Job Management Enterprise Application

Thomas Soddemann

Rechenzentrum der MPG (RZG), Institut für Plasmaphysik, Boltzmann-Str. 2,
85748 Garching, Germany
soddemann@rzg.mpg.de

Abstract. This paper describes the development of a Job Management Enterprise Application (JMEA) which was developed by the DEISA material science and plasma physics joint research activities. It is capable of submitting jobs to a UNICORE server infrastructure and managing them. Since it is a Java EE application, it can be used by multiple users concurrently. Furthermore, it prefetches and caches request results in order to able of responding as quick as possible to client requests. In addition to normal user credentials it also supports the use of proxy credentials and explicit trust delegation.

1 Introduction

The Distributed European Infrastructure for Supercomputing Applications (DEISA) [1] is a consortium of leading national supercomputing centers that currently deploys and operates a persistent, production quality, distributed supercomputing environment with continental scope. The purpose of this FP6 funded research infrastructure is to enable scientific discovery across a broad spectrum of science and technology, by enhancing and reinforcing European capabilities in the area of high performance computing. This becomes possible through a deep integration of existing national high-end platforms, tightly coupled by a dedicated network and supported by innovative system and grid software.

The DEISA supercomputing grid is a European research infrastructure resulting from the integration of national High Performance Computing (HPC) infrastructures. This integration of national resources – using modern grid technologies such as UNICORE [6] – is expected to contribute to a significant enhancement of HPC capability and capacity in Europe.

DEISA is structured as a layer on top of the national supercomputing services, and coexists with them. This infrastructure addresses the computational challenges that require the coordinated action of the different national supercomputing environments and services for both efficiency and performance. DEISA provides scientific users with transparent access to a European pool of computing resources. The coordinated operation of this environment is tailored to enable new, ground breaking applications in computational sciences.

Eleven partners contribute currently to the DEISA infrastructure with their top level supercomputers: BSC, Spain; CINECA, Italy; CSC, Finland; ECMWF, UK; EPCC, UK; HLRS, Germany, IDRIS, France; FZJ, Germany, LRZ, Germany;

W. Lehner et al. (Eds.): Euro-Par 2006 Workshops, LNCS 4375, pp. 254–263, 2007.

RZG ,Germany; SARA, Netherlands This heterogeneous grid of super-computers includes of the most recent systems from leading vendors (IBM – PowerPC970, Power 4, 4+, 5, SGI – ALTIX, NEC – SX8).

Science Gateways, Portals and Web Service interfaces, are crucial for enhancing the user's adoption of sophisticated supercomputing infrastructures, by hiding from them the complexities of the computational environment. This extends up to the point that users make in their view direct use of an application. The choice of resource utilization is completely left to the infrastructure providing access to this application. So the portal solutions play the role of an application service provider (ASP).

The DEISA joint research activities in material sciences and plasma physics were faced with the development of comfortable means of access to the DEISA resources for standard applications in their fields like CPMD [2] and CP2K [3] for material sciences, and TORB [4] for plasma physics, a so-called science gateway.

Within DEISA several options for job submission across Cluster boundaries exist, e.g., the Multi Cluster Load Leveler (MC-LL) allows submitting jobs from the command line to any of the connected Load Leveler clusters (but naturally not to non MC-LL sites). The middle ware service activity within the DEISA project chose to employ the UNICORE suite as the default job submission interface to all heterogeneous compute resources within DEISA. Hence any DEISA job submission portal solution should be able to interface the UNICORE infrastructure deployed in DEISA in order to submit jobs in behalf of its users.

The UNICORE suite consists essentially of three components and one local batch scheduling system interface implementation establishing the connection to resource management systems like MC-LL and others. On the server side the central component is the Network Job Scheduler (NJS). It takes care of job submission, job management as well as file transfers and work flow execution. A gateway is the central entry point from the client perspective and several NJSs can be connected to it. A grid infrastructure relying on UNICORE can have more than one gateway e.g. for an increased fault tolerance. A user typically employs a rich client application (the UNICORE Client) to connect to several NJSs via a single gateway (see Fig. 1).

