Design and Implementation of the gLite CREAM Job Management Service

Cristina Aiftimiei[a,1], Paolo Andreetto[a], Sara Bertocco[a], Simone Dalla Fina[a], Alvise Dorigo[a], Eric Frizziero[a], Alessio Gianelle[a], Moreno Marzolla[*,b], Mirco Mazzucato[a], Massimo Sgaravatto[a], Sergio Traldi[a], Luigi Zangrando[a]

[a] *Istituto Nazionale di Fisica Nucleare (INFN)*
via Marzolo 8, I-35131 Padova (Italy)
[b] *Dipartimento di Scienze dell'Informazione, Università di Bologna*
Mura A. Zamboni 7, I-40127 Bologna (Italy)

Abstract

Job execution and management is one of the most important functionality provided by every modern Grid systems. In this paper we describe how the problem of job management has been addressed in the gLite middleware by means of the CREAM and CEMonitor services. CREAM (Computing Resource Execution and Management) provides a job execution and management capability for Grids, while CEMonitor is a general-purpose asynchronous event notification framework. Both components expose a Web Service interface allowing conforming clients to submit, manage and monitor computational jobs to a Local Resource Management System.

Key words: Web Services, gLite Middleware, Grid Computing, Grid Job

[*]Corresponding Author. Address: Dipartimento di Scienze dell'Informazione, Università di Bologna, Mura A. Zamboni 7, I-40127 Bologna, Italy. This work was done while the author was with the Istituto Nazionale di Fisica Nucleare (INFN), Padova, Italy.

Email addresses: cristina.aiftimiei@pd.infn.it (Cristina Aiftimiei), paolo.andreetto@pd.infn.it (Paolo Andreetto), sara.bertocco@pd.infn.it (Sara Bertocco), simone.dallafina@pd.infn.it (Simone Dalla Fina), alvise.dorigo@pd.infn.it (Alvise Dorigo), eric.frizziero@pd.infn.it (Eric Frizziero), alessio.gianelle@pd.infn.it (Alessio Gianelle), marzolla@cs.unibo.it (Moreno Marzolla), mirco.mazzucato@pd.infn.it (Mirco Mazzucato), massimo.sgaravatto@pd.infn.it (Massimo Sgaravatto), sergio.traldi@pd.infn.it (Sergio Traldi), luigi.zangrando@pd.infn.it (Luigi Zangrando)

[1]On leave from NIPNE-HH, Romania

1. Introduction

Grid middleware distributions are often large software artifacts, which include a set of components providing a basic functionality. Such capabilities include (but are not limited to) data storage, authentication and authorization, resource monitoring, and job management. The job management component is used to submit, cancel, and monitor jobs which are executed on a suitable computational resource, usually referred as a Computing Element (CE). A CE is the interface to a usually large farm of computing hosts managed by a Local Resource Management System (LRMS), such as LSF or PBS. Moreover, a CE implements additional features with respect to the ones provided by the underlying batch system, such as Grid-enabled user authentication and authorization, accounting, fault tolerance and improved performance and reliability.

In this paper we describe the architecture of Computing Resource Execution and Management (CREAM), a system designed to efficiently manage a CE in a Grid environment. CREAM provides a simple, robust and lightweight service for job operations. It exposes an interface based on Web Services, which enables a high degree of interoperability with clients written in different programming languages: currently Java and C++ clients are provided, but it is possible to use any language with a Web Service framework. CREAM itself is written in Java, and runs as an extension of a Java-Axis servlet inside the Apache Tomcat application server [1].

As stated before, it is important for users to be able to monitor the status of their jobs. This means checking whether the job is queued, running, or finished; moreover, extended status information (such as exit code, failure reason and so on) must be obtained from the job management service. While CREAM provides an explicit operation for querying the status of a set of jobs, it is possible to use a separate notification service in order to be notified when a job changes its status. This service is provided by CEMonitor, which is a general-purpose asynchronous notification engine. CEMonitor can be used by CREAM to notify the user about job status changes. This feature is particularly important for specialized CREAM clients which need to handle a large amount of jobs. In these cases, CEMonitor makes the expensive polling operations unnecessary, thus reducing the load on CREAM and increasing the overall responsiveness.

CREAM and CEMonitor are part of the gLite [2] middleware distribution and currently in production use within the EGEE Grid infrastructure [3]. Users can install CREAM in stand-alone mode, and interact directly with it through custom clients or using the provided C++-based command line tools. Moreover, gLite users can transparently submit jobs to CREAM through the gLite Workload Management System (WMS). For the latter case, a special component called Interface to Cream Environment (ICE) has been developed. ICE receives job submission and cancellation requests coming from a gLite WMS, and forwards these requests to CREAM. ICE then handles the entire lifetime of a job, including registering each status change to the gLite Logging and Bookkeeping (LB) service [4].

1.1. Related Works

The problem of job management is addressed by any Grid system. Different job management services have been developed starting from different requirements; furthermore, each service must take into account the specific features of the middleware it belongs to.

The UNICORE (Uniform Interface to Computing Resources) [5] system was initially developed to allow German supercomputer centers to provide seamless and secure access to their computational resources. Architecturally, UNICORE is a three-tier system. The first tier is made of clients, which submit requests to the second tier (server level). The server level of UNICORE consists of a Gateway which authenticates requests from UNICORE clients and forwards them to a Network Job Supervisor (NJS) for further processing. The NJS maps the abstract requests into concrete jobs or actions which are performed by the target system. Sub-jobs that have to be run at a different site are transferred to this site's gateway for subsequent processing by the peer NJS. The third tier of the architecture is the target host which executes the incarnated user jobs or system functions.

The Advanced Resource Connector (ARC) [6] is a Grid middleware developed by the NorduGrid collaboration. ARC is based on the Globus Toolkit[2], and basically consists of three fundamental components: the *Computing Service* which represents the interface to a computing resource (generally a cluster of computers); the *Information System* which is a distributed database maintaining a list of know resources; and a *Brokering Client* which allows

[2]Globus and Globus Toolkit are trademarks of the University of Chicago

resource discovery and is able to distribute the workload across the Grid.

The Globus Toolkit provides both a suite of services to submit, monitor, and cancel jobs on Grid computing resources. GRAM4 refers to the Web Service implementation of such services [7]. GRAM4 includes a set of WSRF-compliant Web Services [8] to locate, submit, monitor, and cancel jobs on Grid computing resources. GRAM4 is not a job scheduler, but a set of services and clients for communicating with different batch/cluster job schedulers using a common protocol. GRAM4 combines job-management services and local system adapters with other service components of the Globus Toolkit in order to support job execution with coordinated file staging.

Initially, the job management service of the gLite middleware was implemented by the legacy LGC-CE [9], which is based on the pre-Web Service version of GRAM. The development of CREAM was motivated by some shortcomings of the LCG-CE related to performance and security issues. These issues and other requirements behind the development of CREAM will be discussed in Section 3.1.

1.2. Organization of this paper

This paper is organized as follows. In Section 2 we give a high level overview on the job management chain in the gLite middleware. Then, in Section 3 we restrict our attention on the CREAM and CEMonitor services: we illustrate the requirements defined in the gLite design document for the Computing Element, and give a high level description of CREAM and CEMonitor. Internal details on CREAM are given in Section 4, and details on CEMonitor are given in Section 5. The interactions with CREAM and CEMonitor which are necessary to handle the whole job submission sequence are then explained in Section 6. Section 7 describes how CREAM and CEMonitor are built and deployed in the gLite production infrastructure. Section 8 contains performance considerations, and we discuss conclusions and future works in Section 9.

2. Job Management in the gLite Middleware

In this section we give a brief introduction to the job management architecture of the gLite middleware. The interested reader is referred to [2, 9] for a more complete description.

Fig. 1 shows the main components involved in the gLite job submission chain. We will consider job submission to the CREAM CE only. The Job-

Controller+LogMonitor+CondorG and LCG-CE components are responsible for job management through the legacy LCG-CE, and will not be described in this paper.

There are two entry points for job management requests: the gLite WMS User Interface (UI) and the CREAM UI. Both include a set of command line tools which can be used to submit, cancel and query the status of jobs. In gLite, jobs are described using the Job Description Language (JDL) notation, which is a textual notation based on Condor classads [10]. In Fig. 1 we have emphasized the paths from the WMS UI to CREAM (top) and to the legacy LCG-CE (bottom).

