ISTITUTO NAZIONALE DI FISICA NUCLEARE
Sezione di Padova

INFN/TC–08/6
October 9, 2008

STANDARDS-BASED JOB MANAGEMENT IN GRID SYSTEMS

Paolo Andreetto[1], Sergio Andreozzi[2], Antonia Ghiselli[2],
Moreno Marzolla[1], Valerio Venturi[2], Luigi Zangrando[1]
[1)] *INFN, Sezione di Padova, via Marzolo 8, I-35131 Padova, Italy*
[2)] *INFN-CNAF, viale Berti Pichat 6/2, I-40127 Bologna, Italy*

Abstract

The Grid paradigm of accessing heterogeneous distributed resources proved to be extremely effective, as many organizations are relying on Grid middlewares for their computational needs. Many of such 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 concern, 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 important Grid 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 form interoperable building blocks for a Grid architecture, and describe how existing Grid software components have actually been re-engineered to support these specifications. From this experience we draw some conclusions on the strengths and weaknesses of the standards, and how they can be improved to address some of the issues we encountered.

PACS: 89.20.Ff; Computer Science and Technology

Published by **SIS-Pubblicazioni**
Laboratori Nazionali di Frascati

1 Introduction

Many large-scale organizations are currently managing their resources using some kind of Grid middleware or Grid infrastructure. The Grid allows seamless access to remote, distributed resources. In particular, job execution and management is one of the capabilities offered by virtually 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 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 can not be forwarded to another Grid based on a different middleware, even if the owner is authorized to access resources on both.

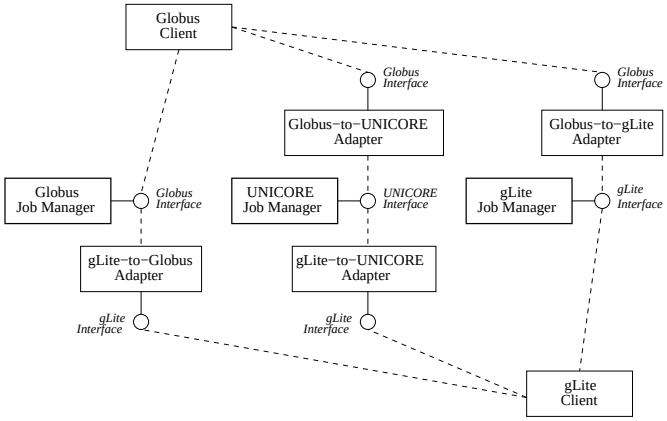
Two specific problems of Grids are *accessibility* and *application portability*. Accessibility includes resource access and portability: currently multiple Grid infrastructures are deployed with little if no interoperability between these infrastructures. Hence a decision to use one particular Grid infrastructure, and hence use one Grid distribution, will curtail 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 aspect to this problem is that of application portability. Each Grid distribution currently specifies its own unique interface to each service. The implication of this is that user applications are Grid distribution specific. An application written for the gLite middleware [9] will not be compatible with the same application written to use a UNICORE Grid [17].

Resource sharing across multiple middlewares is motivated by the increasing demand of scientific applications. As an example [30], 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 of 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) [12] code that could run on HPC resources within DEISA[2]. Hence, cross-Grid usage lead to the benefit of significantly accelerating the drug discovery step.

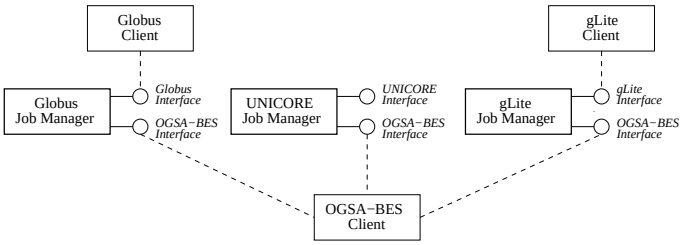
[1] http://wisdom.eu-egee.fr/

[2] http://www.deisa.org/

The problem of sharing resources among heterogeneous Grid middlewares has traditionally been addressed by means of ad-hoc components called *bridges* or *adapters*. An adapter is a component which connects two specific kind of middlewares, say X and Y. The adapter translates messages originating from middleware 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, which achieve *interoperation* between two incompatible systems. Solutions base on adapters have limited scalability: in order to achieve interoperation among N different kind of middlewares, one has to develop $O(N^2)$ different adapters (one for each pair of systems).



(a) Interoperation



(b) Interoperability

Figure 1: Interoperation vs Interoperability. (a) shows the *interoperation* scenario, using adapters to convert incompatible interfaces. (b) shows the *interoperation* scenario, in which components are enhanced with standards-compliant interfaces that can be accessed by any standard client.

On the other hand, full *interoperability* can be achieved with the adoption of common, standardized and possibly open (i.e., non proprietary) interfaces and protocols. 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 leading standards organizations, software vendors, and users. The Open Grid Service Architecture (OGSA) [22] describes an architecture for a service-oriented Grid environment for business and scientific use, developed within the OGF.

The approach based on open standards is more scalable than the adapter-based one: implementing the same interface on N different middlewares requires effort proportional to $O(N)$, as each one of the N middlewares must be enhanced by implementing the standard interface.

The interoperation and interoperability scenarios are depicted in Figure 1. We consider three different Grid job management components, based on the Globus [23], UNICORE [17] and gLite [9] middlewares 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; or job management services, the currently defined standard in based on the OGSA–Basic Execution Service (BES) and Job Submission Description Language (JSDL) specifications, which will be described in detail later in this paper. 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.

Note that the "client" mentioned so far might be a complex service as well. Figure 2 shows how standards-compliant job management services could be used by so-called "Grid Metaschedulers". A Metascheduler can dispatch jobs to multiple job execution services, according to appropriate scheduling decisions. Taking advantage of standard interfaces, a Grid Metascheduler can expose a standard BES interface to clients, and can dispatch jobs to other BES compliant execution services (including other metaschedulers).