In the case of a portal application the job submission and management part needs to be implemented by an appropriate interface component. This components has to be able to carry out all tasks usually performed by the UNICORE client. Ideally, this interface is as general as possible in order to be able to deal with resource management systems besides UNICORE. The next section describes the requirements for such general job submission and management interface component.

The UNICORE suite offers a client library, the Arcon library, implementing an API which offers most of the desired functionality in dealing with the UNICORE server side. Unfortunately, this client library suffers from some minor deficiencies which mainly affect its use in a multi user multi threaded environment. This will be discussed in section 3.1.

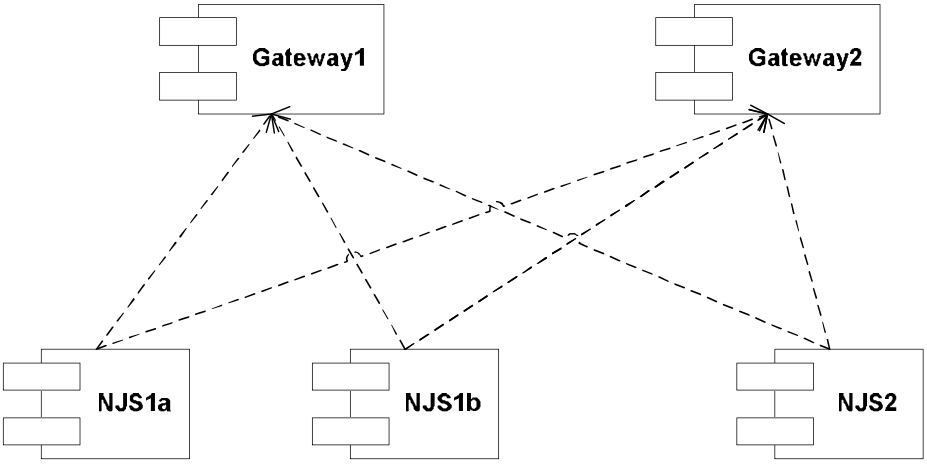

Fig. 1. Multiple gateways and NJSes deployed in a more sophisticated UNICORE infrastructure such as DEISA. Each NJS connects to all gateways. A user sees all NJSes regardless of the choice of the gateway.

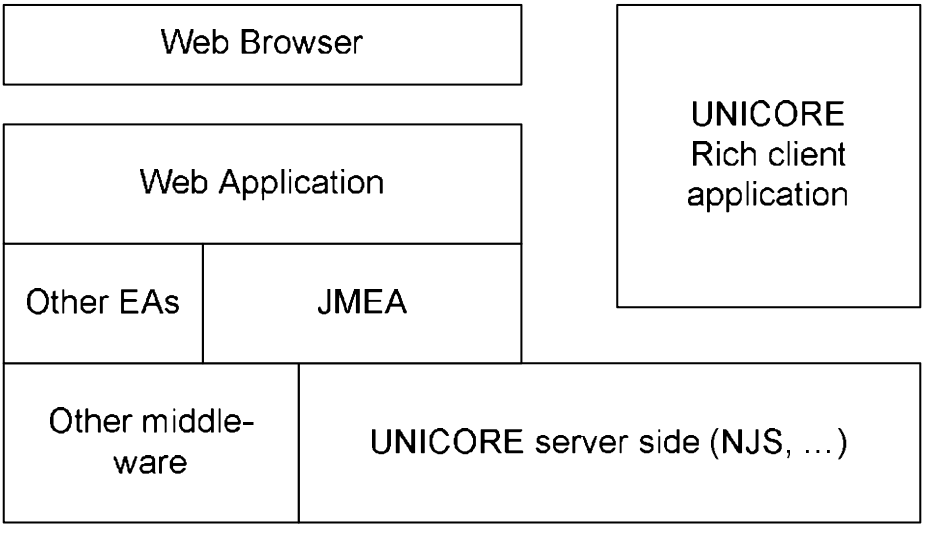

Fig. 2. Access to UNICORE within DEISA (without DESHL command line tooling). The Job Management Enterprise Application sits on top of the UNICORE server side infrastructure. A user accessing DEISA resources via the material science and plasma physics portal application makes direct use if JMEA for submission and handling. Furthermore, he is able to make use of other non-UNICORE related features like file system access. The UNICORE rich client application offers essentially the same functionality with respect to UNICORE. It speaks directly with the UNICORE server side. Pros and Cons of these approaches are discussed in detail in [5].