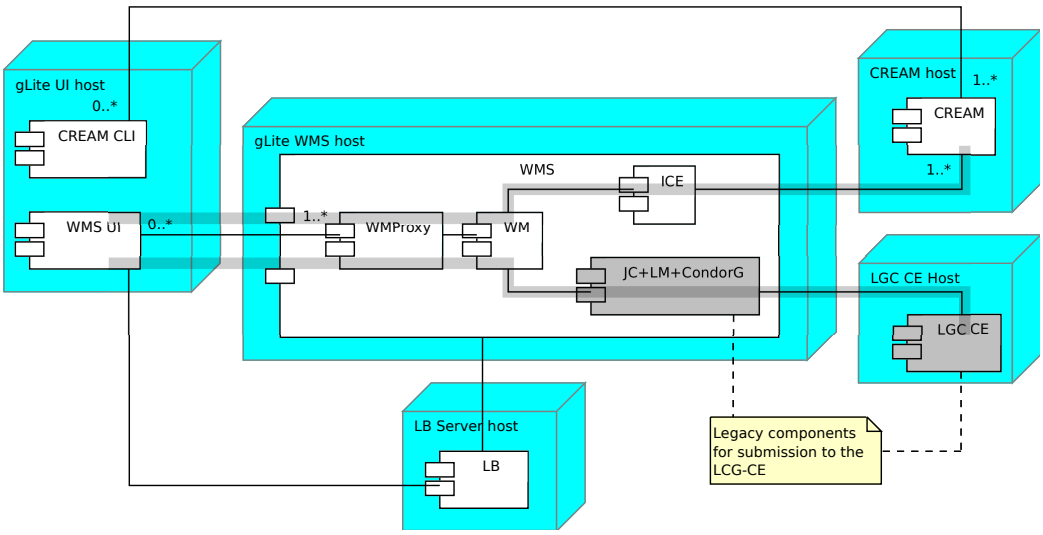

Figure 1: Job submission chain (simplified) in the gLite middleware. Emphasized paths are for job submissions from the WMS User Interface to CREAM (top) and to the legacy LCG-CE (bottom).

The CREAM UI is used to interact directly with a specific CREAM CE. It is a set of command line tools, written in C++ using the gSoap engine [11]. The CREAM CLI provides a set of commands to invoke the Web Services operations exposed by CREAM (the list of available operations is given in Section 4).

On the other hand, the gLite WMS UI allows the user to submit and monitor jobs through the gLite Workload Management System (WMS) [12]. The WMS is responsible for the distribution and management of tasks across Grid resources (in particular Computing Elements), in such a way that appli-

cations are efficiently executed. Job management through the WMS provides many benefits compared to direct job submission to the CE:

- The WMS can manage multiple CEs, and is able to forward jobs to the one which better satisfies a set of requirements, which can be specified as part of the job description;
- The WMS can be instructed to handle job failures: if a job aborts due to problems related to the execution host (e.g. host misconfiguration) the WMS can automatically resubmit it to a different CE;
- The WMS provides a global job tracking facility using the LB service;
- The WMS supports complex job types (job collections, job with dependencies) which can not be handled directly by the CEs.

Note that there is a many to many relationship between the gLite WMS UI and the WMS, that is, multiple User Interfaces can submit to the same WMS, and multiple WMSs can be associated with the same WMS UI.

The WMS exposes a Web Service interface which is implemented by the WMProxy component. The core of the WMS is the Workload Manager (WM), whose purpose is to accept and satisfy requests for job management. For job submissions, the WM tries to locate an appropriate resource (CE) where the job can be executed. The decision of which resources should be used is the outcome of the matchmaking process between the requests and the available resources. The user can specify a set of *requirements* in the job description. These requirements represent a set of constraints which the WM tries to satisfy when selecting the CE where the job will be executed.

Currently, the gLite WMS can submit jobs to CREAM and to the legacy LCG-CE. Each CE is uniquely identified by a URI called *ce-id*. Looking at the ce-id, the WM is able to discriminate whether the job is to be submitted to a CREAM-based CE, or to a LCG-CE (URIs denoting a CREAM CE have the prefix `cream-` in the path). Interaction with the LCG-CE is handled by the Job Controller/Log Monitor/CondorG (JC/LM/CondorG) modules within the WMS. In the case of submission to CREAM-based CEs, jobs are managed by a different module, called ICE. ICE receives job submissions and other job management requests from the WM component of the WMS through a simple messaging system based on local files. ICE then uses the operations of the CREAM interface to perform the requested operation.

Moreover, it is responsible for monitoring the state of submitted jobs and for taking the appropriate actions when job status changes are detected (e.g. to trigger a possible resubmission if a Grid failure is detected).

ICE can obtain the state of a job in two different ways. The first one is by subscribing to a job status change notification service implemented by a separate component called CEMonitor (more details in Section 5). CEMonitor [13] is a general purpose event notification framework. CREAM notifies the CEMonitor component about job state changes by using the shared, persistent CREAM backend. ICE subscribes to CEMonitor notifications, so it receives all status changes whenever they occur. As a fallback mechanism, ICE can also poll the CREAM service to check the status of "active" jobs for which it did not receive any notification for a configurable period of time. This mechanism guarantees that ICE knows the state of jobs even if the CEMonitor service becomes unavailable or has not been installed.

In general, jobs may require a set of input data files to process, and produce a set of output data files. The set of input files is called InputSandBox (ISB), and the set of output files is called OutputSandBox (OSB). For both submission to the LCG-CE and to CREAM, data staging (i.e., copying files from/to remote locations) is performed by the Job Wrapper (JW) which runs on the execution node and which encompasses the run of the actual user payload. In either cases, the WM component can safely assume that data staging is performed downstream on the job submission chain.

The LB service [4] is used by the WMS to store various information on running jobs, and provide the user with an overall view on the job state. The service collects events in a non blocking asynchronous way, and this information can be used to compute the job state. LB is also used to store events such as the transfer of jobs from one component to another one (e.g., from the WMproxy to the WM): in this way, the user knows the location of each job. The job status information gathered by the LB is made available through the gLite UI commands. Note that in case of direct submissions through the CREAM UI, the LB service is not used; however, CREAM itself provides the *JobInfo* operation for reporting detailed job status information.

3. CREAM and CEMonitor

3.1. Requirements

The development of CREAM and CEMonitor has been driven by the need to provide a modern replacement of the LCG-CE for the gLite middleware.

The legacy LCG-CE suffered from several problems, including:

- Security issues; in particular, the LCG-CE does not support proper delegation of user credentials;
- Poor performance: the CE could sustain a low submission rate even on modern hardware (see Section 8 for actual measurements);
- Reliability issues: in some situations, user jobs disappeared from the Grid, while still running on the execution node. In these situations, only site administrators could terminate such "zombie" jobs.
- Lack of support: the code was no longer maintained, so fixing bugs and making any improvement was increasingly difficult. For this reason, the EGEE project decided to start the development of a new Computing Element.

A set of requirements for the new CE were identified and described in the EGEE design document [14]. Thus, the development of CREAM and CEMonitor was constrained by these requirements, which also affected many architectural decisions which were made. The main (non functional) requirements can be summarized as follows:

R1. *Expose a Web Service interface.* One of the cornerstones defined in [14] is the adoption of the Service Oriented Architecture (SOA) paradigm. According to the SOA, a complex software system should be realized as a collection of loosely coupled components, each one providing a specific service. The SOA paradigm can be implemented in different ways, the most common one being through Web Services technologies based on the family of XML languages. Thus, CREAM and CEMonitor expose Web Service interfaces, defined using the Web Service Description Language (WSDL) notation; details are given in Sections 4 and 5. Despite the fact that processing XML messages implies a considerable overhead with respect to ad-hoc binary message protocols, this overhead was considered acceptable for this particular use case.

R2. *Authentication based on VOMS proxies.* Authentication in gLite is currently based on the Public Key Infrastructure (PKI) using GSI proxies with VOMS extensions [15]. For this reason, CREAM and CEMonitor (like every other component in gLite) must support authentication and authorization based on VOMS proxies. Section 3.3 contains more details on the security infrastructure.

R3. *Support for proper credential delegation.* The gLite architecture heavily relies on the mechanism of *delegation* to securely transfer user's credentials to a service. The *delegated* service can then act on behalf of the user as long as the delegation remains valid (see Section 3.3). Therefore, CREAM supports credential delegation, because it needs to access input and output data files stored on remote locations. Note that credential delegation is computationally heavy, and if abused can greatly reduce the submission throughput to a CREAM CE (see Section 8 for performance considerations).

R4. *Support for multiple batch systems* The Compute Element must support multiple different batch systems (LSF, PBS/Torque, and others).

R5. *Provide an asynchronous notification service for job status changes.* Explicit polling of jobs on the execution service is not always efficient, especially when a large number of jobs are in execution on the underlying batch system. This motivated the development of CEMonitor, which can be coupled with CREAM to notify users each time one of their jobs changes status. CEMonitor is described in Section 5.

R6. *Ability to operate as stand-alone components.* CREAM and CEMonitor were developed to be usable also as stand-alone components, that is, outside the gLite WMS service. This proved to be a wise decision, as the recent trend in the area of Grid middleware development is to assemble middlewares from interoperable components [16]. Section 8 reports the experience of the ALICE high energy physics experiment which is uging CREAM in stand-alone mode, that is, without the gLite WMS.

R7. *Minimum performance and reliability requirements*: the EGEE deployment team defined a set of minimum performance and reliability requirements for the CE which had to be satisfied before the certification process could start. According to these requirements, the CE must handle 5000 simultaneously running jobs submitted from at least 50 different users. The CE must handle the abovementioned load for one month, during which it must run unattended without significant performance degradation. The acceptable job failure rate due to the CE must be less than 0.1% over a period of one month. Performance results are described in Section 8.

In order to reduce the development effort, CREAM and CEMonitor rely on some libraries and software packages developed by the EGEE collaboration. In particular, CREAM use the gLite *delegation service* to implement

credential delegation (needed by requirement R3), and support for multiple batch systems (needed by requirement R4) is provided by Batch-system Local ASCII Helper (BLAH). Additional security components used by CREAM are LCAS, LCMAPS and `glExec`, described in Section 3.3.