The main problem with the interoperability approach is that defining common interfaces for the major Grid services (e.g., job submission, information modelling, authorization and authentication, resource monitoring and so forth) is a difficult and lengthy process. Given that these standards are developed as a cooperation of the major Grid middleware developers, the resulting standards are often made of the (very small) common subset of features offered by each Grid.

In this paper we consider three recently defined standards for Grid middlewares,

[3] http://www.ogf.org

[4] http://www.oasis-open.org

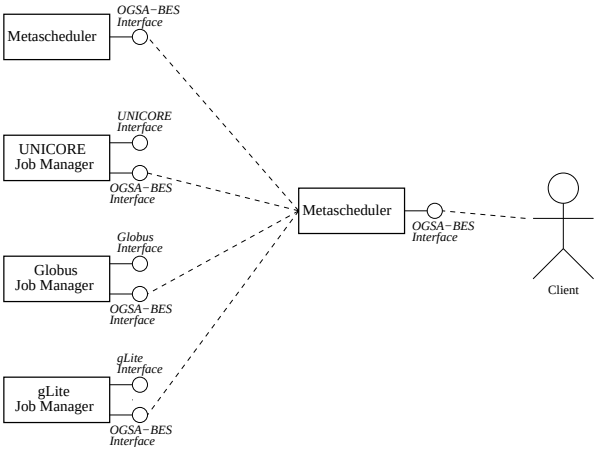


Figure 2: Standards-based job management with Grid metaschedulers

which are related to three major services provided by almost every middleware:

- the BES specification [19], an OGF standard which addresses job submission and management. The implementation of the BES specification in the gLite Computing Resource Execution And Management (CREAM) Computing Element [1] will be discussed in Section 2.
- the Security Assertion Markup Language (SAML) [11], an OASIS-standardized XML language for releasing assertions regarding authentication, attributes and authorization. The implementation of the SAML notation within the gLite Virtual Organization Membership Service (VOMS) service will be discussed in Section 4.
- GLUE [5,4], a conceptual model and reference realizations to concrete data models for describing advertisable capabilities of Grid services and resources. The GLUE specification together with an implementation will be discussed in Section 3.

We discuss in Section 5 a complete architecture for job management based on the standards-compliant components introduced in this paper. Finally, conclusions and future works will be analyzed in Section 6.

2 Job Management

One of the most important functionalities offered by any Grid system is the possibility of submitting and managing jobs, which will then be executed on suitable computational resources. While the exact notion of *job* varies from Grid to Grid, there are many common features which can be isolated. For example, a job usually consists of executing

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 job features across different Grid infrastructures was the motivation of the development of standards for job descriptions and job management. In this way users have a single notation for describing jobs, regardless of the system where they will be executed.

2.1 The JSDL Specification

The JSDL [6] is an XML-based notation for describing the requirements of computational jobs for submission to Grid environments. The JSDL notation is defined by means of a normative XML Schema that facilitates the expression of those requirements as a set of XML elements.

The aim of JSDL is to provide a notation for describing the structure and requirements of individual jobs. Other, equally important, aspects of job submission and management are outside the scope of JSDL. For example, many Grid systems provide the notion of *structured job collections*; as an example, 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 an optional human-readable description of the job. It is a complex element which contains sub-elements for specifying 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. It is used as a high level generic container holding more specific application definitions. 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 specific features which are not present. The JSDL specification has an extension mechanism by means of which it is possible to add specific 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.

2.2 The BES Specification

The BES specification [19] 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 *resources*, where a resource is a generic term 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

Table 1: BES Port-Types and Operations

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

the current BES specification does not include any specific way for accessing and managing individual (contained) resources, so that different implementations provide different access mechanisms.

Technically speaking, the BES Web Services Description Language (WSDL) document defines two Web Service (WS) port-types, which are shown in Table 1 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 the service and to stop it 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) [26]. 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: (1) *pending*, the service has created the activity, but the latter is not yet running on any computational resource: (2) *running*, the activity is executing 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* BES operation; (5) *failed*, the activity has failed due to some error or failure (terminated and failed are terminal states). The state model can be extended to consider new states.

The BES-Factory port-type defines operations for the creation and manipulation of

activities and set of activities. Moreover, it contains an operation (*GetFactoryAttributes-Document*) for retrieving attribute information about the BES service itself. Such information contains, among others, the human-readable service name, the total number of activities currently active in the service, the EPR to activities currently active in the service, and the number of contained resources accessible by the BES. The *GetFactory-AttributesDocument* operation only returns a very simple description of the capabilities of the BES service. However, the BES specification (as well as JSDL) provide standard extension mechanisms so that additional XML elements can be inserted in the normative BES/JSDL 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 the emerging GLUE2 specification, as will be described in Section 3.

2.3 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 [9]. We show in Figure 3 a high-level view of the main architectural components of CREAM.

