logic, an assertion must identify the VOMS server that released it, the entity (normally a user) whose assertion is addressed, and the VOMS attributes.

```
<saml:Assertion ... >
  <saml:Issuer Format="urn:..:x509SubjectName">
    CN=omii002.cnaf.infn.it,L=...
  </saml:Issuer>
  <saml:Subject>
    <saml:NameID Format="urn:..:x509SubjectName">
      CN=Valerio Venturi, OU=...
    </saml:NameID>
  </saml:Subject>
  <saml:AttributeStatement>
    ...
  </saml:AttributeStatement>
</saml:Assertion>
```

5.4 Expressing the VOMS attributes using SAML

The core functionality of VOMS is expressing attributes, but the Fully Qualified Attribute Name (FQAN) and Tag attributes don't map naturally to SAML, thus we need to use the following `<saml:Attribute>` elements: *vo*, *group* and *role* for the FQAN, and a fourth *tag* attribute.

Concerning the `<saml:AttributeVaue>` containing the VOMS attributes, we defined a new `complexType` type, the `FQANType`, in order to carry the priority attribute of *group* and *role*. Such a type is simply an extension of the `<xs:token>` type, which is itself a built-in type over `<xs:string>` that represents a tokenized string in the W3C recommendation of XML [44,10]. The following XML schema fragment defines the FQANType complex type:

```
<complexType name="FQANType">
  <simpleContent>
    <extension base="xs:token">
      <attribute name="priority"
      type="xs:positiveInteger"/>
    </extension>
  </simpleContent>
</complexType>
```

Similarly, we defined two new `complexType` types to carry a *tag* attribute. The `TAGType` type contains a sequence of `TAGValue` types, which are extension of the `<xs:string>` XML type. Both the XML schemas follow:

```
<complexType name="TAGType">
  <attribute name="TAGname"
    type="xs:string" use="required"/>
  <attribute name="TAGdescription"
    type="xs:string" use="optional"/>
  <sequence>
    <element ref="TAGValue"
      minoccours="1"
      maxoccours="unbounded"/>
  </sequence>
</complexType>

<complexType name="TAGValue">
  <simpleContent>
    <extension base="xs:string">
      <attribute name="qualifier"
        type="xs:normalizedString"
        use="required"/>
    </extension>
  </simpleContent>
</complexType>
```

The qualifier attribute of `TagValue` may be empty, indicating that its content should be assigned directly to the user.

At the time of writing, we are discussing with other VO management tools implementors a common SAML Attribute profile, that will define a format for SAML Attributes of interest for VOs. Given that, the format described above is likely to change in the future.

5.5 Sending SAML Assertions to Grid Services

In the Attribute Certificate (AC) based VOMS, there are command line tools that allow an AC retrieved from a VOMS AA to be embedded in a proxy certificate: over the years, this has proved a very convenient way of sending AC to Grid Services. Some tools using SAML Assertions are using the same proxy-based logic. For example, the Globus Toolkit 4[15] Community Authorization Service (CAS) [26] binds authorization Assertions to a proxy. GridShib [9] does the same with authentication Assertions. As described in section 5.2, the Web Service Security specification defines a way to send SAML security tokens as part of the SOAP Header. We have preferred this solution, being based on an already consolidated standard and allowing support

[15] http://www.globus.org/toolkit

for services not supporting the use of proxy certificates for authentication. Following is an example of a SOAP message carrying a SAML Assertion (omitted for brevity):

```
<soap:Envelope xmlns:soap="...">
 <soap:Header>
  <wsse:Security wsse="...">
   <saml:Assertion xmlns:saml="...">
    ...
   </saml:Assertion xmlns:saml="...">
  </wsse:Security>
 </soap:Header>
 <soap:Body>
  ...
 </soap:Body>
</soap:Envelope>
```

6 Putting the Components Together

We now describe how the standards and the components introduced in the previous sections can be used as building blocks for implementing a job submission and management service for Grids. The architecture is shown in Figure 12 (shaded components are those we actually implemented).