3.2. Deployment Layout

Fig. 2 shows the typical deployment of a Computing Element based on CREAM and CEMonitor. Both applications run as Java-Axis servlets [17] in the Tomcat application server [1]. Requests to CREAM and CEMonitor traverse a pipeline of additional components which take care of authorization issues; one of these components is the *Authorization Framework*, which is an Axis plugin for validating and authorizing the requests received by the services (more details on the security infrastructure will be given shortly).

CREAM uses an external relational database to store its internal state. This improves fault-tolerance as it guarantees that this information is preserved across restarts of CREAM. Moreover, the use of a SQL database improves responsiveness of the service while performing queries which are needed by the normal CREAM operations, such as getting the list of jobs associated with a specific user. The database, which is associated to a single CREAM instance, is accessed through the JDBC interface; in the gLite deployment we are using MySQL [18], but any database accessible through JDBC is supported. Note that the database server can be installed on a dedicated host, as shown in Fig. 2, or can share the same machine as CREAM and CEMonitor.

CREAM interacts with CEMonitor [13] to provide an asynchronous job status notification service. For each job status change, CREAM notifies CEMonitor, which in turn check whether there are subscriptions registered for that notification. If so, the notification is sent to the user which requested that. Further information on CEMonitor will be given in Section 5. Note that it is also possible to use CREAM without CEMonitor, for example if CREAM is installed behind a firewall which blocks outbound connections. In this case, of course, asynchronous job status change notifications are not available.

CREAM can be associated to multiple batch queues (note the one-to-many association shown in Fig. 2). CREAM submits requests to the LRMS through BLAH [19], an abstraction layer providing a unified interface to the underlying LRMS. BLAH, in turn, interacts with the client-side LRMS environment, which might consist of a set of command line tools which interact

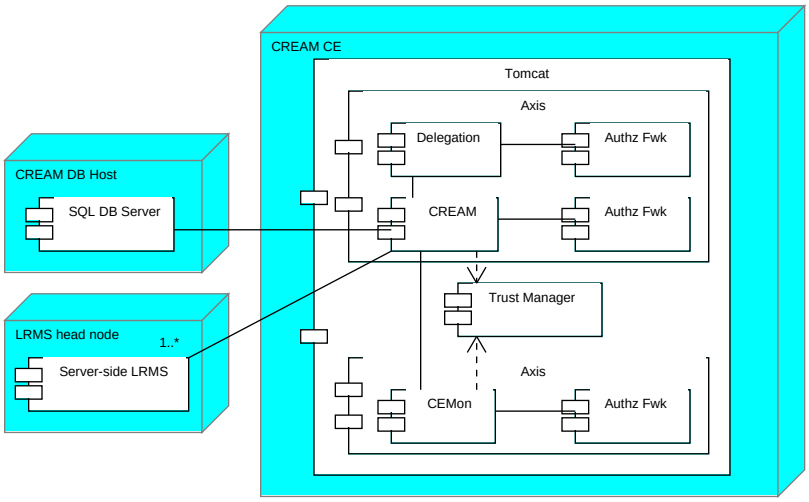

Figure 2: Typical deployment of a CREAM service

with the server-side LRMS. At the time of writing BLAH supports LSF, PBS/Torque, and Condor [20]; support for Sun Grid Engine (SGE) is currently being implemented as well. It is also possible to create other, ad-hoc connectors to interact with other types of batch systems. Note that a single instance of CREAM can provide access to multiple underlying LRMS.

3.3. Security

The Grid is a large collaborative resource-sharing environment. Users and services cross the boundaries of their respective organizations and thus resources can be accessed by entities belonging to several different institutions. In such a scenario, security issues are of particular relevance. There exists a wide range of authentication and authorization mechanisms, but Grid security requires some extra features: access policies are defined both at the level of Virtual Organizations (VOs) and at the level of single resource owners. Both these aspects must be taken into account. Moreover, as we will see in the following, Grid services have to face the problem of dealing with the delegation of certificates and the mapping of Grid credentials into local batch system credentials.

Trust Manager. The Trust Manager is the component responsible for carrying out authentication operations. It is external to CREAM and CEMonitor, and is an implementation of the J2EE security specifications [21].

Authentication is based on PKI. Each user (and Grid service) wishing to access CREAM or CEMonitor is required to present an X.509 format certificate [22]. 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. One interesting challenge in a Grid environment is the so-called *proxy delegation*. It may be necessary for a job running on a CE to perform some operations on behalf of the user owning the job. Those operations might require proper authentication and authorization support. For example, we may consider the case where a job running on a CE has to access a Storage Element (SE) to retrieve or upload some data. This aim is achieved in the Trust Manager using *proxy certificates*. RFC3820 proxy certificates are an extension of X.509 certificates [23]. The generation of a proxy certificate is as follows. If a user wants to delegate her credential to CREAM, she has to contact the *delegation Port-type* of the service. CREAM creates a public-private key pair and uses it to generate a Certificate Sign Request (CSR). This is a certificate that has to be signed by the user with her private key. The signed certificate is then sent back to CREAM. This procedure is similar to the generation of a valid certificate by a CA and, in fact, in this context the user acts like a CA. The certificate generated so far is then combined with the user certificate, thus forming a chain of certificates. The service that examines the proxy certificate can then verify the identity of the user that delegated its credentials by unfolding this chain of certificates. Every certificate in the chain is used to verify the authenticity of the certificate at the previous level in the chain. At the last step, a CA certificate states the identity of the user that first issues the delegated proxy.

Authorization Framework. The aim of the authorization process is to check whether an authenticated user has the rights to access services and resources and to perform certain tasks. The decision is taken on the basis of policies that can be either local or decided at the VO level. Administrators need a tool that allows them to easily configure the authorization system in order to

combine and integrate both these policies. For this reason, CREAM adopts a framework that provides a light-weight, configurable, and easily deployable policy-engine-chaining infrastructure for enforcing, retrieving, evaluating and combining policies locally at the individual resource sites. The framework provides a way to invoke a chain of policy engines and get a decision result about the authorization of a user. The policy engines are divided in two types, depending on their functionality. They can be plugged into the framework in order to form a chain of policy engines as selected by the administrator in order to let him set up a complete authorization system. A policy engine may be either a Policy Information Point (PIP) or a Policy Decision Point (PDP). PIPs collect and verify assertions and capabilities associated with the user, checking her role, group and VO attributes. PDPs may use the information retrieved by a PIP to decide whether the user is allowed to perform the requested action, whether further evaluation is needed, or whether the evaluation should be interrupted and the user access denied. In CREAM both VO and "ban/allow" based authorizations are supported. In the former scenario, implemented via the VOMS PDP, the administrator can specify authorization policies based on the VOs the jobs' owners belong to (or on particular VO attributes). In the latter case the administrator of the CREAM-based CE can explicitly list all the Grid users (identified by their X.509 Distinguished Names) authorized to access CREAM services. For what concerns authorization on job operations, by default each user can manage (e.g. cancel, suspend, etc.) only her own jobs. However, the CREAM administrator can define specific "super-users" who are empowered to manage also jobs submitted by other users.

Credential Mapping. The execution of user jobs in a Grid environment requires isolation mechanisms for both applications (to protect these applications from each other) and resource owners (to control the behavior of these arbitrary applications). In the absence of solutions based on the virtualization of resources (VM), CREAM implements isolation via local credential mapping, exploiting traditional Unix-level security mechanisms like a separate user account per Grid user or per job. This Unix domain isolation is implemented in the form of the `gLExec` system [24], a sudo-style program which allows the execution of the user's job with local credentials derived from the user's identity and any accompanying authorization assertions. This relation between the Grid credentials and the local Unix accounts and groups is determined by the Local Credential MAPping Service (LCMAPS) [25]. `gLExec`

also uses the Local Centre Authorization Service (LCAS) to verify the user proxy, to check if the user has the proper authorization to use the `gLExec` service, and to check if the target executable has been properly "enabled" by the resource owner.

4. The CREAM service

The main functionality of CREAM is job management. Users submit jobs described as a JDL expression [26], and CREAM executes it on an underlying LRMS (batch system). The JDL is a high-level, user-oriented notation based on Condor classified advertisements (classads) [10] for describing jobs and their requirements. CREAM uses a JDL dialect which is very similar to the one used to describe jobs in the gLite WMS. There are however some differences between the CREAM and WMS JDL, which are motivated by the different role of the job execution and workload management services. As described in Section 2, the gLite WMS receives job submission requests which possibly include a set of user-defined requirements, which are used by the WM to select the CE where the job is executed. Of course, once the selection is done, there is no need for the CE to further process the job requirements as they are no longer relevant. Similarly, there are other kind of information which only make sense for the CREAM JDL, and not for the WMS JDL.

CREAM supports the execution of batch (normal) and parallel (MPI) jobs. Normal jobs are single or multithreaded applications requiring one CPU to be executed; MPI jobs are parallel applications which usually require a larger number of CPUs to be executed, and which make use of the MPI library for interprocess communication.

As already introduced in Section 2, applications executed by CREAM might request a set of input data files to process (ISB), and might produce a set of output data files (OSB). CREAM transfers the ISB to the executing node from the client node and/or from Grid storage servers. The ISB is staged in before the job is allowed to start. Similarly, files belonging to the OSB are automatically transferred out of the execution node when the job terminates.