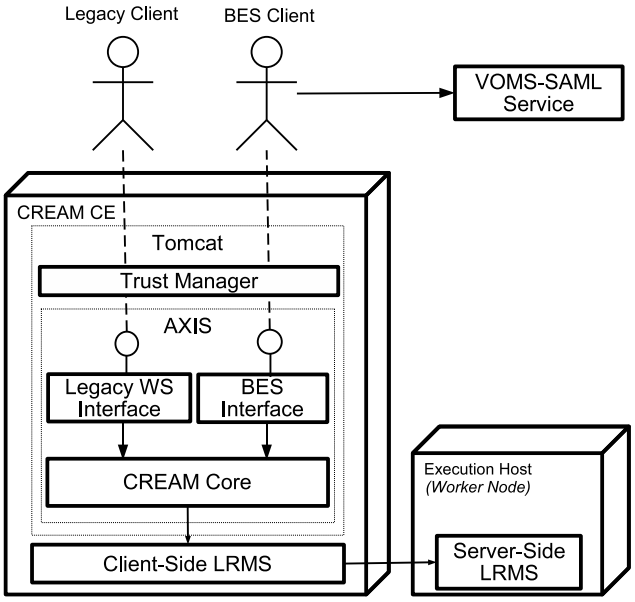


Figure 3: High level structure of the CREAM service enhanced with BES interface

CREAM runs as a Java-Axis servlet [5] in the Tomcat application server [6]. 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, given that the service interface is decoupled from the core, it was possible to support the BES interface together with the legacy one. allowing BES and non-BES (legacy) 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 might consist of a set of command line tools which interact 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, thus it is easy to identify a BES resource with a specific queue of the batch systems. More complex configuration are possible, for example defining some *shares* [4] 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. The core of the service does not provide a single built-in selector, but can be instrumented with a customized module which obtains user credentials and resource information from the core and schedule the correct internal resource. Moreover, users can access the 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.example/path?b=<batch_system>&q=<queue_name>` (note that the CREAM administrator can restrict or forbid access to individual batch queues). Other approaches for accessing contained resources exist, for example using the WS-Resource specification [25]. However, the one adopted in CREAM has the advantage of being far simpler to

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

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

implement, and allows access to contained resources also to non-WSRF compliant clients.

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 format certificate [27]. 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, it must contact an appropriate service capable of releasing SAML (signed) assertions for each request. One of such components is VOMS-SAML, which will be described in detail in Section 4.

3 Information Modelling

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 1.3 schema [5], NorduGrid schema [29]). As specified in the OGSA [22], this information is made available via a Grid information service [15] 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 the unification of the various information models into a community standard specification called GLUE 2.0 [35]. Two documents from this group completed the public comment phase and are close to become OGF proposed recommen-

dations. They cover a conceptual model [4] and reference realizations [3] to concrete data models for Grid resource descriptions.

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, 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.

3.1 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.

3.2 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 identi-

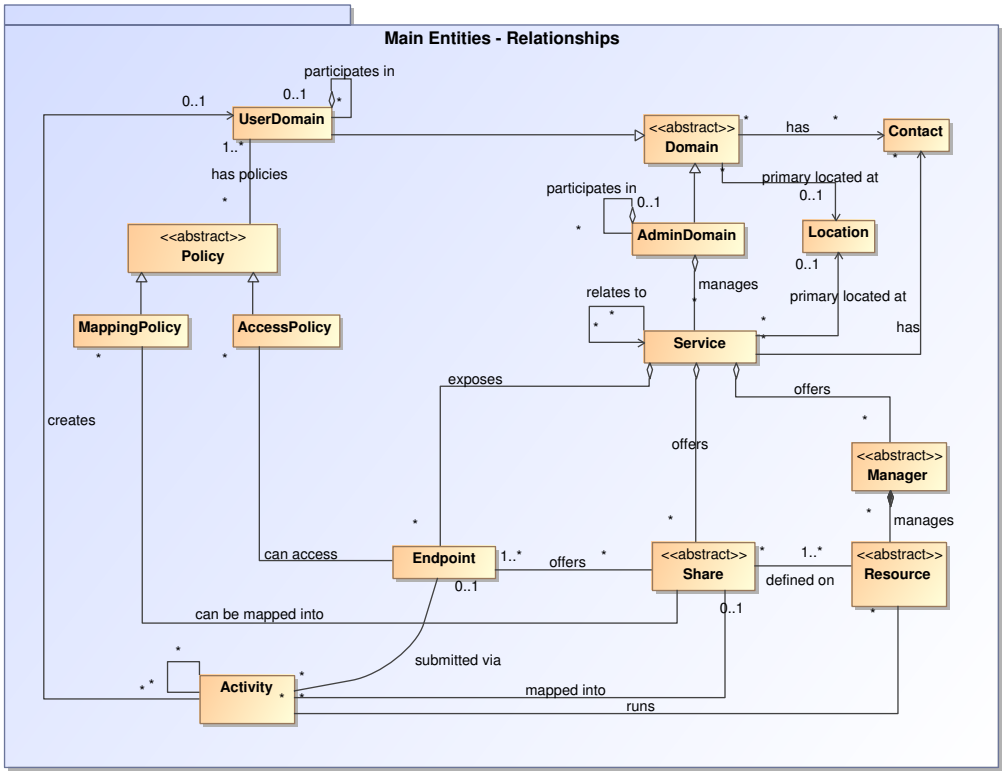


Figure 4: GLUE 2.0 Information Model - Main Entities

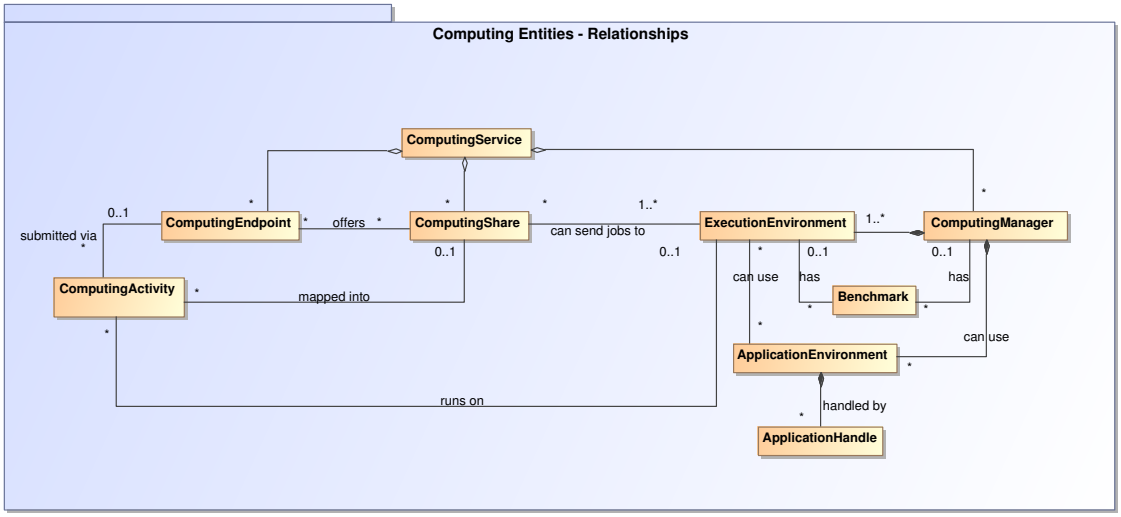


Figure 5: GLUE 2.0 Information Model - Computing Entities

fied 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.