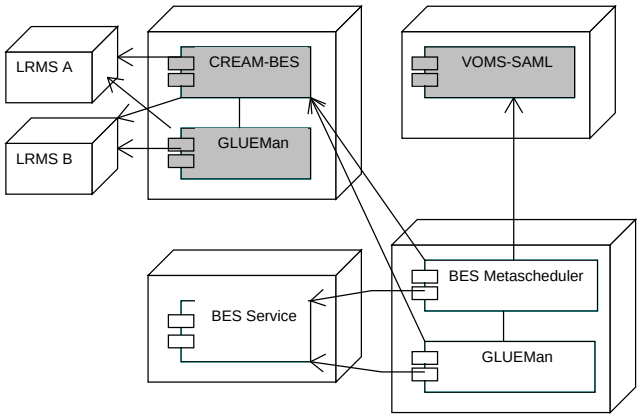


Fig. 12 Architecture for a standards-based job submission and management service. Shaded components have actually been implemented

We start by considering the BES/JSDL compliant job submission services. The diagram contains a CREAM service (only the BES interface is shown); as already described in Section 3, CREAM provides the following features:

- it implements the BES interface together with the (mandatory) support for the JSDL specification;
- it manages different contained resources (LRMSs, i.e. batch queues). As an example, in Figure 12 the CREAM service is associated with two batch systems, denoted as "LRMS A" and "LRMS B".
- it provides a mechanism for retrieving information about the status of the contained resources by means of an information provider.

The *GetFactoryAttributesDocument* BES operation can be extended to return an XML GLUE 2.0 representation of the current status of the execution service, together with the basic resource model described in the BES specification (refer to Section 3.2). In order to expose a GLUE 2.0 compliant model, BES services need to interact with a GLUE 2.0 information provider. As a concrete example, CREAM interacts with an external source of information in a pluggable way, so it is possible for the administrator to select the suitable extension for the given information system (BDII, R-GMA, GLUEMan). GLUEMan collects information directly from the resources available to the BES service (the LRMS). CREAM periodically contacts GLUEMan to get the up-to-date GLUE 2.0 representation of the resources, and provides this representation to clients by means of the *GetFactoryAttributesDocument* operation. In our implementation, GLUEMan can not be accessed directly by clients, but only through CREAM. The reason is that GLUE 2.0 is a data model, not an interface for accessing resource data.

The architecture shown in Figure 12 also includes a BES-compliant Grid metascheduler, which is a component that accepts JSDL activities from clients and forwards them to another BES endpoint for execution (note that the other endpoint might be itself a metascheduler). Since the metascheduler exposes an ordinary BES interface, it must provide informations on the contained resources as ordinary BES services do. Similarly to CREAM, metaschedulers can be enhanced to contact a local GLUEMan component. In this case, the local GLUEMan periodically fetches and aggregates GLUE 2.0 information provided by the BES endpoints the metascheduler is connected to, by invoking their *GetFactoryAttributesDocument* operation. The GLUEMan associated with the metascheduler acts as an aggregator for the data gathered from the information providers of the other BES endpoints.

We observe that the metascheduler must interacts also with the VOMS-SAML service (ordinary BES services do not need that). This is due to the particular role of the metascheduler, as it must forward user's requests to other job management services. With this respect, it acts as a client for the destination job management service. So, the metascheduler must obtain suitable credentials from the VOMS-SAML service, put those credentials in the request header as SAML assertions and send the request to the destination BES service, as every ordinary BES client would do.

We now summarize how clients interact with the CREAM-BES, VOMS-SAML and GLUEMan services for submitting and managing their activities. The UML Sequence Diagram shown in Figure 13 illustrates the interaction between a client and a plain BES service (not a BES metascheduler), such as CREAM-BES. Before