Section 3.2 describes a implementation of the Job Submission and Management interface described in section 2. It is implemented in form of a J2EE [10] version 1.4 compliant Job Management enterprise application developed within the DEISA project. It circumvents most of the problems described in section 3.1 and provides in addition functionality well beyond those of an ordinary client library. It is being used in the DEISA material science and plasma physics portal application.

The last sections provides conclusion and outlook of future developments.

2 Requirements for a Job Management Application

Ideally, a job management component for a portal application is able to connect to all grid job managers and local batch scheduling system. Adhering to this design principle, an API has to be used which is independent of any underlying job management component specifics. E.g. it should be able to work with a Globus [11] GRAM as well as with UNICORE NJS.

Hence, we identified a set of operation which we think are applicable to all kinds of resource management systems. This particular choice was motivated by the APIs of several different resource management systems [7,8] and the job management API of GGF's SAGA research group specified in their strawman API document [9]. It contains: submit (submitting the job request, cancel (canceling a job request which is not being executed), delete (delete a finished job request), kill (kill a running job request), halt (halt a job which is being executed), resume (resume a previously halted job). In addition it should be possible to retrieve information about the available resources, information about the status of jobs owned by a particular user, and results from a finished job such as console output. It is evident that the implementation of the job manager has to be able to deal with the identity management of the underlying resource management system. Fig. 3 shows the JobManager interface definition. All previously mentioned methods have been integrated into the definition. Additionally the client can select the gateway machine (if necessary as in the case of UNICORE) and retrieve a list of all known virtual site, which are part of the resource infrastructure.

Furthermore, the application should be able to cope with different instances of the same or different job management implementation at the same time, e.g. UNICORE and Globus. Hence, in addition to the JobManager Interface, a Factory for the JobManager is needed.

In the following we motivate two further requirements based on our experiences with UNICORE. But they should also hold for other Grid and cluster middle ware such as the Globus Tool Kit. The the requirements are derived from the design principle of giving best user satisfaction by having shortest request to response times and letting the user/client select the level of sophistication.

The UNICORE NJS requires a considerable amount of time to process many of its requests, e.g. request for a job status. Since it is undesirable to have users wait for anything, the application should answer right away. It should even answer if the targeted NJS should be unavailable for whatever reason.

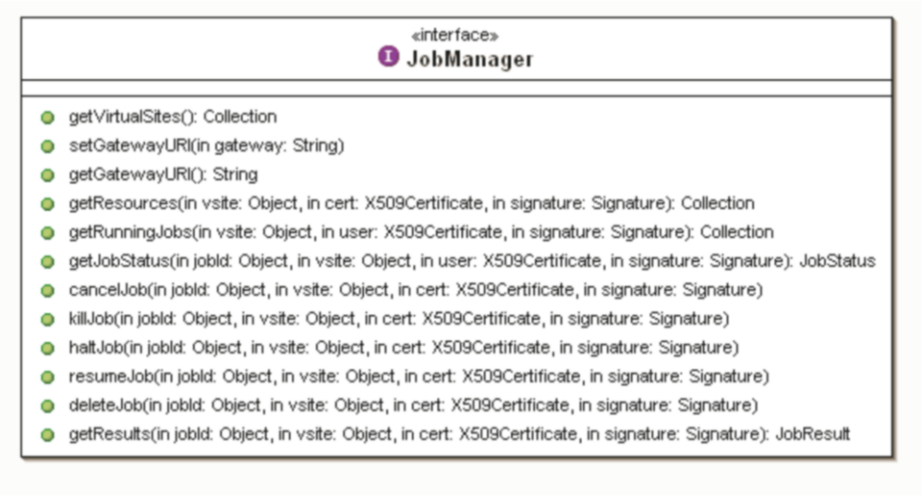

Fig. 3. JobManager Interface definition

Ideally, the application should find out about available resources and cache those information without user interaction.