3.3 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

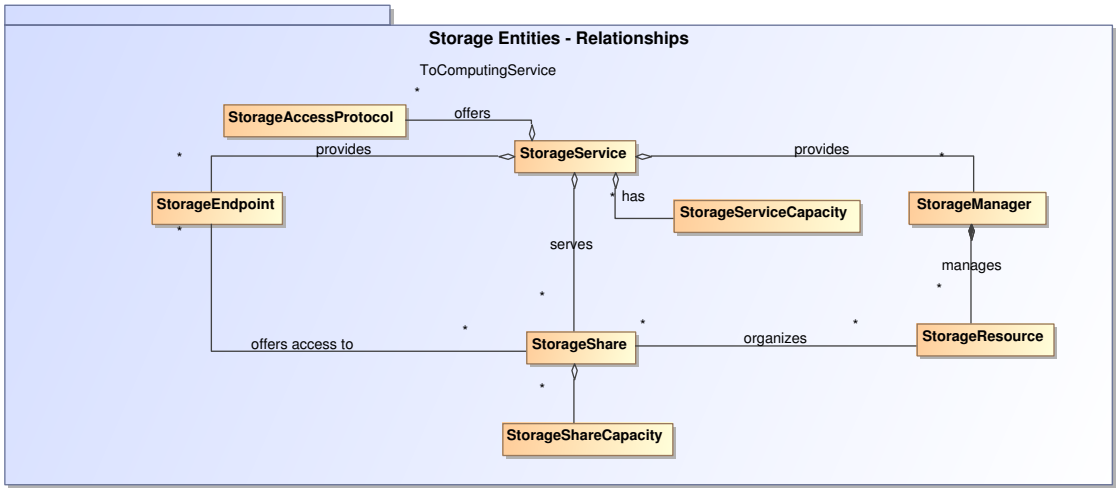


Figure 6: GLUE 2.0 Information Model - Storage Entities

managing storage shares or for accessing them. The *Storage Resource* 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.