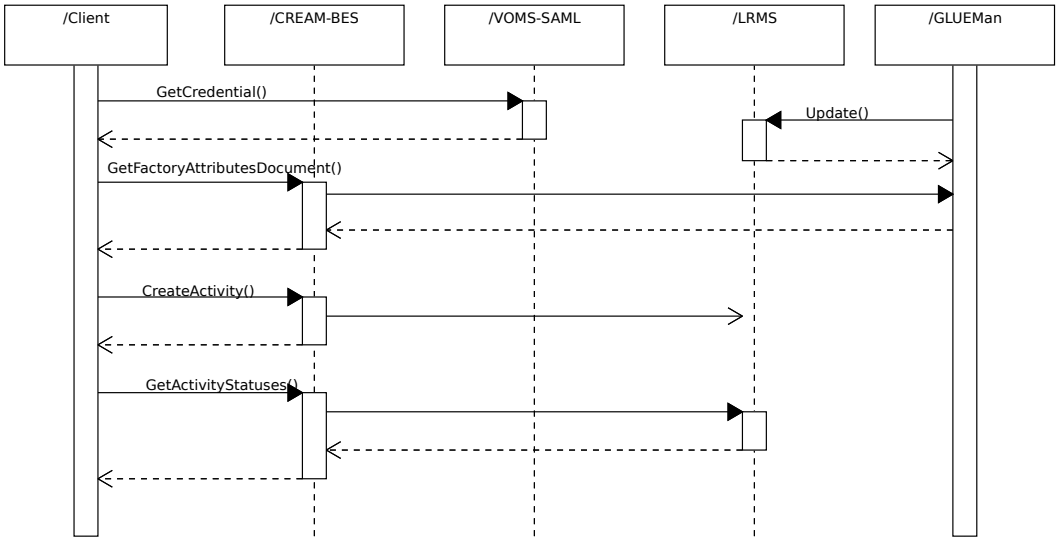


Fig. 13 Interactions between a client and the different components of a standards-based job management service.

invoking any of the BES operations, the user must get a signed SAML assertion from the VOMS-SAML service. The returned assertion remains valid for a period of time (the duration of an assertion is normally defined by the SAML service), and contains information about the capabilities of the user. The SAML assertion must be used to invoke any of the BES operations: the target service is able to verify the authenticity of the assertion by using the certificates of the Certification Authority (CA) and the VO the user belongs to. According to the capabilities contained in the assertion, the target service is able to allow or deny the user to perform a specific action.

For job management operations (*CreateActivity*, *TerminateActivities*, *GetActivityStatuses*) CREAM-BES forwards the request to the underlying LRMS, and sends back to the client the result. For the *GetActivityDocuments* operation, CREAM-BES fetches the JSDL used to instantiate the activity from its internal data store. For the *GetFactoryAttributesDocument* operation, the request is forwarded to GLUEMan, which returns an XML rendering of the GLUE 2.0 model representing the current status of the BES service; such XML rendering is then sent to the client. Note that, according to the GLUE 2.0 model, the status of an execution service includes the list of activities currently running on the service. Thus, GLUEMan needs to periodically poll the LRMS in order to get the number and ID of all non-terminated activities.

7 Discussion and lessons learned

So far we have discussed how the adoption of open standards helps to achieve interoperability between different middlewares: Grids can be assembled by composition of interoperable building blocks.

In order to validate these specifications and assess their completeness and usability, we have implemented them as described in the previous sections. In this way we have been able to verify whether the interfaces provided all the needed functionalities with the appropriate level of performance and reliability. Missing features may suggest the existence of use cases which are not supported by the specifications. It is important to consider that the BES/JSDL, GLUE 2.0 and SAML specifications have been developed independently from each other. Building a standards-based job management service out of them can provide important hints whether they can be fitted together, or whether important links are missing.

Job Description and Management As described in Section 3, we implemented the BES and JSDL specifications in the gLite CREAM execution service. The legacy interface of CREAM uses Condor classads [41] as the job description notation. Condor classad allows description of simple jobs or structured job collections (workflows, parametric jobs or parallel jobs using MPI). The CREAM service supports simple jobs only, as in the gLite middleware structured job collections are handled by the Workload Management System (WMS) [3]. Thus, adding support for JSDL in CREAM was easy because the service already supported all the features which can be described with the JSDL notation. CREAM translates both classad and JSDL job descriptions into an internal representation used by the CREAM Java core for further processing.

The lack of support for structured job collections can be a serious limitation for more sophisticated execution services, such as the gLite WMS. The WMS supports workflows by taking a single classad describing all jobs and their dependencies, and scheduling each job for execution

on a–possibly different–execution service. In its current form, JSDL is not appropriate for being used in components like the WMS.

An issue we encountered is that the BES specification requires conformant services to reject activities whose JSDL contains elements which are not supported by the receiving entity. This raises important interoperability problems: a client has no way of knowing in advance which JSDL elements are supported by a BES service. Our experience shows that any non-trivial JSDL has significant probabilities of being rejected by most of the BES services it is submitted to, because each one would complain on a different unsupported element. Those services actually accepting the request might be silently ignoring JSDL elements, which is not the standard behavior. The extension mechanism used by the JSDL specification makes things worse. It is not possible to insert service-specific hints (which might be safely ignored by other services) into a JSDL, because BES implementations not recognizing the extension elements will reject the request. This problem could be addressed in future revisions of JSDL by allowing clients to mark elements of the submitted request as "optional", so that the receiving service can safely ignore them.