3 Architecture and Implementation of the Job Management Enterprise Application (JMEA)

The Job Management Enterprise Application (JMEA) is as its designed as a multi-component application. It offers interface components such as the JobManager for synchronous interaction and a similar component for asynchronous interaction. Furthermore, persistent classes store jobs, job results, and job stati. Service components take care of the things like the periodic querying of job stati and automatic retrieval of results.

In the following we discuss why we choose to develop a new UNICORE client library as a replacement for the current Arcon release. The second subsection deals with the details of the implementation of JMEA.

3.1 The Arcon Client Library

In the case of UNICORE a client library exists which could be used to implement the UNICORE job manager: the Arcon library.

The Arcon library is client library which allows application to interact with the UNICORE server side (gateway and NJS). Although it works perfectly well for a single user application with a single thread interacting with the UNICORE server side, it has some limitations when employed in a multi user – multi thread environment.

In order to avoid race conditions in multi threaded applications, one should omit static variables, unless they are used for communication between the threads

and their access is synchronized. In the latter case, one has to make sure that synchronized access does not lead to a performance bottleneck.

In the case of the JobManager class of the Arcon library three static variables can be identified which can have an impact on the use in a multi user environment:

- `outcome_dir` which specifies the directory, streamed files will be stored
- `buffer_size` which reflects the buffer size for connections
- `always_poll` which tells if request are always asynchronous or not.

The static nature of those variables make it difficult to have per user settings which are desirable at least for the `outcome_dir`.

The abstract class *Connection* implements three static variables:

- `keep_open` which defines if the next connection is kept open after use.
- `compression` which tells if the transmission should be compressed or not.
- `encrypt` which defines whether the next retrieved connection should be encrypted communication or not.

While encryption may never been turned off, it is trivial to see that compression and keep open will definitely affect a multi user multi thread environment. Setting `keep_open` to true in one thread in order to retrieve a connection which stays open may result in an error, if that connection has been closed after first use, if a concurrent thread acquired a `keep_open=false` connection.

There are other classes which implement static variables such as *VsiteManager*. In our opinion, those do not limit the Arcon libraries use in a multi user environment.

The Arcon library implements its own proprietary logging mechanism which is not compatible with any of the existing logging mechanisms as log4j and the java.logging API. This makes it difficult to route logging messages to the applications or container logging files and impossible to influence the logging in the standard way.

Furthermore, in the implementation of some classes, e.g. in *Connection*, exceptions are used for flow control (bad style).

The main disadvantage at the time of the decision not to use the the original Arcon library was the missing support for Explicit Trust Delegation (ETD) [12] which allows a special user (agent) to formulate job request on behalf of a user by using his own set of credentials. In the mean time an EDT supporting Arcon library can be found at [13], which has not officially been released yet.

Furthermore, the Arcon library does not directly support Proxy Certificates which are favored in some environments over the use of ETD.

Based on these facts the decision was made to re-implement a client library, the new JobManager Library. Parts of the Arcon library's code base have been reused and those parts which were identified to be problematic have been replaced.

The new JobManager library now supports ETD, Proxy Certificates and is usable in a multi-user/multi threaded environment without limitations.

3.2 The Components

Fig. 4 sketches the main components of the job management enterprise application.