3.4 GLUE Realization

The GLUE realizations document [3] describes the mapping of the conceptual model into three different concrete 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://egee-cream-bes.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>
          <Staging> ... </Staging>
        </ComputingEndpoint>
      </ComputingService>
    </Services>
  </AdminDomain>
</glue:Domains>
```

3.5 GLUEMan

We now describe the design and implementation of GLUEMan [7], a framework based on Web-Based Enterprise Management (WBEM)[8] 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.

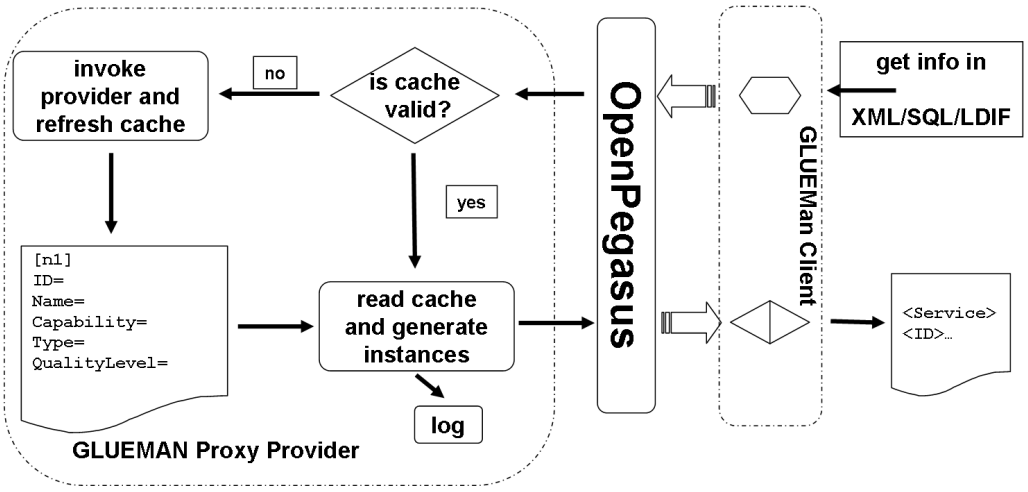


Figure 7: GLUEMan: Simplified Functional View

In Figure 7, we can see a simplified view of the GLUEMan components: a proxy

[7] http://glueman.sf.net/

[8] http://www.dmtf.org/standards/wbem/

provider and a client. The proxy provider is a component added to decouple the interaction of Open Pegasus[9] 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 Common Information Model (CIM) [13] 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.

3.6 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) [14]. 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 [28]. 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

[9] http://openpegasus.org

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.

3.7 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 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.

3.8 Integration of GLUEMan in Grid Middleware

Given a 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 ex-

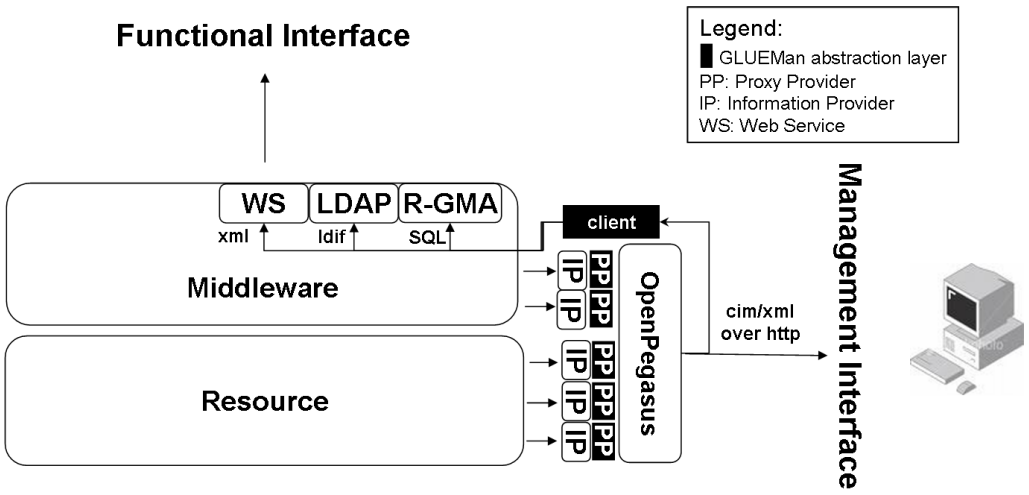


Figure 8: Integration of GLUEMan in Grid Middleware

tract 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) [24], finally, the R-GMA service (Relational Grid Monitoring Architecture) [18] 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.

4 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 Virtual Organizations (VOs) [21]. 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.

Essential in VOs is the ability to establish sharing relationships among *any* potential participants, independent of the nature of the resources and middlewares. Ensur-

ing integrity, confidentiality and interoperability between heterogeneous systems can be achieved using a Web Service Architecture (WSA), 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 estabilished 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.

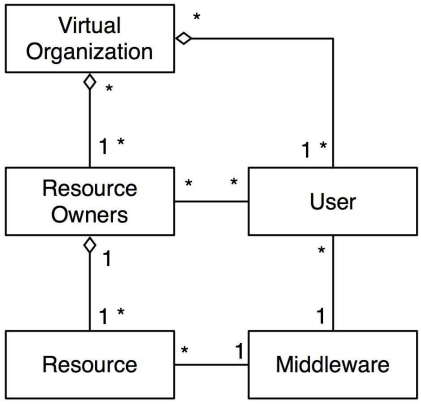


Figure 9: Relationships in a Virtual Organization

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 [9] and Globus[10]. We consider VOMS as a de-facto standard service for Grid authorization systems, but in order to reach complete interoperability it must expose a standardized WS interface.

On the other side, the SAML [11] 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

[10]http://www.globus.org/

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.

4.1 The OASIS Security Assertion Markup Language

The aim of the OGSA authorization working group[11] is to define the specifications needed to allow for interoperability and pluggability of authorization components from multiple authorization domains in the OGSA framework. The group leverages authorization work that is ongoing in the WS community (e.g. SAML, XACML [31], and the WS-Security [34] 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 it's currently working on defining profiles for them:

- the **Attribute Authority** is a service that releases assertions regarding users' attributes. The protocol being defined uses SAML;
- the **Authorization Service** releases authorization decisions based on users' and resources' attributes. The protocol being defined uses XACML and the SAML profile for eXtensible Access Control Markup Language (XACML);
- the **Credential Validation Service** validates users' credentials (digitally signed attributes assertions) to be used by an authorization service. The protocol being defined uses SAML and WS-Trust [32].

The interactions between the functional components of OGSA Authorization are shown in Figure 10. In such a vision, an Attribute Authority (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 [11], and bindings into standards messaging or communication protocols [10].

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

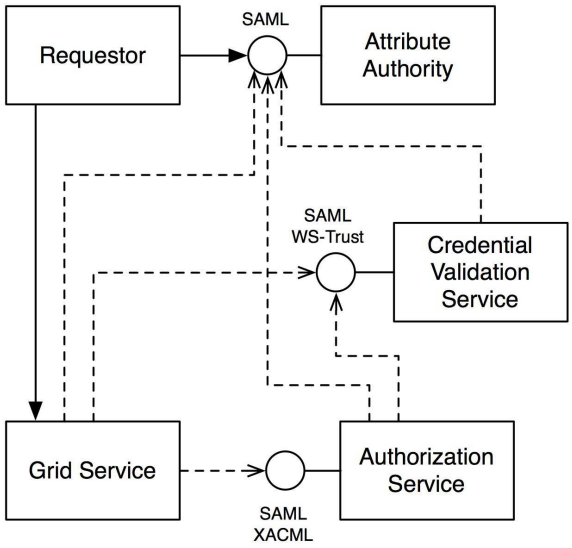


Figure 10: Functional components for a Grid authorization service

4.2 The OASIS Web Service Security

The Web Services Security (WSS) specification [34] 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 [33].

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 possibile range of use patterns. A driving use case has been those Grid middlewares not using proxy certificates [38]. The main interactions between a client and the re-engineered VOMS service are shown in Figure 11.

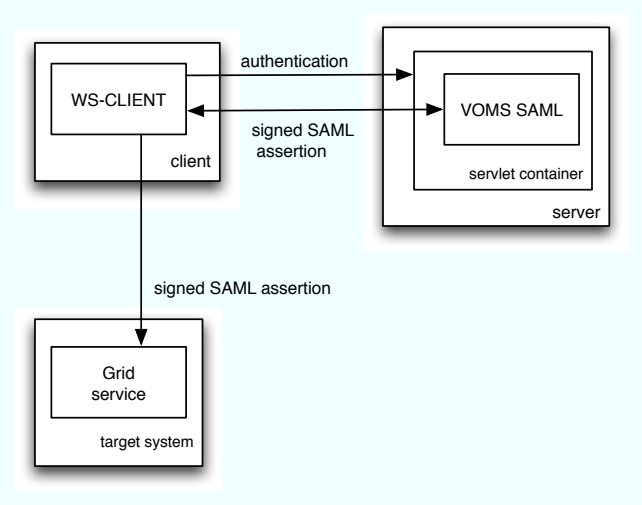


Figure 11: Interactions between a client, a re-engineered VOMS service and a target system

4.3 Service Interface

The VOMS SAML service exposes an interface according to SAML protocols [11] and bindings [10]. 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 successfull `<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 [36].

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>
```

4.4 Expressing the VOMS attributes using SAML

The core functionality of VOMS is expressing attributes, but the Fully Qualified Attribute Name attribute and the Tag attribute 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 [37,8]. 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 schema 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 moment 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.