As an example, consider the following JDL processed by CREAM:

```
[
  Type = "job";
```

```
  JobType = "normal";
  Executable = "/sw/command";
  Arguments = "60";
  StdOutput = "sim.out";
  StdError = "sim.err";
  OutputSandbox = {
    "sim.err",
    "sim.out"
  };
  OutputSandboxBaseDestURI = "gsiftp://se1.pd.infn.it:5432/tmp";
  InputSandbox = {
    "file:///home/user/file1",
    "gsiftp://se1.pd.infn.it:1234/data/file2",
    "/home/user/file3",
    "file4"
  };
  InputSandboxBaseURI = "gsiftp://se2.cern.ch:5678/tmp";
]
```

With this JDL a *normal* (batch) job will be submitted. Besides the specification of the executable `/sw/command` (which must already be available in the file system of the executing node, since it is not listed in the ISB), and of the standard output/error files, it is specified that the files `file1`, `file2`, `file3`, `file4` will have to be transferred to the executing node as follows:

- `file1` and `file3` will be copied from the client UI file system
- `file2` will be copied from the specified GridFTP server (`gsiftp://se1.pd.infn.it:1234/data/file2`)
- `file4` will be copied from the GridFTP server specified by the `InputSandboxBaseURI` JDL attribute (`gsiftp://se2.cern.ch:5678/tmp`)

It is also specified that the files `sim.err` and `sim.out` (specified by the `OutputSandbox` attribute) must be automatically uploaded into `gsiftp://se1.pd.infn.it:5432/tmp` when the job completes its execution.

The pre- and post-staging of data is handled by a shell script, called Job Wrapper (JW), which is what is actually sent for execution on the LRMS. As the name suggests, the script "wraps" the executable by taking care of fetching external data, then calling the executable and finally putting the output

data to the correct remote locations. The JW is assembled by CREAM according to the JDL and sent to the LRMS.

Other typical job management operations (job cancellation, job status with different levels of verbosity and filtering, job listing, job purging) are supported. Moreover users are allowed to suspend and resume jobs submitted to CREAM-based CEs, provided that the underlying LRMS supports this feature.

For what concerns security, authentication (implemented using a GSI based framework [7]) is properly supported in all operations. Authorization on the CREAM service is also implemented, supporting both VO based policies and policies specified in terms of individual Grid users. A Virtual Organization is a concept that supplies a context for operation of the Grid that can be used to associate users, their requests, and a set of resources. CREAM interacts with the VO Membership Service (VOMS) [15] to manage VOs; VOMS is an attribute issuing service which allows high-level group and capability management and extraction of attributes based on the user's identity. VOMS attributes are typically embedded in the user's proxy certificate, enabling the client to authenticate as well as to provide VO membership and other evidence in a single operation.

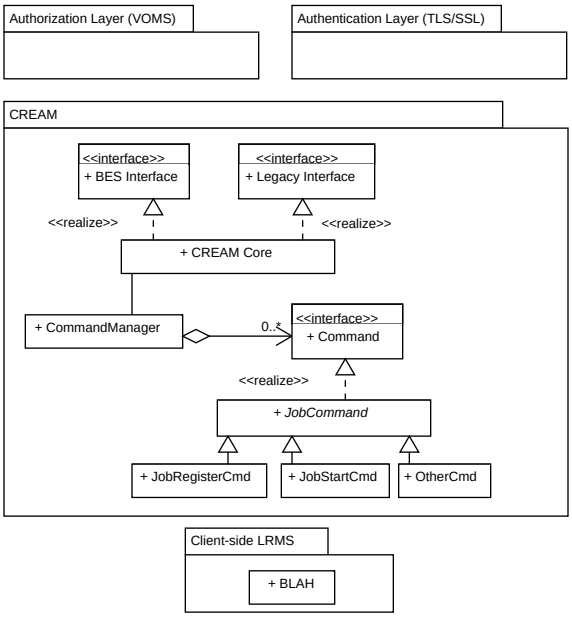

Figure 3: CREAM internal architecture

Fig. 3 shows the (simplified) internal structure of CREAM. As can be seen, CREAM exposes two different Web Service interfaces, which are shown in Fig. 3: a legacy interface, and a Basic Execution Service (BES)-compliant one. The operations of the legacy interface are listed in Table 1.

Lease Management Operations	
SetLease	Creates a new lease, or renews an existing lease
GetLease	Gets information on a lease with given ID
JobSetLeaseId	Associates a lease with a job
GetLeaseList	Gets the list of all active leases
DeleteLease	Deletes a lease, and purge all associated jobs
Job Management Operations	
JobRegister	Registers a new job for future execution
JobStart	Starts execution of a registered job
JobCancel	Request terminates a job
JobPurge	Purges all information of a job
JobSuspend	Suspends execution of a running job
JobResume	Resumes execution of a suspended job
JobStatus	Gets the status of a job
JobInfo	Gets detailed information about a job
JobList	Gets the list of all active jobs
Service Management Operations	
acceptNewJobSubmissions	Enables/disables new job submissions
getServiceInfo	Gets general information about the service

Table 1: CREAM interface operations

The first group of operations (*Lease Management*) allows the user to define and manage leases associated with jobs. When job submissions arrive through the gLite WMS, it is essential that all jobs submitted to CREAM eventually reach a terminal state (and thus eventually get purged from the CREAM server), even in cases when CREAM can no longer be contacted due to network partitioning. The gLite WMS has been augmented with an additional component, ICE, which is responsible for interacting with CREAM. ICE and CREAM use a lease-based protocol to ensure that all jobs get eventually purged by CREAM. Each job submitted through ICE has an asso-

ciated *lease time*, which must be periodically renewed using the *JobLease* CREAM operation. ICE is responsible for renewing the leases associated to active jobs, i.e. jobs which are not yet terminated. Should a lease expire before the actual termination of a job, CREAM will purge all jobs associated with that lease and free all the CE resources used by them.

The second group of operations (*Job Management*) is related to the core functionality of CREAM as a job management service. Operations are provided to create a new job, start execution of a job, suspend/resume or terminate a job. Moreover, the user can get the list of all owned jobs, and it is also possible to get the status of a set of jobs. The CREAM job state model is shown in Fig. 4, and job states are described in Table 2.

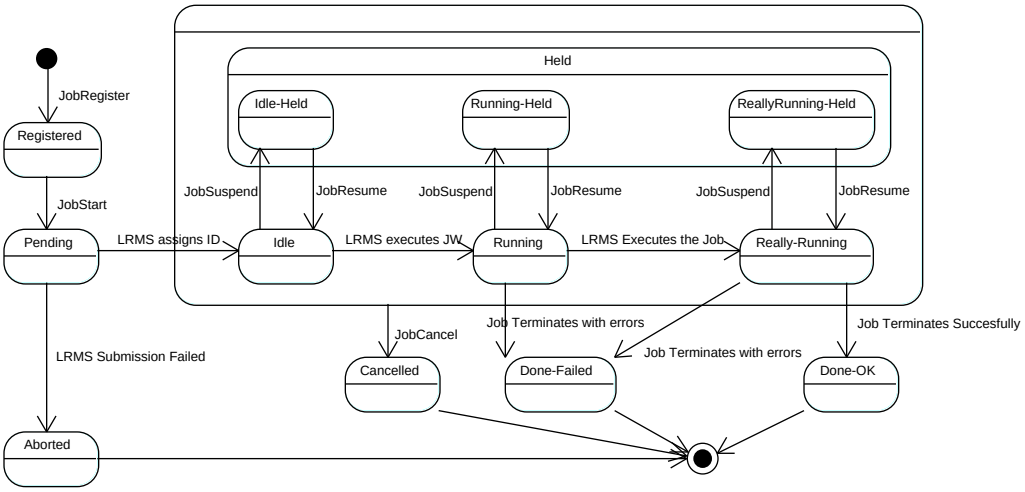

Figure 4: CREAM job states

Finally, the third group of operations (*Service Management*) deals with the whole CREAM service. It consists of two operations, one for enabling/disabling new job submissions, and one for accessing general information about the service itself. Note that only users with administration privileges are allowed to enable/disable job submissions.

Recently we implemented an additional interface to the CREAM service, compliant with the Basic Execution Service (BES) specification. BES [27] defines a standard interface for execution services provided by different Grid systems. The aim of BES is to favor interoperability of computing elements between different Grids: the same BES-enabled CE could be "plugged" into

Registered	The job has been submitted to CREAM with the *JobRegister* operation
Pending	The user invoked the *JobStart* operation to start the job execution
Idle	The LRMS (batch system) accepted the job for execution. The job is now in the LRMS queue
Running	The Job Wrapper is being executed
Really-Running	The actual user job is being executed
Held	The job has been suspended, e.g. because the user issued the *JobSuspend* operation. The job can be resumed in its previous state with the *JobResume* operation
Done-OK	The job terminated correctly
Done-Failed	The job terminated with errors
Cancelled	The job has been cancelled, e.g. because the user invoked the *JobCancel* operation to terminate it
Aborted	Submission to the LRMS failed

Table 2: Description of the CREAM job states

any compliant infrastructure. BES defines basic operations for job submission and management. More specifically, the BES specification defines two Web Services *port-types*: *BES-Factory*, containing operations for creating, monitoring and controlling sets of jobs, and *BES-Management*, which allows clients to monitor the details of and control the BES itself. The Port-types and associated operations are shown in Table 3.