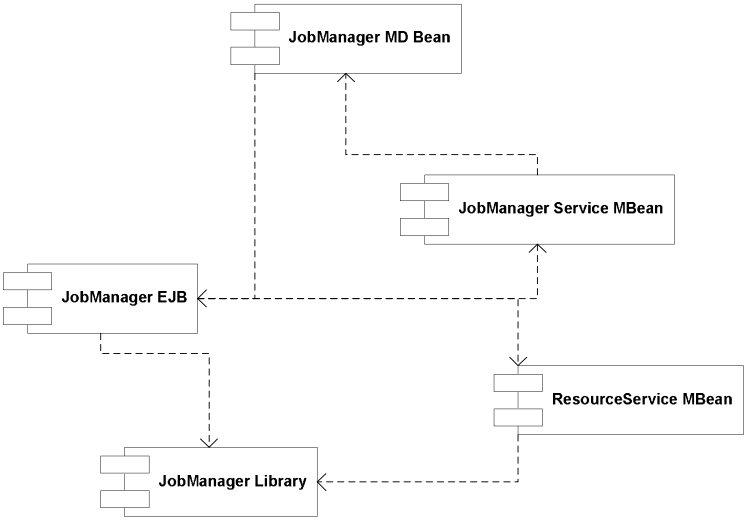

Fig. 4. Main components of the Job Management Enterprise Application

The JobManager Enterprise Java Bean (EJB) is the workhorse of the application. Its main purpose is to wrap calls to the JobManager library and handle persistent objects (see below). This EJB is implemented as a stateless EJB which allows the Application container to pool instances. The size of the pool can adjusted to optimize the balance the number of concurrent requests and responsiveness of the application.

The JobManager Message Driven Bean wraps the job update methods JobManager EJB and allows each of them to be executed asynchronously. These beans receive messages containing the AJO id of the job whose status is to be updated or performs a bulk update on all jobs for a particular user.

The JobManager Service Management Bean performs service tasks such as the periodic initiation of status update requests for running jobs. It employs the JobManager Message driven beans and the RunningJobs persistent object in order to achieve its goal.

Resources such as available software etc. are published in UNICORE on a per site basis. Hence the enterprise application queries the resources in configurable intervals and makes the result available to all of its users. The JobManager Resource Service Management Bean is responsible for querying the various NJS server instances and caching the query result. Furthermore, it provides information about the status of an NJS (available/not available). Since it has cached information, a temporarly unavailability does not mean that a client does not get information from it.

Several persistent classes have been used in order to store job requests, job stati, job results, and running jobs. Hibernate [14] is used for object relational (O/R) mapping purposes. Among the persistent classes are the JobStatus class, JobResult class and the RunningJobs class. JobStatus contains all status information available from UNICORE. In a similar way the results of finished jobs including stdout and stderr are stored in objects of the JobResult class. Objects of the RunningJobs class contains information about all running jobs for a particular user who is known to the enterprise application.

As mentioned in sec. 2, especially the UNICORE NJS requires some time to process certain requests. To circumvent this behavior and give an answer to the client side as quick as possible, information such as the job status, which obviously can change (e.g. from queued to running to finished), is polled periodically and cached by the application. The JobManagerService MBean operates a timer service which periodically initiates the polling by sending messages to the JobManager Message Driven Beans. These utilize instances of the JobManager EJB in order to retrieve and cache the required information from the NJS. When a client asks the JobManager EJB about such information it answers by querying the cache. A similar conecept is applied for the caching of a site's resource information.

Generally, requests should be grouped, if possible. Refreshes should be performed at suitable intervals. This shortens the time to a response on client requests for information using the application. Only information which need a special user authorization have to be retrieved separately, e.g. asynchronously after the user/client has authenticated himself to the application and provided the necessary credentials so that the application is able to act on the user's behalf. If the concept of explicit trust delegation (ETD) is enabled in the targeted UNICORE deployment, this task is trivial, since the NJS checks the authorization.

3.3 Security

In the current version of the enterprise application different security models are in place. The enterprise application itself can be protected by employing the container's security mechanisms. Once authenticated and authorized, a client can access all methods of all EJBs deployed in this application. There are currently no priviledged methods which need role based access restrictions. Access to the NJS via the enterprise application can be achieved in three ways of authentication by either employing

- the concept of Explicit Trust Delegation [12],
- Proxy credentials [15], or
- using the user credentials,

UNICORE is able to deal with one level of proxy certificates if the NJS `check_signers` property has been switched on. In that case the client needs to provide a *Signature* object for most of the operations which result in an interaction with a NJS. The JobManager library act in this case as a mediating NJS.