Information Modeling As observed in Section 6, the standard BES specification lacks an adequate information model. As GLUE 2.0 is becoming an official OGF standard, this limitation will hopefully be fixed with the adoption of the GLUE 2.0 XML rendering of resources. This solves the problem of *representing* information about the BES service and its contained resources. However, the problem of *querying* the information model is still open. Letting the server return the whole GLUE 2.0 representation of all its contained resources can potentially generate a large amount of data which needs to be transferred to the client. However, clients will likely be interested in only a tiny fraction of the data. This suggests the idea of letting the client pass to the BES services a query written in some appropriate notation (e.g., XPath or XQuery). The server applies the query to the XML rendering of the GLUE 2.0 information model, and only returns the matching fragments. Of course, performance as well as security considerations must be carefully taken into account: as an example, remember that the full GLUE 2.0 resource model for a compute element include details about all running jobs; however, each user must be prevented from getting information about activities submitted by other users.

Resource Selection In the resource selection context, a related issue is the specification of JSDL requirements with respect to the GLUE 2.0 information model. Currently, activity requirements are specified using a simple notation involving range predicates over some basic attributes; all the requirements for an activity are specified inside a `<Resources>` JSDL element (see Section 3.1). For example, an activity can request an execution host with a given lower bound on the number of processors, a lower bound on the amount of available physical memory, and so forth. In order to take advantage of a more detailed information model, we need a better notation for specifying requirements on a GLUE 2.0 resource representation.

Security Aspects Security considerations are outside the scope of the BES specification; unfortunately, this raises the issue that it is possible to claim BES compliance without actually being interoperable, due to the use of incompatible security infrastructures. To mitigate this problem, several *profiles* have been proposed. A profile complements BES/JSDL by defining a set of specifications and/or extensions which must be supported by all conforming implementations. The HPC-Basic Profile [20] mandates a security mechanism based on TLS/SSL using X.509 certificates [46], or username/password embedded in the SOAP header. These mechanisms are not appropriate for production use: some middlewares (e.g., gLite) require clients to delegate credentials to the execution service, so that the delegate can execute operations (e.g., access remote files) on behalf of the delegator. In gLite, the legacy CREAM interface exposes a separate delegation port-type which must be used by clients to delegate their credentials. Note that SAML assertions can not help here, because assertions are bound to the entity which originally requested them and can not be securely transferred to third parties unless a proper delegation mechanism is used.

Web Services technologies Web Service technologies, while increasingly common, are becoming quite complex to implement properly. Tooling support for WSDL or XML schemas have bugs or incomplete support for the underlying specification, which sometimes result in hard-to-debug issues. Full interoperability between services based around different tools (e.g., Axis2 vs. XFire) is not always guaranteed without workarounds. Bugs in the tools supporting WS specifications are *accidental*, rather than *essential* [11]. Software bugs are likely to disappear as the tools mature. However, the intrinsic (*essential*) complexity of WS technologies can be addressed by employing alternative way of accessing the services. For example, in [7] the authors propose a way to access a BES interface using plain HTTP instead of SOAP.

Specifications developed in isolation As stated in Section 2, the OGSA defines a Grid architecture in term of independent services, which can be considered in isolation. This approach allows a substantial simplification of the architecture, by breaking down a complex system into simpler

pieces which can be analyzed independently. However, care must be taken to ensure that the individual components are linked to each other [42]. We highlighted the links between job management, security and information modeling in Section 6. Unfortunately, it turned out that an appropriate integration of the three components was more difficult than expected. For example, there is a clear dependency between execution services, security and data transfer, which is not properly handled with the existing specifications (refer to the discussion above on the lack of credential delegation in BES to enable secure, server-initiated data staging). Another missing link is between execution services and information systems: the BES defines a simple mechanism for representing resource information, but support of more advanced information models (such as GLUE 2.0) is possible only through extensions which have not yet been standardized.