BES uses the Job Submission Description Language (JSDL) [28] as the notation for describing computational jobs. The legacy CREAM interface was defined before BES was available, and also provides additional methods which are not provided by BES (notably, the possibility to renew a user proxy certificate, which is useful to avoid user proxy expiration while a job is running). The BES interface for CREAM uses a different security mechanism, which is based on Security Assertion Markup Language (SAML) assertions [29]. It should be observed that there are currently no production users of the BES/JSDL/SAML interface for CREAM; we consider the current BES and JSDL specifications too limited to be usable in production [30], so we are putting effort in improving these specifications within the Open Grid Forum (OGF) community, rather than support them as they are now.

BES-Management Port-type	
StartAcceptingNewActivities	Administrative operation: requests that the BES service start accepting new activities
StopAcceptingNewActivities	Administrative operation: requests that the BES service stop 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 termination of a set of activities
GetActivityDocuments	Requests the JSDL document for a set of activities
GetFactoryAttributeDocument	Requests the XML document containing the properties of this BES service

Table 3: BES Port-Types and Operations

CREAM can be seen as an abstraction layer on top of an LRMS (batch system), which extends the LRMS capabilities with an additional level of security, reliability, and integration with a Grid infrastructure. CREAM supports different batch systems (requirement R4 on Section 3.1) through the concept of *LRMS connectors*. An LRMS connector is an interface for a generic batch system. Currently, CREAM supports all the batch systems supported by BLAH [19] through a specific instance of LRMS connector called the *BLAH connector module*.

CREAM has been developed around an internal core, which is a generic command executor. The core accepts abstract commands which are enqueued and executed by a pool of threads. It is possible to customize the core by defining concrete implementations of the abstract command interface. Two kind of commands can be defined: *synchronous* and *asynchronous*. Synchronous commands must be executed immediately upon receipt, while asynchronous command execution can be deferred at a later time. Moreover, it is possible to define *sequential* or *parallel* commands. When a parallel command is being executed, other commands (parallel or sequential) can be concurrently executed by other threads in the pool. When a sequential com-

mand is being executed, no other commands operating on the same job are executed by any other thread, until the sequential command terminates. The job management interfaces (both the BES and the legacy one) instantiate the correct command type to execute the operations requested by the users.

During the development of CREAM, several design decisions were made in order to cope with the main limitations of job management systems: a single operation can take a significant amount of time to complete (depending on the number of running jobs and/or the kind of underlying LRMS), so that clients are prone to experience broken connections due to timeouts. This problem is *essential* rather than *accidental* (using the terminology of Brooks [31]) because it is dependent on the underlying LRMS. In order to cope with this, the following design decisions were applied:

- *Process user requests asynchronously.* Given that a single job management operation can, in the worst case, take tens of seconds to complete, the methods exposed by the CREAM interface return as soon as the appropriate LRMS operation has been scheduled for execution. For example, if the *JobCancel* operation returns succesfully, it does *not* mean that the job has been cancelled, but only that the appropriate LRMS cancel operation has been scheduled. Actual cancellation might require a longer time, and might even fail eventually due to LRMS internal reasons. After succesfully issuing the *JobCancel* operation, the client must either check with *JobStatus* if the job has actually been terminated, or wait to receive an appropriate asynchronous status change notification.

- *Bulk operations.* Job management usually involves users sending hundreds of jobs to a single CE. Sometimes the client wants to execute the same operation on many jobs, e.g., cancel all running jobs, check the status of all running jobs and so on. Issuing a single command for each job is inefficient, so CREAM supports *bulk commands*. Most of the operations shown in Table 1 accept a list of job IDs as input, and apply the same operation to all jobs whose ID appears in the list. If the underlying batch system supports bulk operations as well, BLAH issues a single bulk command to the LRMS, otherwise multiple individual commands are sent. Asynchronous command execution is especially important in the case of bulk commands, because their completion is likely to require a much longer than the SOAP connection timeout.

- *Favor notifications over polling.* Querying the status of a large num-

ber of jobs is a particularly slow operation; unfortunately, it is also one of the most frequently invoked ones, so it must be supported efficiently. CREAM addresses this problem in two ways. The first is to rely on BLAH for receiving status change notifications from the LRMS. BLAH can parse the LRMS log files to get status changes without using the (usually slow) command line tools provided by the batch system. CREAM stores all status changes for each job in its internal SQL database, so a *JobStatus* or *JobInfo* operation only involve a SQL query. The second is to provide users with an asynchronous job status notification system provided by CEMonitor (see Section 5).

- *Master-Worker paradigm.* CREAM, as any other job management service, must be able to accept and process multiple commands in parallel. In order to do so, command execution is delegated to a pool of worker threads. If different commands are related to different LRMSs, they can be actually executed in parallel, reducing the response time as observed by clients.

5. The CEMonitor Service

The purpose of CEMonitor is to provide an asynchronous event notification framework, which can be coupled with CREAM to notify the users when job status changes occur.

Figure 5 shows the internal structure of the CEMonitor service. Similarly to CREAM, CEMonitor is a Java application which runs in an Axis container within the Tomcat application server. CEMonitor uses the same authentication/authorization mechanisms as CREAM, which has been discussed in Section 3. The operations supported by CEMonitor are shown in table 4.

CEMonitor publishes information as *topics*. For each topic, CEMonitor maintains the list of *events* to be notified to users. Topics can have three different levels of *visibility*: *public*, meaning that everybody can receive events associated with the topic; *group*, meaning that only member of a specific VO can receive notifications; and *user*, meaning that only the user which created the topic can receive notifications. Users can create *subscriptions* for topics of interest. Each subscription has a unique ID, an expiration time and an update frequency f. CEMonitor checks every $1/f$ seconds whether there are new events for the topic associated to the subscription; if so, the events are sent to the subscribed users. Unless a subscription is explicitly renewed by

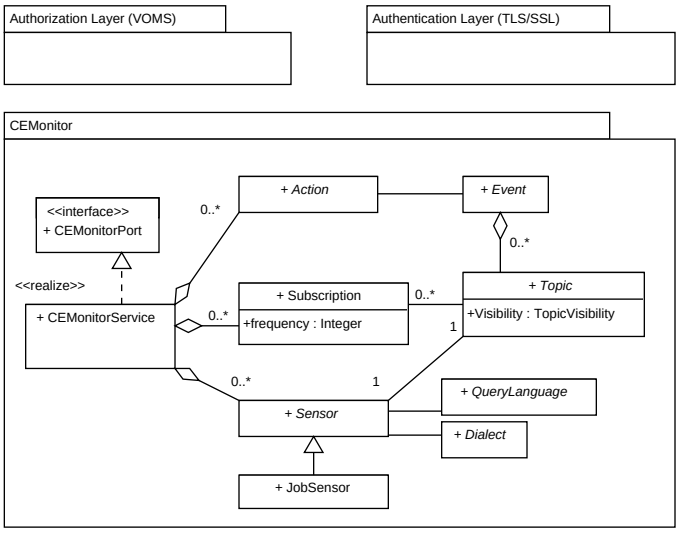

Figure 5: Internal structure of CEMonitor

its creator, it is removed after the expiration time and no more events will be notified.

Each topic is produced by a corresponding *sensor*. A sensor is a component which is responsible for actually generating events to be notified for a specific topic. Sensors can be plugged at runtime: when a new sensor is added, CEMonitor automatically instantiates the corresponding topic users can subscribe to. The most important sensor we currently use is called *JobSensor*, which fires an event for each job status changes. When CREAM detects that a job changes its status (for example, an Idle job starts execution, thus becoming Running), it notifies the JobSensor by sending a message on the network socket where the sensor is listening. Then, the JobSensor triggers a new notification which is eventually sent to all subscribed users.

Each sensor can provide either asynchronous notifications to registered listeners, or can be queried synchronously. In both cases, sensors support a list of so-called *query languages*. A query language is a notation (e.g., XPath, classad expressions and so on) which can be used to ask a sensor to provide only events satisfying a user-provided condition. When an event satisfies a condition, CEMonitor triggers an *action* on that event. In most cases, the action simply instructs CEMonitor to send a notification to the user for that event. Of course, it is possible to extend CEMonitor with additional types of user-defined actions. When registering for asynchronous notifications with

the *Subscribe* operation (see Table 4), the user passes a query expressed in one of the supported query languages as parameter. For that subscription, only events matching the query are notified.

Sensors support different *dialects*. A dialect is a specific output format which can be used to render events. This means that a sensor can publish information in different formats (e.g., job status change information could be made available either in Condor classad format [10], or in XML format). When a user subscribes to a topic, she can also specify an appropriate dialect for rendering the notifications. CEMonitor will then apply the correct rendering before sending the notifications.