In the case of Explicit Trust delegation, it is not necessary for the client to provide a *Signature* object. The JobManger library here is used in the Agent mode, and signs all request with as a agent while setting the user attribute fields in the AJOs accordingly. The only thing needed in both cases is the X509 certificate.

In the case of ETD it should in principle suffice to provide a X500 DN of the user. This would simplify things, if the certificate is not available to the JobManager application. In discussions with the NJS authors it became clear that the certificate is used to identify users in cases where different users have the same DN. Nevertheless, in our opinion, only certificate authorities should be used which do not reuse their DNs or have overlapping name spaces. E.g. CAs, which are members of the EUGridPMA, qualify for that.

4 Conclusion and Outlook

The Job Management is currently in production use for the materials science and plasma physics portal of the according DEISA activities. In the mean time extension plans towards an integration of WS-GRAM are made. Furthermore, JMEA will be used in the implementation of the UNICORE connector for GridSAM [16]. This will allow users of the OMII [16] and hence users of the AHE [17] to submit jobs to the DEISA infrastructure. The reader might have missed a treatment of file transfer. Currently, file transfers are not implemented in JMEA itself (apart from the fact, that files can be embedded in job objects). An additional file management application is used to transfer file with appropriate mechanisms. Since JMEA will primarily be used in connection with web applications and web services, transfers of large volume data using HTTP/HTTPS as the transport layer in form of a multi-part HTTP request or embedded in an XML document is for performance reasons not advisable.

The Arcon library could easily be extended to integrate seamlessly into a multi threaded – multi user environment and a part of this work has already been incorporated in the unreleased version [13]. This would certainly help to build third party applications on top of UNICORE which do not rely on the UNICORE rich client.

A few ideas came into the authors' minds when working on JMEA. These could also enhance the integrability of UNICORE into third party applications. In some applications connectors to UNICORE have easy access to the *Principal* class of the user, but often accessing the user/client certificate or even the whole certificate chain is everything else than trivial. Hence, *X500Principal* object should be used rather than the whole certificate in the class *UserAttributes*.

Furthermore, there are requests which do not necessarily need to be performed by a user. E.g. requests for resource information could be performed by an ETD agent as well. Result could then be handled by the agent itself and e.g. delivered to all users. This would reduce the number of request to make to an NJS. But currently an agent is not allowed to make such a request. This should be changed in our opinion.

References

1. DEISA, Distributed European Infrastructure for Supercomputing Applications, http://www.deisa.eu
2. CPMD, http://www.cpmd.org
3. CP2K, http://cp2k.berlios.de
4. Hatzky R, Tran TM, Knies A, Kleiber R, Allfrey SJ. Phys.Plasmas 2002; 9: 898.
5. Thomas Soddemann, Concurrency Computat.: Pract. Exper., DOI 10.1002/cpe
6. UNICORE, http://unicore.sf.net/
7. http://www-03.ibm.com/servers/eserver/clusters/software/loadleveler.html
8. http://gridengine.sunsource.net
9. A. Merzky, et al., https://forge.gridforum.org/sf/docman/do/downloadDocument/projects.saga-rg/docman.root/doc12183
10. Java 2 Enterprise Edition
11. The Globus Toolkit, http://www.globus.org/
12. D. Snelling, et al., http://www.fujitsu.com/downloads/MAG/vol40-2/paper12.pdf
13. Unreleased EDT version of the arcon library, http://fisheye1.cenqua.com/browse/unicore/unicore/optional/arconclient
14. Hibernate http://www.hibernate.org/
15. Proxy Certificate RCF3820
16. Open Middleware Infrastructure Institute, http://www.omii.ac.uk/
17. Application Hosting Environment, http://www.realitygrid.org/AHE/