Conclusions

In this paper we described three standards which are relevant to job execution and management across Grid middlewares: BES/JSDL, SAML and GLUE 2.0. We illustrated how actual software components have been developed to support these standards. Specifically, the CREAM and VOMS components from the gLite middleware have been re-engineered to support BES/JSDL and SAML, respectively. The GLUEMan information provider supporting GLUE 2.0 has been developed from scratch. We proposed a general Grid architecture for job management which is based around these specifications. A prototype implementation has been made, integrating CREAM-BES with VOMS-SAML and GLUEMan in the context of the Open Middleware Infrastructure Institute for Europe (OMII-Europe) project [16]

Interoperability through open standards is increasingly being considered as a way to allow efficient resource sharing among Grids. Furthermore, interoperability allows sharing code (software components) between middlewares, reducing development costs. Given that the source software of most production Grids is available through permissive open source licenses, this is a clear advantage.

Unfortunately, Grid standards are only recently emerging: most of them offer limited capabilities, and are being developed in isolation without taking into appropriate consideration their inter-dependencies. As a result, those standards are only slowly being adopted by Grid middlewares, and in any case they are not used in production systems, where legacy interfaces are being preferred. Nevertheless, we believe that all the limitations which we observed can and will be addressed as the standards mature and further experience in their usage is gained.

[16] http://www.omii-europe.org/

Acknowledgements This work has been supported by the European Union via the OMII-Europe project (EU Project Number INFSO-RI-031844).

References

1. Aiftimiei, C., Andreetto, P., Bertocco, S., Cesini, D., Corvo, M., Dalla Fina, S., Da Ronco, S., Dongiovanni, D., Dorigo, A., Gianelle, A., Grandi, C., Marzolla, M., Mazzucato, M., Miccio, V., Sciaba', A., Sgaravatto, M., Verlato, M., Zangrando, L.: Job submission and management through web services: the experience with the CREAM service. Journal of Physics, Conference Series **119**(6), 062,004 (2008). DOI 10.1088/1742-6596/119/6/06200
2. Alfieri, R., Cecchini, R., Ciaschini, V., dell'Agnello, L., Frohner, Á., Lőentey, K., Spataro, F.: From gridmap-file to VOMS: managing authorization in a Grid environment. Future Generation Computer Systems **21**(4), 549–558 (2005). DOI 10.1016/j.future.2004.10.006
3. Andreetto, P., et al.: The gLite Workload Management System. Journal of Physics, Conference Series **119**(6), 062,007 (2008). DOI 10.1088/1742-6596/119/6/062007
4. Andreozzi, S., Burke, S., Ehm, F., Field, L., Galang, G., Konya, B., Litmaath, M., Millar, P., Navarro, J.: GLUE 2.0–Reference Realizations to Concrete Data Models (2008). URL http://forge.ogf.org/sf/docman/do/listDocuments/projects.glue-wg/docman.root.public_comment. OGF Proposed Recommendation in Public Comment
5. Andreozzi, S., Burke, S., Ehm, F., Field, L., Galang, G., Konya, B., Litmaath, M., Millar, P., Navarro, J.: GLUE 2.0 Specification. OGF Specification GFD-R-P.147 (2008). URL http://www.ogf.org/documents/GFD.147.pdf
6. Andreozzi, S., Canaparo, M., Carpene, M.: GLUEMan: a WBEM-based Framework for Information Providers in Grid Services. In: International Conference on Enterprise Distributed Object Computing Workshops, pp. 377–384. IEEE Computer Society, Munich, Germany (2008). DOI 10.1109/EDOCW.2008.34
7. Andreozzi, S., Marzolla, M.: A RESTful approach to the OGSA Basic Execution Service specification. In: M. Perry, H. Sasaki, M. Ehmann, G.O. Bellot, O. Dini (eds.) Fourth International Conference on Internet and Web Applications and Services, ICIW 2009, 24-28 May 2009, Venice/Mestre, Italy, pp. 131–136. IEEE Computer Society (2009). DOI 10.1109/ICIW.2009.26
8. Anjomshoaa, A., Brisard, F., Drescher, M., Fellows, D., Ly, A., McGough, S., Pulsipher, D., Savva, A.: Job Submission Description Language (JSDL) Specification, Version 1.0 (2005). URL http://www.gridforum.org/documents/GFD.56.pdf. OGF Specification GFD-R.056
9. Barton, T., Basney, J., Freeman, T., Scavo, T., Siebenlist, F., Welch, V., Ananthakrishnan, R., Baker, B., Keahey, K.: Identity Federation and Attribute-based Authorization through the Globus Toolkit, Shibboleth, Gridshib, and MyProxy. In: 5th Annual PKI R&D Workshop (2006)
10. Biron, P.V., Permanente, K., Malhotra, A.: Xml schema part 2: Datatypes; second edition. W3C Recommendation (2004). URL http://www.w3.org/TR/xmlschema-2/
11. Brooks Jr., F.: No silver bullet–essence and accidents of software engineering. Computer **20**(4), 10–19 (1987). DOI 10.1109/MC.1987.1663532
12. Cantor, S., Hirsch, F., Kemp, J., Philpott, R., Maler, E.: Bindings for the oasis security assertion markup language (SAML) v2.0. OASIS Standard saml-bindings-2.0-osn (2005). URL http://docs.oasis-open.org/security/saml/v2.0/saml-bindings-2.0-os.pdf
13. Cantor, S., Kemp, J., Philpott, R., Maler, E.: Assertions and protocols for the oasis security assertion markup language (SAML) v2.0. OASIS Standard saml-core-2.0-os