We show in Fig. 6 an example of job status change notification. The notification is in Condor classad format, and contains a set of attributes with their associated values. `CREAM_JOB_ID` is the ID of the job which changed status; `CREAM_URL` is the endpoint of the CREAM service where the job is being executed; `JOB_STATUS` is the current job status (in human-readable format); `TIMESTAMP` represents the time (in seconds since epoch) when the job status change happened; `WORKER_NODE` is the name of the execution host for the job. In this case, the job has not started execution yet, so the information on the worker node is reported as not available. Figure 7 shows an XML rendering of the same information.

```
[
  CREAM_JOB_ID = "CREAM986407854";
  CREAM_URL = "https://cream-02.pd.infn.it:8443/ce-cream/services/CREAM2";
  JOB_STATUS = "REGISTERED";
  TIMESTAMP = "1232444196000";
  WORKER_NODE = "N/A"
]
```

Figure 6: Job status change notification in classad Dialect

It must be stressed that CEMonitor is not strictly coupled with CREAM. It is instead a generic framework for information gathering and provisioning. For example in the context of the Open Science Grid (OSG) ReSS project is used to manage Grid resource information [32].

```
<status>
  <cream_job_id>CREAM986407854</cream_job_id>
  <cream_url>
    https://cream-02.pd.infn.it:8443/ce-cream/services/CREAM2
  </cream_url>
  <job_status>REGISTERED</job_status>
  <timestamp>1232444196000</timestamp>
  <worker_node>N/A</worker_node>
</status>
```

Figure 7: Job status change notification in XML Dialect

6. Putting the components together

In this section we summarize the interactions between ICE and CREAM/CEMonitor with the UML Sequence Diagram shown in Fig. 8. The same kind of interaction can be performed by a generic client submitting jobs directly to CREAM (i.e., without using the gLite WMS).

The relevant messages shown in the diagram are as follows:

1. ICE invokes the *getProxyReq* operation on the Delegation service. The request parameter is a string which represents the delegation ID which will be associated to the delegated credentials.
2. The delegation service replies with a Certificate Sign Request (CSR), which is a RFC3280 style proxy certificate request in PEM format with Base64 encoding [22].
3. ICE signs the CSR on behalf of the user which originally submitted the job. This is possible because ICE itself is using a user proxy certificate which has been delegated to the WMS. Then, ICE sends back: the ID of the delegation session initiated on step 1 and the RFC3280 style proxy certificate, signed by ICE on behalf of the user, in PEM format with Base64 encoding.
4. The Delegation service transfers the delegation ID/signed proxy to CREAM. Note that both CREAM and the delegation service execute on the same physical host, so they can communicate locally.
5. ICE requests the creation of a new lease, with a given lease ID. ICE maintains a single lease for each user submitting jobs, so there are as many lease IDs as the number of unique users submitting to a specific CREAM CE.

Service Management Operations	
GetInfo	Gets information about the CEMonitor service, including the version and a brief description of the service, plus a list of available Topics and Actions.
Lease Management Operations	
Subscribe	Subscribes for notifications. The user specifies the topic, a query to be executed and a set of actions to trigger when the Query succeeds. The notification rate can also be specified as parameter.
Update	Updates an existing Subscription: it is possible to modify the topic, query, triggered actions and/or notification rate.
GetSubscriptionRef	Gets the list of all subscription IDs and associated expiration times belonging to the caller.
GetSubscription	Gets detailed information on a set of subscriptions given their unique IDs.
Unsubscribe	Removes an existing subscription. Events associated to that subscription will no longer be notified.
PauseSubscription	Pauses the stream of notifications associated with a given subscription ID.
ResumeSubscription	Resumes sending notifications associated with a previously paused subscription.
GetTopics	Gets the list of Topics supported by CEMonitor.
GetTopicEvent	Gets the list of events associated with the specified Topic.

Table 4: CEMonitor interface operations

6. ICE is now ready to submit jobs to CREAM using the existing delegation ID and lease ID. The first step is to invoke the *JobRegister* CREAM operation: this operation prepares the job for execution, by first creating some temporary files for internal use on the CE host.
7. The CREAM service registers the job, creates all the temporary files and returns a CREAM job ID which can be used from now on to refer to this job.
8. ICE invokes the *JobStart* operation, using the CREAM job ID as parameter, to request that the job is actually transferred to the LRMS, and to request that execution begins.

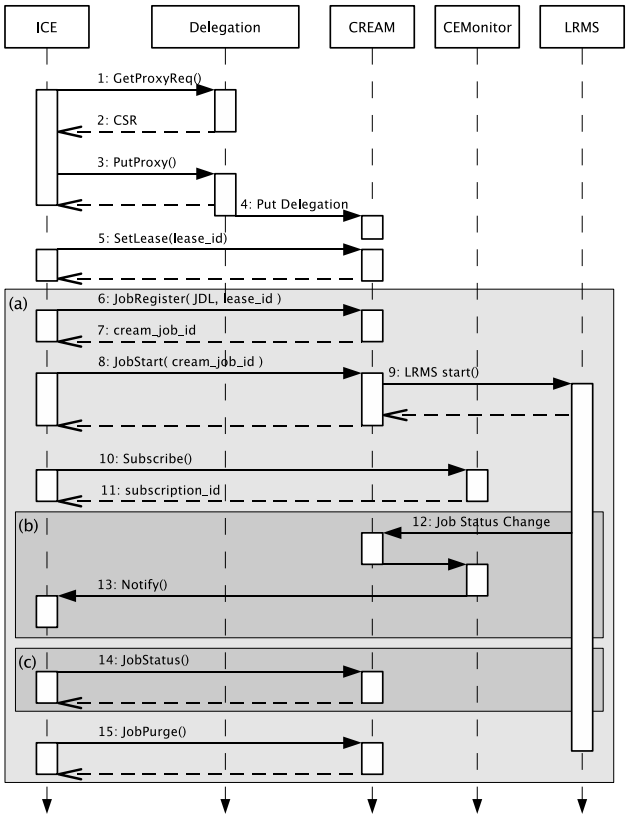

Figure 8: Overall job submission sequence diagram

9. CREAM forwards the job to the LRMS; the job is added to the LRMS batch queue, and will eventually be executed.
10. ICE subscribes to CEMonitor to receive job status change notifications. This is done only if there are no active subscriptions on that specific CREAM CE; if so, there is no need to create a new subscription, as it is possible to use the existing one.
11. CEMonitor returns a Subscription ID, which can be used later on to renew, modify or cancel the subscription.
12. The LRMS, through BLAH (see Section 4), notifies CREAM about each job status change. CREAM in turn informs CEMonitor.
13. CEMonitor notifies ICE a job status change; note that, in order to reduce round-trip times, CEMonitor batches multiple related notifications which are sent together to subscribed clients.
14. ICE periodically queries the job states directly to the CREAM service

using the *JobStatus* operation.

15. When the job terminates, ICE invokes the *JobPurge* operation to remove all temporary files which have been created on the CE node.

We remark that it is sufficient to perform a single delegation operation and to create a single lease for each user. So, after the first job has been submitted, all subsequent submissions for the same user require only the interactions shown in box (a) of Fig. 8. The interactions in box (b) are executed whenever CEMonitor notifies new job status changes. Finally, the interactions shown in box (c) are executed only when ICE does not receive status change notifications for some jobs for longer than a configurable threshold.

We omitted from Fig. 8 the operations required to renew the delegations when they are about to expire, and to renew the leases when they are about to expire. Delegation renewal involves exactly the same operations required for delegating credentials for the first time (operations 1 through 4 in the sequence diagram); lease renewal is performed by calling *SetLease* with an existing lease ID, as in operation 5 in the diagram.

7. Build, Installation and Usage

All the components of the gLite middleware (including CREAM and CEMonitor) are built using the ETICS Build and Test facility [33]. ETICS is an integrated system for the automated build, configuration, integration and testing of software. Using ETICS it is possible to integrate existing procedures, tools and resources in a coherent infrastructure, additionally providing an intuitive access point through a Web portal. The ETICS system allows developers to assemble multiple components, each one being developed independently, into a coherent software release. Each software component can use its own build method (e.g., Make for C/C++ code, Ant for Java code and so on), and ETICS provides a wrapper around that so that components or subsystems can be checked out and built using a common set of commands. The ETICS system can automatically produce and publish installation packages for the components it builds; multiple target platforms can also be handled.

CREAM and CEMonitor are included in the gLite 3.1 software distribution, which is provided as a set of different deployment modules (also called *node types*) that can be installed separately. CREAM and CEMonitor are installed and configured together as one of these modules, called `creamCE`. For what concerns the installation, the main supported platform, at present, is

CERN Scientific Linux 4 (SLC4), 32-bit flavor; porting the whole gLite stack to CERN Scientific Linux 5 (64 bit) is underway. For the SLC4 platform, the gLite `creamCE` module is available in RPM [34] format and the recommended installation method is via the gLite *yum* repository. For what concerns the configuration, there exists a manual configuration procedure, and a gLite compliant configuration tool also exists. The tool adopted to configure gLite Grid Services is YAIM (YAIM Ain't an Installation Manager) [35]. YAIM provides simple configuration methods that can be used to set up uniform Grid sites. YAIM has been implemented as a set of bash scripts: it supports a component based model with a modularized structure including a YAIM core component, common to all the gLite middleware software, supplemented by component specific modules, all distributed as RPMs. For CREAM and CE-Monitor appropriate plugins for YAIM were implemented in order to get a fully automated configuration procedure.

8. Performance Considerations

We evaluate the performance of the CREAM service in term of throughput (number of submitted jobs/s), comparing CREAM with the LCG-CE currently used in the gLite middleware, considering the submission through the WMS. To do so, we submit 1000 identical jobs to an idle CE. The jobs are submitted using the credentials of four different users (each user submits 250 jobs).