4.5 Sending SAML Assertions to Grid Services

In theAttribute 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[12] Community Authorization Service (CAS) [20] binds authorization Assertions to a proxy. GridShib [7] does the same with authentication Assertions. As described in section 4.2, 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 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="...">
    ...
```

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

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

5 Putting the Components Together

We now describe how the standards and the components introduced in the previous sections can be used as standard building blocks for a complete Grid architecture for job submission and management.

The architecture is shown in Figure 12. We actually implemented the components shown with a shaded background, and we described them in the previous sections.

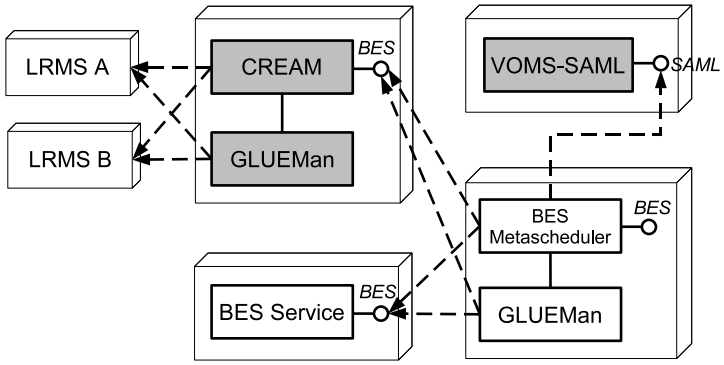


Figure 12: Complete architecture for a standards-based job submission and management service

We start by considering the BES/JSDL compliant job submission services. The schema contains a CREAM service (only the BES interface is shown). As already described in Section 2, CREAM provides the following features:

- it implements the BES interface together with the (mandatory) support for the JSDL specification;
- it can manage different contained resources (LRMS, i.e. batch queues), as stated into the BES specification. In Figure 12 we see that the CREAM service is associated with two batch systems, denoted as “LRMS a” and “LRMS B”.
- a mechanism for retrieving information about the status of the contained resources by means of an information provider.

The status of the BES endpoint and its contained resources is returned by the *GetFactoryAttributesDocument* operation (see Table 1). By default, all BES implementations must support a simple hierarchical information model. The *GetFactoryAttributesDocument* returns a `<FactoryResourceAttributesDocument` XML 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 resource.

As can be seen, a very limited set of informations can be exposed through the `<FactoryResourceAttributesDocument>`. 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 [3]. The resulting XML block can be inserted in the "placeholder" denoted in the schema fragment above with the `<xsd:any ... >` element.

CREAM interacts with an external source of information in a pluggable way, allowing the administrator to select the suitable extension for the given information system (BDII, R-GMA, GLUEMan). In order to expose to the users a GLUE 2.0-compliant information model, BES services need to interact with a GLUE 2.0 information provider such as GLUEMan. This association is shown in Figure 12 as a line connecting the CREAM and GLUEMan services. GLUEMan collects information on the resources available to the BES service (in this case, the LRMS A and B). The BES service periodically contacts GLUEMan to get the up-to-date GLUE 2.0 representation of the resources, and provides this representation to BES clients by means of the *GetFactoryAttributesDocument* operation.

The architecture shown in Figure 12 also also contains a BES-compliant Grid metascheduler, which is a component which accepts JSDL activities from clients and forwards them to another BES endpoint for execution (this other endpoint might be another metascheduler). Since the metascheduler exposes an ordinary BES interface, it must provide informations on the contained resources (in our example, the two BES endpoints it forwards jobs to) via the *GetFactoryAttributesDocument* operation. The standard BES resource description can be enhanced also in this case with a more complete GLUE 2.0 XML rendering, which the metascheduler can obtain by contacting the local GLUEMan component. This component periodically fetches the GLUE 2.0 information provided by the BES endpoints by invoking their *GetFactoryAttributesDocument* operation, and merging the information returned by them. In this case, the GLUEMan component associated with the metascheduler acts as a mere aggregator for the data gathered by the information providers of the other BES endpoints.

We observe that the metascheduler interacts also with the VOMS-SAML service. Once again, this is due to the particular role of the metascheduler, as it must forward requests from the user 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.

6 Conclusions

In this paper we described three major Grid standards which are relevant to job execution and management across Grid middlewares: the JSDL/BES specification, the SAML notation and the GLUE 2.0 information model (note that the GLUE 2.0 specification has passed is public comment phase and is expected to become an OGF proposed recommendation in the near future). For each specification, we described how it has been implemented in an actual software component. Specifically, the CREAM and VOMS components from the gLite middleware have been reengineered to support BES/JSDL and SAML respectively, while the GLUEMan information provider supporting the GLUE 2.0 candidate specification has been developed from scratch. We finally described a general Grid architecture for job management which makes use of these specifications in general, and the abovementioned components in particular. A prototype implementation of this architecture has been made, integrating CREAM with VOMS-SAML and GLUEMan, in the context of the OMII-Europe project [13].

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

The adoption of open standard like those described in this paper helps to achieve interoperability between different Grid middlewares; moreover, it also allows middlewares to be assembled by using basic "building-blocks" which can be easily assembled together as they only rely on standard interfaces. So, for example, any BES-compliant CE can be plugged into an existing middleware which supports the BES specification.