(2005). URL http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf
14. Case, D.A., T. E. Cheatham, I., Darden, T., Gohlke, H., Luo, R., Jr., K.M.M., Onufriev, A., Simmerling, C., Wang, B., Woods, R.: The amber biomolecular simulation programs. J. Computat. Chem. **26**, 1668–1688 (2005). DOI 10.1002/jcc.20290
15. Common Information Model (CIM) Infrastructure, version 2.3 final. DMTF Document DSP00004 (2005). URL http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf
16. CIM Query Language Specification. DMTF DSP0202 (2007). URL http://www.dmtf.org/standards/published_documents/-DSP0202_1.0.0.pdf
17. CIM-XML (2009). URL http://www.dmtf.org/standards/wbem/CIM-XML
18. Common Management Programming Interface (CMPI). Open Group Technical Standard C051 (2004). URL http://www.opengroup.org/pubs/catalog/c051.htm
19. Czajkowski, K., Fitzgerald, S., Foster, I., Kesselman, C.: Grid information services for distributed resource sharing. In: HPDC '01: Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing, pp. 181–194. IEEE Computer Society, Washington, DC, USA (2001). DOI 10.1109/HPDC.2001.945188
20. Dillaway, B., Humphrey, M., Smith, C., Theimer, M., Wasson, G.: HPC Basic Profile, Version 1.0 (2007). URL http://www.ogf.org/documents/GFD.114.pdf. OGF Specification GFD-R-P.114
21. Ellert, M., Grønager, M., Konstantinov, A., Kónya, B., Lindemann, J., Livenson, I., Nielsen, J., Niinimäki, M., Smirnova, O., Wäänänen, A.: Advanced resource connector middleware for lightweight computational grids. Future Generation Computer Systems **23**(2), 219–240 (2007). DOI 10.1016/j.future.2006.05.008
22. Erwin, D.W.: UNICORE–a grid computing environment. Concurrency and Computation: Practice and Experience **14**(13–15) (2002). DOI 10.1002/cpe.691
23. Fisher, S., et al.: R-GMA: An Information Integration System for Grid Monitoring. In: On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE, pp. 462–481 (2003)
24. Foster, I., Grimshaw, A., Lane, P., Lee, W., Morgan, M., Newhouse, S., Pickles, S., Pulsipher, D., Smith, C., Theimer, M.: OGSA Basic Execution Service Version 1.0 (2007). URL http://www.ogf.org/documents/GFD.108.pdf. OGF Specification GFD.108
25. Foster, I., Kesselman, C.: Globus: a metacomputing infrastructure toolkit. International Journal of High Performance Computing Applications **11**(2), 115–128 (1997). DOI 10.1177/109434209701100205
26. Foster, I., Kesselman, C., Pearlman, L., Tuecke, S., Welch, V.: The community authorization service: Status and future. In: Proceedings of Computing in High Energy Physics 03 (CHEP '03) (2003)
27. Foster, I., Kesselman, C., Tuecke, S.: The anatomy of the grid: Enabling scalable virtual organizations. Int. J. High Perform. Comput. Appl. **15**(3), 200–222 (2001). DOI 10.1177/109434200101500302
28. Foster, I., Kishimoto, H., Savva, A., Berry, D., Djaoui, A., Grimshaw, A., Horn, B., Maciel, F., Siebenlist, F., Subramaniam, R., Treadwell, J., Von Reich, J.: The Open Grid Services Architecture (OGSA), version 1.5. OGF Specification GFD-I.080 (2006). URL http://www.ogf.org/documents/GFD.80.pdf
29. Good, G.: The LDAP Data Interchange Format (LDIF). IETF RFC 2849 (2000)
30. Goodale, T., Jha, S., Kaiser, H., Kielmann, T., Kleijer, P., Merzky, A., Shalf, J., Smith, C.: A Simple API for Grid Applications (SAGA). OGF Specification GFD-R-P.90 (2008). URL http://www.ggf.org/documents/GFD.90.pdf