The layout of the testbed is shown in Fig. 9. All jobs are submitted using a WMS UI installed on the host `cream-15.pd.infn.it` located at INFN Padova. We always use the gLite WMS UI (see Fig. 1) for submissions to both CREAM and the LCG-CE (that is, we do not use direct CREAM submission): the reason is that, at the moment, the vast majority of users are submitting jobs through the gLite WMS. The UI transfers the jobs to the WMS host `devel19.cnaf.infn.it` located at INFN CNAF in Bologna. The WMS submits jobs through ICE to the CREAM service running on `cream-21.pd.infn.it` located at INFN Padova. The JobController+CondorG+LogMonitor components of the WMS submit jobs to a LCG-CE running on `cert-12.pd.infn.it`, also located at INFN Padova. Both CREAM and the LCG-CE are connected to the same (local) batch system running the LSF batch scheduler.

We are interested in examining the submission rate from ICE and JC/CondorG/LM to CREAM and LCG-CE respectively; this is an HB

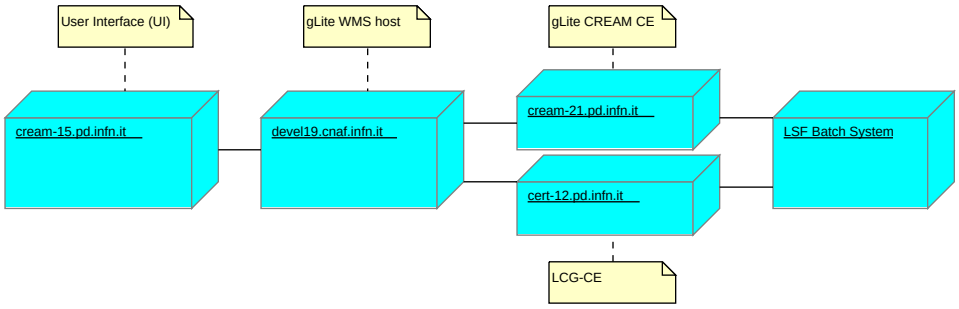

Figure 9: Layout of the testbed

(Higher is Better) metric, as higher submission rate denotes better performance. To compute the submission rate we consider the time elapsed since the first job is dequeued by ICE or JC from their respective input queues, to the time the last job has been successfully transferred to the batch system. Note that we do not take into consideration the time needed to complete execution of the jobs, as this time is independent from the CE.

In order to ensure that the transfer from the WMS UI to the WMS is not the bottleneck in our tests, we execute the following steps:

1. We switch off the ICE or JC component of the WMS;
2. We submit 1000 jobs from the WMS UI;
3. When all the jobs have been successfully transferred to the WMS node, we switch on ICE (or JC, depending on the kind of test we are performing). At this point ICE (or JC) finds all the jobs in its input queue, so what we measure here is the actual transfer rate from the WMS to the CE.

We analyze the impact of two factors on the submission throughput. The factors we consider are the following:

- Use of an *automatic proxy renewal* mechanism vs *no proxy renewal*. The automatic proxy renewal mechanism is normally used for long-running jobs, to ensure that the credentials delegated to the CE are automatically refreshed before expiration. Automatic proxy renewal works by first having the user register her credentials to a so-called *MyProxy Server*. The gLite WMS receives a "fresh" proxy from the MyProxy server, and ICE or JC+CondorG are responsible for delegating the new

	Proxy Renewal	**Delegation**	**Submission Rate (jobs/sec)**	
			CREAM/ICE	LCG-CE/JC+CondorG+LM
Test A	Disabled	Explicit	0.9624	0.3952
Test B	Disabled	Automatic	0.1660	0.3633
Test C	Enabled	Explicit	0.8976	0.3728
Test D	Enabled	Automatic	0.9191	0.3863

Table 5: Test results; higher (better) submission rates are shown underlined

credentials to the CE. We remark that no proxy is actually refreshed in our tests, since transfer of all jobs to the CE completes long before the user credentials expire. Nevertheless, the proxy renewal mechanism has an impact on the submission rate to CREAM via ICE, as will be explained later.

- Use of *automatic* vs *explicit delegation* (see Section 3.3). When *automatic* delegation is active, the WMS UI delegates a new proxy certificate to the WMS, which in turn delegates the proxy again to the CE, *for each job submitted to the CE*. Thus, a new delegation operation on the CE is executed before each submitted job. If *explicit* delegation is used, the user explicitly delegates a proxy before the first job is submitted, and uses the same delegation ID for all subsequent submissions. Thus, in this case only a single delegation operation is performed on the CE node.

We analyze four different scenarios with a total of 8 independent runs, corresponding to a 2^2 factorial design with two replications [36]; each test has been repeated two times, and the average of the measured submission rates is considered.

Table 5 shows the submission rates for all the experiments. We observe that the submission rates from JC+CondorG+LM to the LCG-CE remain more or less the same across the different experiments. On the other hand, submission rates from ICE to the CREAM CE are higher in three of our experiments, but incur a significant penalty in Test B.

The reason for this is in the different way in which CREAM/ICE and LCG-CE/JC+CondorG+LM implement the transfer of user credentials from the WMS to the CE node. As already described in section 3, CREAM exposes a delegation port-type to allow clients to securely delegate their credentials to

the CE. The delegation operation (steps 1–4 from Fig. 8) involves the creation on the server side of a public/private key pair, which takes a considerable amount of time. Explicit delegation (Test A and C) allows ICE to delegate only once for each user: in our tests, as we are submitting 250 jobs for each of 4 different users, only four delegation operations are performed, and this causes a significant improvement of the submission rate.

The JC+CondorG+LM does not implement a proper delegation operation, but *for each job* transfers the user credentials to the LCG-CE using a more lightweight mechanism. This explains why the submission rate achieved by LCG-CE/JC+CondorG+LM is more or less independent from the delegation mechanism used (automatic or explicit). The lack of delegation on the LCG-CE was one of the reasons why CREAM was developed, as credential transfer without proper delegation is no longer considered acceptable.

In Test D we have automatic delegation together with proxy renewal. This implies that *all* delegated user proxies are automatically renewed. Note that if the same user performs two delegations, the delegated credentials will expire on different times, and thus in general should be treated separately. However, if the proxy renewal mechanism is active, all delegations will be renewed before expiration, so from the user point of view all her credentials have duration equal to the duration of the proxy renewal mechanism. For this reason, in situations like Test D, ICE considers all proxies "equivalent" by performing a single delegation operation to CREAM for each user which requested automatic credentials renewal.

The CREAM based CE was also tested and used for real production activities. To assess the performance and the reliability of CREAM, and in particular to verify its usability in production environments, the Alice LHC experiment [37] performed some tests which took place during the summer of 2008. About 55000 standard production Alice jobs, each one lasting about 12 hours, were submitted on a CREAM based CE at the FZK[3] Tier-1 center. The CREAM service showed a remarkable stability: no failures were seen and no manual interventions were needed during the whole test period. It should be observed that Alice is using CREAM in stand-alone mode (i.e., using direct job submissions, bypassing the gLite WMS). For users which do not need the sophisticated matchmaking capabilities of the gLite WMS, it is

[3]Forschungszentrum Karlsruhe, now Karlsruher Institut für Technologie, `http://www.kit.edu/`

much more efficient to submit directly to CREAM. Doing so it is possible to bypass the intermediate steps shown in Fig. 1; furthermore, it is much easier to deploy a single CREAM server rather than a full WMS installation. When used outside the gLite middleware, CREAM provides access to multiple batch queues, using a Web Service interface, with a security layer based on PKI so that job submissions can happen from remote clients. CREAM clients can be written in any language with support for Web Services and related technologies (tools for generating stubs from WSDL interfaces exist for almost any programming language). CEMonitor provides an asynchronous notification service which is usually not provided by conventional batch system managers.

After this first successful assessment, the submission to CREAM based CREAM CEs has been fully integrated in the Alice *Alien* software environment [38]. Alice jobs are currently being submitted in about a dozen of CREAM CEs deployed in several sites of the EGEE Grid.

9. Conclusions

In this paper we described CREAM and CEMonitor, two software components which are used to implement a job execution and management service in the gLite middleware. CREAM manages submissions of jobs to a LRMS. CREAM provides additional features on the top of the underlying batch system, such as Grid-enabled user authentication and authorization and integration with the rest of the gLite infrastructure. CEMonitor is a general purpose event notification service, which can be coupled with CREAM to allow users to receive notifications about job status changes without polling the service.

CREAM and CEMonitor have been integrated into the gLite WMS using an additional component called ICE. ICE receives requests from the gLite WM, and handles all interactions with CREAM and CEMonitor. ICE takes care of delegating user credentials to CREAM, subscribing to CEMonitor for receiving job status change notifications, and actually submitting and monitoring jobs. ICE registers to the gLite LB service all status changes, such that Grid users know exactly the location and the status of their jobs.