However, since these standards are relatively new, there are some shortcomings in using them in a production environment. Since middleware developers started implementing them, a number of issues appeared. We now describe those which in our opinion are the most important.

As regards the job description, the JSDL specification only allows the description of individual jobs. However, most middlewares allow users to submit for execution structured collections of jobs. For example, in the gLite middleware there is the notion of DAGs and parametric jobs. A DAG is a set of jobs with dependencies modeled as a Directed Acyclic Graph. The gLite Workload Management System takes care of scheduling all the individual jobs respecting their dependencies. A parametric job represents a set of identical jobs which are invoked with different input parameters. Currently the JSDL specification does not allow the representation of structured jobs; however, JSDL extensions are being worked out to address this limitation.

Concerning the information modeling, as observed in Section 5, the standard BES specification lacks an adequate 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 effectively *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 over the network to the client for processing. Moreover, client applications will likely not be interested in the whole GLUE 2.0 representation of all the resources within a BES service, but only in a limited subset of it. This suggests the use case of letting the client pass to the BES services a query statement written in some appropriate language (XPath or XQuery, for example), so that the query can be processed server-side and only the result returned to the client. Querying the information service through the BES interface is still an open issue.

In the resource selection context, a related issue is the specification of JSDL requirements with respect to the GLUE 2.0 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. 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 the more detailed information model, we need a better notation for specifying requirements on a GLUE 2.0 resource representation.

The final considerations are devoted to the security aspects. BES currently lacks a minimum agreed security profile. Security considerations are outside the BES specification, so that one can claim BES compliance without actually being interoperable with a different producer's BES client, because they use different security settings. To mitigate this issue, several so-called *profiles* have been proposed to complement the BES/JSDL specifications. A profile defines a set of specifications and/or extensions which must be supported by all conforming implementations. The HPC-Basic Profile [16] defines initial security based on TLS/SSL using X.509 certificates [38], or TLS/SSL using username/-password. This security set-up is not realistic for infrastructure integration purposes. Secure data staging probably requires a suitable mechanism for the client to delegate its credentials to the BES service, so that the BES service can execute operations (e.g., access remote files) on behalf of the client. This is heavily used in e.g. the gLite middleware (the legacy CREAM interface exposes a separate delegation port-type which must be used by clients to delegate their credentials), but this is not defined by the BES specification.

References

[1] C. Aiftimiei, P. Andreetto, S. Bertocco, D. Cesini, M. Corvo, S. Dalla Fina, S. Da Ronco, D. Dongiovanni, A. Dorigo, A. Gianelle, C. Grandi, M. Marzolla, M. Mazzucato, V. Miccio, A. Sciaba', M. Sgaravatto, M. Verlato, and L. Zangrando. Job submission and management through web services: the experience with the CREAM service. *Journal of Physics, Conference Series*, 119(6), 2008.

[2] Roberto Alfieri, Roberto Cecchini, Vincenzo Ciaschini, Luca dell'Agnello, Ákos Frohner, Alberto Gianoli, Károly Lörentey, and Fabio Spataro. VOMS, an authorization system for virtual organizations. In *European Across Grids Conference*, pages 33–40, 2003.

[3] S. Andreozzi, S. Burke, F. Ehm, L. Field, G. Galang, B. Konya, M. Litmaath, P. Millar, and J.P. Navarro. GLUE 2.0 - Reference Realizations to Concrete Data Models. `http://forge.ogf.org/sf/docman/do/listDocuments/projects.glue-wg/docman.root.public\_comment`, 2008. OGF Proposed Recommendation in Public Comment.

[4] S. Andreozzi, S. Burke, F. Ehm, L. Field, G. Galang, B. Konya, M. Litmaath, P. Millar, and J.P. Navarro. GLUE 2.0 Specification.

http://forge.ogf.org/sf/docman/do/listDocuments/projects.glue-wg/docman.root.public_comment, 2008. OGF Proposed Recommendation in Public Comment.

[5] S. Andreozzi, S. Burke, L. Field, S. Fisher, B. Kónya, M. Mambelli, J.M. Schopf, M. Viljoen, A. Wilson, and R. Zappi. GLUE Schema Specification - Version 1.3, Jan 2007.

[6] Ali Anjomshoaa, Fred Brisard, Michel Drescher, Donal Fellows, An Ly, Stephen McGough, Darren Pulsipher, and Adreas Savva. *Job Submission Description Language (JSDL) Specification, Version 1.0*, November 1 2005. OGF Specification GFD-R.056, `http://www.gridforum.org/documents/GFD.56.pdf`.

[7] T. Barton, J. Basney, T. Freeman, T. Scavo, F. Siebenlist, V. Welch, R. Ananthakrishnan, B. Baker, and K. Keahey. Identity Federation and Attribute-based Authorization through the Globus Toolkit, Shibboleth, Gridshib, and MyProxy. *5th Annual PKI R&D Workshop*, April 2006.

[8] Paul V. Biron, Kaiser Permanente, and Ashok Malhotra. Xml schema part 2: Datatypes; second edition. W3C Recommendation, `http://www.w3.org/TR/xmlschema-2/`, October 28 2004.

[9] Stephen Burke, Simone Campana, Antonio Delgado Peris, Flavia Donno, Patricia Mendez Lorenzo, Roberto Santinelli, and Andrea Sciabà. *gLite 3 user guide–Manuals Series*, January17 2007. Version 1.1, Document identifier CERN-LCG-GDEIS-722398. `https://edms.cern.ch/document/722398/1.1`.