31. Graham, S., Karmarkar, A., Mischkinsky, J., Robinson, I., Sedukhin, I.: Web Services Resource 1.2 (WS-Resource) (2006). URL http://docs.oasis-open.org/wsrf/wsrf-ws_resource-1.2-spec-os.pdf. OASIS Standard wsrf-ws_resource-1.2-spec-os
32. Gudgin, M., Hadley, M., Rogers, T.: Web Services Addressing 1.0–Core, W3C Recommendation (2006). URL http://www.w3.org/TR/2006/REC-ws-addr-core-20060509
33. Housley, R., Polk, W., Ford, W., Solo, D.: RFC3280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile (2002). URL http://www.ietf.org/rfc/rfc3280.txt
34. The Unofficial Specification of the INI Format, version 1.3 (2003). URL http://www.cloanto.com/specs/ini.html
35. Kónya, B.: The NorduGrid/ARC Information System–technical description and reference manual. Tech. Rep. NORDUGRID-TECH-4 (2007). URL http://www.nordugrid.org/documents/arc_infosys.pdf
36. Laure, E., Fisher, S.M., Frohner, Á., Grandi, C., Kunszt, P., Krenek, A., Mulmo, O., Pacini, F., Prelz, F., White, J., Barroso, M., Buncic, P., Hemmer, F., Di Meglio, A., Edlund, A.: Programming the Grid with gLite. Computational Methods in Science and Technology **12**(1), 33–45 (2006)
37. Moses, T.: OASIS eXtensible Access Control Markup Language (XACML), Version 2.0. OASIS Standard oasis-access_control-xacml-2.0-core-spec-os (2005). URL http://www.oasis-open.org/committees/xacml
38. Nadalin, A., Goodner, M., Gudgin, M., Barbir, A., Granqvist, H.: Ws-trust 1.3. OASIS Standard ws-trust-1.3-spec-os (2007). URL http://docs.oasis-open.org/ws-sx/ws-trust/v1.3/ws-trust.pdf
39. Nadalin, A., Kaler, C., Monzillo, R., Hallam-Baker, P.: Web service security: SAML token profile 1.1. OASIS Standard wss-v1.1-spec-os-SAMLTokenProfile (2006). URL http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-SAMLTokenProfile.pdf
40. Nadalin, A., Kaler, C., Monzillo, R., Hallam-Baker, P.: Web service security: Soap message security 1.1 (ws-security 2004). OASIS Standard Specification wss-v1.1-spec-os-SOAPMessageSecurity (2006). URL http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-SOAPMessageSecurity.pdf
41. Raman, R.: Matchmaking Frameworks for Distributed Resource Management. Ph.D. thesis, University of Wisconsin-Madison (2001)
42. Riedel, M., et al.: Interoperation of world-wide production e-science infrastructures. Concurrency and Computation: Practice and Experience **21**(8), 961–990 (2009). DOI 10.1002/cpe.1402
43. Scavo, T.: Saml v2.0 deployment profiles for x.509 subjects. Committee Draft 02 (2007). URL http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml2-profiles-deploy-x509-cd-02.odt2
44. Thompson, H.S., Beech, D., Maloney, M., Mendelsohn, N.: XML Schema Part 1: Structures; Second Edition. W3C Recommendation (2004). URL http://www.w3.org/TR/xmlschema-1/
45. Tröger, P., Rajic, H., Haas, A., Domagalski, P.: Standardization of an API for Distributed Resource Management Systems. In: Proc. of the Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2007), pp. 619–626. Rio de Janeiro, Brazil (2007)
46. Tuecke, S., Welch, V., Engert, D., Pearlman, L., Thompson, M.: Internet X.509 Public Key Infrastructure (PKI) Proxy Certificate Profile. RFC 3820 (Proposed Standard) (2004). URL http://www.ietf.org/rfc/rfc3820.txt
47. WBEM: Web-based Enterprise Management (2009). URL http://www.dmtf.org/standards/wbem/