CREAM and CEMonitor expose a Web Service interface, which allows easy interoperability with heterogeneous client applications. Recently, the Grid community is putting considerable effort in defining standard interfaces to Grid services. The reason for this interest is twofold: standard interfaces

allow different middlewares to easily share resources and services. Moreover, standard interfaces improve the software development cycle by allowing developers to import software components from other middleware stacks. For these reasons, we implemented a prototype support for the BES and JSDL specifications in CREAM [30]. It must be observed that these specifications, in their current status, are inappropriate for production use, as they only provide basic functionality. The JSDL specification is severely limited because it only allows users to describe simple (batch) jobs, while structured jobs such as collections of tasks with dependencies cannot be represented using the current JSDL. Furthermore, security considerations are outside the scope of the BES specification, which results in the possibility for different services to claim standard-compliance without being interoperable due to the use of mutually incompatible security settings. To address these problems, the Grid community is currently defining extensions of the BES and JSDL specifications within the Production Grid Infrastructure Working Group [4].

CREAM and CEMonitor have been deployed and are currently in production use in the gLite infrastructure of the EGEE project. Some of the larger sites have begun to experiment with usage scenarios which are beyond those which were foreseen in the original requirements. In particular, deployments were a single CREAM server manages a large batch system consisting of thousands of execution nodes pose a real challenge. We are currently considering new ways to improve the scalability of CREAM far beyond the levels defined by the requirements (see Section 3.1). One approach is to adopt clustered configuration, allowing multiple service instances to balance load and tolerate failures. However, as CREAM and CEMonitor are both stateful services, special care must be taken in order to guarantee that each instance shares the same internal status, while avoiding single points of failure. We are also investigating how some ideas from the *cloud computing* paradigm could be integrated into CREAM. In particular, we are considering the possibility of dynamically adjusting the size (number of hosts) of the underlying LRMS to allow the system to automatically scale whenever needed. This could be done, for example, by implementing a LRMS based on Amazon's EC2 service, such that the batch system pool could be dynamically increased by instantiating new virtual hosts.

[4] `http://forge.gridforum.org/sf/projects/pgi-wg`

Acknowledgments

EGEE-3 is a project funded by the European Union under contract INFSO-RI-222667.

References

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

[2] E. Laure, S. M. Fisher, Á. Frohner, C. Grandi, P. Kunszt, A. Krenek, O. Mulmo, F. Pacini, F. Prelz, J. White, M. Barroso, P. Buncic, F. Hemmer, A. Di Meglio, A. Edlund, Programming the Grid with gLite, Computational Methods in Science and Technology 12 (1) (2006) 33–45.

[3] Enabling Grid for E-sciencE (EGEE) project web site, `http://www.eu-egee.org/`.

[4] D. Kouřil, et al., Distributed tracking, storage, and re-use of job state information on the grid, in: Proceedings of CHEP'04, Interlaken, Switzerland, 2004.

[5] D. W. Erwin, UNICORE–a grid computing environment, Concurrency and Computation: Practice and Experience 14 (2002) 1395–1410. doi:`http://dx.doi.org/10.1002/cpe.691`.

[6] M. Ellert, M. Grønager, A. Konstantinov, B. Kónya, J. Lindemann, I. Livenson, J. Nielsen, M. Niinimäki, O. Smirnova, A. Wäänänen, Advanced resource connector middleware for lightweight computational grids, Future Generation Computer Systems 23 (2) (2007) 219–240. doi:`http://dx.doi.org/10.1016/j.future.2006.05.008`.

[7] I. Foster, Globus Toolkit Version 4: Software for Service-Oriented Systems, in: IFIP International Conference on Network and Parallel Computing, 2005, pp. 2–13.

[8] I. Foster, et al., Modeling Stateful Resources with Web Services, White paper, version 1.1, Available online at `http://www.ibm.com/developerworks/library/ws-resource/ws-modelingresources.pdf` (Mar. 5 2004).

[9] S. Burke, S. Campana, E. Lanciotti, P. M. Lorenzo, V. Miccio, C. Nater, R. Santinelli, A. Sciabà, gLite 3.1 User Guide–Manuals Series, Version 1.2, Document identifier CERN-LCG-GDEIS-722398. Available online at https://edms.cern.ch/document/722398/1.2 (Jan.7 2009).

[10] R. Raman, Matchmaking Frameworks for Distributed Resource Management, Ph.D. thesis, University of Wisconsin-Madison (2001).

[11] R. van Engelen, gSOAP 2.7.11 User Guide (Oct. 2 2008).

[12] P. Andreetto, et al., The gLite Workload Management System, Journal of Physics, Conference Series 119 (6) (2008) 062007 (10pp). doi:http://dx.doi.org/10.1088/1742-6596/119/6/062007.

[13] CEMonitor home page, http://grid.pd.infn.it/cemon.

[14] EGEE middleware architecture and planning (release 2), EU Deliverable DJRA1.4, https://edms.cern.ch/document/594698/1.0 (Jul. 15 2005).

[15] R. Alfieri, R. Cecchini, V. Ciaschini, L. dell'Agnello, Á. Frohner, K. Lőentey, F. Spataro, From gridmap-file to VOMS: managing authorization in a Grid environment, Future Generation Computer Systems 21 (4) (2005) 549–558. doi:http://dx.doi.org/10.1016/j.future.2004.10.006.

[16] M. Riedel, et al., Interoperation of world-wide production e-science infrastructures, Concurrency and Computation: Practice and Experience 21 (8) (2009) 961–990. doi:http://dx.doi.org/10.1002/cpe.1402.

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

[18] P. DuBois, MySQL, Addison-Wesley Professional, 2008.

[19] E. Molinari, et al., A local Batch System Abstraction Layer for Global Use, in: Proc. XV International Conference on Computing in High Energy and Nuclear Physics (CHEP'06), Mumbay, India, 2006.

[20] D. Thain, T. Tannenbaum, M. Livny, Distributed computing in practice: the Condor experience, Concurrency–Practice and Experience 17 (2-4) (2005) 323–356. doi:http://dx.doi.org/10.1002/cpe.v17:2/4.

[21] Sun Microsystems, Inc., Java^{TM}Platform Enterprise Edition, v5.0, API Specifications (2007).

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

[23] S. Tuecke, V. Welch, D. Engert, L. Pearlman, M. Thompson, RFC3820: Internet X.509 Public Key Infrastructure (PKI) Proxy Certificate Profile, http://www.ietf.org/rfc/rfc3820.txt (Jun. 2004).

[24] D. Groep, O. Koeroo, G. Venekamp, gLExec: gluing grid computing to the Unix world, Journal of Physics: Conference Series 119 (6) (2008) 062032 (11pp). doi:http://dx.doi.org/10.1088/1742-6596/119/6/062032.

[25] Site authorisation and enforcement services: LCAS and LCMAPS, http://www.nikhef.nl/grid/lcaslcmaps/.

[26] M. Sgaravatto, CREAM Job Description Language Attributes Specification for the EGEE Middleware, document Identifier EGEE-JRA1-TEC-592336, Available online at https://edms.cern.ch/document/592336 (Aug. 2005).

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

[28] A. Anjomshoaa, F. Brisard, M. Drescher, D. Fellows, A. Ly, S. McGough, D. Pulsipher, A. Savva, Job Submission Description Language (JSDL) Specification, Version 1.0, OGF Specification GFD-R.056, http://www.gridforum.org/documents/GFD.56.pdf (Nov. 2005).

[29] S. Cantor, J. Kemp, R. Philpott, 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 (Mar. 15 2005).

[30] P. Andreetto, S. Andreozzi, A. Ghiselli, M. Marzolla, V. Venturi, L. Zangrando, Standards-Based Job Management in Grid Systems, Technical Note INFN/TC_08/6, Istituto Nazionale di Fisica Nucleare (INFN) (Oct. 9 2008).

[31] F. Brooks, Jr., No silver bullet–essence and accidents of software engineering, Computer 20 (4) (1987) 10–19. doi:http://doi.ieeecomputersociety.org/10.1109/MC.1987.1663532.

[32] G. Garzoglio, T. Levshina, P. Mhashilkar, S. Timm, ReSS: A Resource Selection Service for the Open Science Grid, in: S. C. Lin, E. Yen (Eds.), Grid Computing, International Symposium on Grid Computing (ISGC 2007), Springer, 2009, pp. 89–98.

[33] M.-E. Bégin, G. D.-A. Sancho, A. D. Meglio, E. Ferro, E. Ronchieri, M. Selmi, M. Zurek, Build, configuration, integration and testing tools for large software projects: Etics, in: N. Guelfi, D. Buchs (Eds.), RISE, Vol. 4401 of Lecture Notes in Computer Science, Springer, 2006, pp. 81–97. doi:http://dx.doi.org/10.1007/978-3-540-71876-5_6.

[34] E. Foster-Johnson, Red Hat RPM Guide, 1st Edition, Red Hat, 2003.

[35] YAIM Home Page, http://yaim.info/.

[36] R. Jain, The Art of Computer System Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling, Wiley, 1991.

[37] ALICE–A Large Ion Collider Experiment at CERN LHC, http://aliceinfo.cern.ch/.

[38] S. Bagnasco, L. Betev, P. Buncic, F. Carminati, C. Cirstoiu, C. Grigoras, A. Hayrapetyan, A. Harutyunyan, A. J. Peters, , P. Saiz, AliEn: ALICE environment on the GRID, Journal of Physics, Conference Series 129 (6). doi:http://dx.doi.org/10.1088/1742-6596/119/6/062012.