[10] S. Cantor, F. Hirsch, J. Kemp, R. Philpott, and E. Maler. Bindings for the oasis security assertion markup language (SAML) v2.0. OASIS Standard saml-bindings-2.0-osn, `http://docs.oasis-open.org/security/saml/v2.0/saml-bindings-2.0-os.pdf`, 2005.

[11] S. Cantor, J. Kemp, R. Philpott, and E. Maler. Assertions and protocols for the oasis security assertion markup language (SAML) v2.0. OASIS Standard saml-core-2.0-os, `http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf`, March 15 2005.

[12] D. A. Case, III T. E. Cheatham, T. Darden, H. Gohlke, R. Luo, K. M. Merz Jr., A. Onufriev, C. Simmerling, B. Wang, and R. Woods. The amber biomolecular simulation programs. *J. Computat. Chem.*, 26:1668–1688, 2005.

[13] Common Information Model (CIM) Infrastructure, version 2.3 final. DMTF Document DSP00004, http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf, October 4 2005.

[14] Common Management Programming Interface (CMPI). Open Group Technical Standard C051, Dec 2004.

[15] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid Information Services for Distributed Resource Sharing. In *Proceedings of the 10th IEEE International Symposium on High-Performance Distributed Computing (HPDC-10), San Francisco, CA, USA*, Aug 2001.

[16] Blair Dillaway, Marty Humphrey, Chris Smith, Marvin Theimer, and Glenn Wasson. HPC Basic Profile, Version 1.0, August 28 2007. OGF Specification GFD-R-P.114, http://www.ogf.org/documents/GFD.114.pdf.

[17] Dietmar W. Erwin. UNICORE–a grid computing environment. *Concurrency and Computation: Practice and Experience*, 14(13–15), 2002.

[18] S. Fisher et al. R-GMA: An Information Integration System for Grid Monitoring. In *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE*, pages 462–481, 2003.

[19] I. Foster, A. Grimshaw, P. Lane, W. Lee, M. Morgan, S. Newhouse, S. Pickles, D. Pulsipher, C. Smith, and M. Theimer. OGSA Basic Execution Service Version 1.0, August 2007. OGF Specification GFD.108, http://www.ogf.org/documents/GFD.108.pdf.

[20] I. Foster, C. Kesselman, L. Pearlman, S. Tuecke, and V. Welch. The community authorization service: Status and future. In *Proceedings of Computing in High Energy Physics 03 (CHEP '03)*, March 24–28 2003.

[21] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International J. Supercomputer Applications*, 15(3):200–222, 2001.

[22] I. Foster, H. Kishimoto, A. Savva, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, J. Treadwell, and J. Von Reich. The Open Grid Services Architecture (OGSA), version 1.5. OGF Specification GFD-I.080, http://www.ogf.org/documents/GFD.80.pdf, Jul 2006.

[23] The Globus Toolkit. http://www.globus.org/toolkit/, 2008.

[24] G. Good. The LDAP Data Interchange Format (LDIF). IETF RFC 2849, Jun 2000.

[25] Steve Graham, Anish Karmarkar, Jeff Mischkinsky, Ian Robinson, and Igor Sedukhin. *Web Services Resource 1.2 (WS-Resource)*, April 1 2006. OASIS Standard wsrf-ws_resource-1.2-spec-os, `http://docs.oasis-open.org/wsrf/wsrf-ws\_resource-1.2-spec-os.pdf`.

[26] Martin Gudgin, Marc Hadley, and Tony Rogers. *Web Services Addressing 1.0–Core, W3C Recommendation*, May 9 2006. `http://www.w3.org/TR/2006/REC-ws-addr-core-20060509`.

[27] R. Housley, W. Polk, W. Ford, and D. Solo. RFC3280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. `http://www.ietf.org/rfc/rfc3280.txt`, April 2002.

[28] The Unofficial Specification of the INI Format, version 1.3. http://www.cloanto.com/specs/ini.html, September 4 2003.

[29] B. Kónya. The NorduGrid/ARC Information System - technical description and reference manual. Technical Report NORDUGRID-TECH-4, May 17 2007. `http://www.nordugrid.org/documents/arc\_infosys.pdf`.

[30] Moreno Marzolla, Paolo Andreetto, Valerio Venturi, Andrea Ferraro, Shiraz Memon, Shahbaz Memon, Bastian Twedell, Morris Riedel, Daniel Mallmann, Achim Streit, Svan van de Berghe, Vivian Li, David Snelling, Katerina Stamou, Zeeshan Ali Shah, and Fredrik Hedman. Open standards-based interoperability of job submission and management interfaces across the grid middleware platforms gLite and UNICORE. *e-Science and Grid Computing, IEEE International Conference on*, pages 592–601, October 10–13 2007.

[31] T. Moses. Oasis extensible access control markup language (xacml), version 2.0. OASIS Standard oasis-access_control-xacml-2.0-core-spec-os, `http://www.oasis-open.org/committees/xacml`, February 2005.

[32] A. Nadalin, M. Goodner, M. Gudgin, A. Barbir, and H. Granqvist. Ws-trust 1.3. OASIS Standard ws-trust-1.3-spec-os, `http://docs.oasis-open.org/ws-sx/ws-trust/v1.3/ws-trust.pdf`, March 2007.

[33] A. Nadalin, C. Kaler, R. Monzillo, and P. Hallam-Baker. Web service security: SAML token profile 1.1. OASIS Standard wss-v1.1-spec-os-SAMLTokenProfile, http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-SAMLTokenProfile.pdf, February 1 2006.

[34] A. Nadalin, C. Kaler, R. Monzillo, and P. Hallam-Baker. Web service security: Soap message security 1.1 (ws-security 2004). OASIS Standard Specification wss-v1.1-spec-os-SOAPMessageSecurity, http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-SOAPMessageSecurity.pdf, 2006.

[35] OGF GLUE Working Group. http://forge.ogf.org/sf/sfmain/do/viewProject/projects.glue-wg, 2008.

[36] Tom Scavo. Saml v2.0 deployment profiles for x.509 subjects. Committee Draft 02, http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml2-profiles-deploy-x509-cd-02.odt2, August 28 2007.

[37] Henry S. Thompson, David Beech, Murray Maloney, and Noah Mendelsohn. XML Schema Part 1: Structures; Second Edition. W3C Recommendation, http://www.w3.org/TR/xmlschema-1/, October 28 2004.

[38] S. Tuecke, V. Welch, D. Engert, L. Pearlman, and M. Thompson. Internet X.509 Public Key Infrastructure (PKI) Proxy Certificate Profile. RFC 3820 (Proposed Standard), June 2004.