A Standards Based Approach to Job Submission Through Web Services

William Lee, Stephen McGough, Steven Newhouse, John Darlington

London e-Science Centre, 180 Queens Gate, Department of Computing,
Imperial College London, London, SW7 2BZ, UK
`{wwhl,asm100,sjn5,jd}@doc.ic.ac.uk`

Abstract. The ability to submit jobs to foreign computational resources is a core requirement in Grid Computing. There are many Distributed Resources Managers (DRM) in existence for deploying jobs to Grid Resources. However, they require jobs to be described in propreitary language. The submission interfaces are often restricted solely for human interaction or consumable only through language-specific APIs. In this paper, we demonstrate the use of the widely accepted and standardised Web Service specifications and related technologies to build a Job Submission Web Service (WS-JDML). Leveraging the Job Description Markup Language (JDML), it provides a DRM-neutral input to the submission service, where transformation can take place in order to hide the underlying DRMs. The input and output sandbox abstraction in JDML also allows data transfer mechanism to be described in a generic manner.

1 Introduction

A Service-oriented Grid Architecture promotes composition of generic services as a way to realise higher-level task. Job submission service is one of the most important building block to allow computational task to be deployed on foreign Grid resource transparently. The service should also encapsulate state monitoring and output transfer.

There are many Distributed Resource Management (DRM) systems available supporting some of these requirements, namely Condor[9], Sun Grid Engine[1] and LSF[8]. However, there are no standardised interface for these heterogeneous systems. They differ not only on their functionalities, but most of which have their own job descriptions. Access to these systems can be characterised by the description language and the interaction interface. Most systems provide a combination of command-line tools, socket-based interface, API with different language-bindings (e.g. DRMAA[5], Java CogKit[13]), and recently HTTPS web access and the latest adoption of Web Service. The heterogeneity hinders the development of interoperable Grid-wide job submission and induces lock-in to a specific vendor.

The growing set of maturing Web Service specifications have encouraged the Grid community to consider open standards as a substrate in building service-oriented architecture. Efforts in reviewing the usability of Open Grid Services Infrastructure[3], and the latest development in WS-Resource Framework[4] and the Grid Application Framework (WS-GAF) have demonstrated the need to closely adhere to already established standards and implementations in order to promote interoperability as well as achieving the network effects. The adoption of any of these frameworks are largely orthogonal to job submission. However, the need to build on the foundation of Web Service is apparent from the commonalities. Related to our work, recent development in the Globus GRAM service[12] tackles the interoperability problem by adopting the Open Grid Services Infrastructure as the interface layer and the XML schematised RSL language as the job submission language.

Our proposal seeks to provide an alternative that employs Web Service standards that are supported by a wide variety of vendor toolings. It demonstrates the design of the Job Submission Web Service interface that utilises a generic language as job description. We show how this interface can be implemented using the J2EE enterprise platform for the added benefits of scalability and fault-tolerance. Through the use of pluggable transformation and connectors, the system allows jobs to be submitted to any resource made available on the grid irrespective of the underlying architecture or DRM systems.

2 WS-JDML Web Service Interfaces

The WS-JDML web service comprises of two related port types decoupling the submission and monitoring capabilities. The semantics of the service centered around the Job Description Markup Language (JDML) that describes job with terms, such as the executable, arguments, input, output and an extensible set of terms describing operational requirements of the job. This section will first discuss

JDML as a stand-alone language to describe aspects of a job, then inspect the web services port types that acts on JDML document.

2.1 Job Description Markup Language

JDML is a common job description language originally developed for use within the European Union Datagrid[11] based around Condor (ClassAds)[10], which was used in the project. JDML has subsequently been adopted as the main deployment language used within the Imperial College eScience Networked Infrastructure (ICENI)[7] for deploying work onto a wide range of architectures. JDML is now being used as one of the Job Description Languages feeding into the Global Grid Forum (GGF) Job Submission Description Language Working Group (JSDL-WG)[6]. The role of the JSDL-WG is to define a common Job submission language that can be used to describe jobs that can be launched. The work that comes out of the JSDL-WG will feed back into JDML.

JDML documents are structured into sections. The primary sections are those describing the job to execute, the environment to use, how to obtain the files required for the job and where to send the result files. The document may be extended with additional sections that contain information that is only relevent for particular DRMs. To allow jobs to be executed anywhere this extra DRM information can be safely ignored and the job will still run successfully.

The JDML document consists of strongly typed named atrribute value pairs. This allows tighter control on JDML documents and more inline checking of the contents, but also enables the value of the attibutes to be constrained (through inequalities and logical conditions) and represented within the document as an XML fragment. The language also supports a number of functions for such things as determining the current date and time, string manipulation and general comparison operations.

The files section of a JDML document describes how to obtain those files required for the executon of the job and what to do with the generated files. Each file is defined along with one or more file transfer descriptions. These define how to transfer the file by such techniques as Grid FTP, HTTP transfer or transfer over a shared file space. If more than one transfer description exists the DRM can choose which one to use. The input and output sandbox defines a virual file systems that exists during the lifetime of the job irrespective of the underlying architecture and absolute file locations.

2.2 Job Submission Port Type

The role of the submission port type is to abstract the submission action using a JDML document as the input. The JDML document details the parameters and operational requirements of a job. The port type responds to a submission request in several ways.

Unrecognised Job Term - Since the JDML document allows extensible set of terms to be attributed for future expandability and DRM-specific actions, the submission port raises a fault if some terms are not recognised as a defensive interpretation.

Invalid Job Term - A fault is raised if the input JDML contains term value that is typed incorrectly, or the value is unrecognised. This caters for some job terms, such as file sandbox containing unsupported transfer mechanisms.

Successful Submission - Upon submission of a JDML that is attainable, the job is scheduled or run asynchrously to the response. The submission port responds with a URI that uniquely identifies the job that can be passed to the the monitoring port to retrieve job status and output.

The use of an URI as a reference to the job instance allows the submission and monitoring ports to be decoupled functionally and physically. The monitoring service can reside on distributed resources or be indirectly addressed through intermediate gateways, firewalls, etc.. in a transport-neutral manner. By standardising on the URI scheme in use, client can be adapted to resolve the physical destination of the monitoring port. Although this metadata is opaque to the client (e.g. mapping of unique ID to DRM specific ID), it can be used by multiple co-operative monitoring services to achieve load-balancing and fault-tolerance.

2.3 Job Monitoring Port Type

The monitoring port type abstracts the process of retrieving job status and managing output transfer. Client interacts with the job monitoring port with status request message augmented with job identifier URI.

The current design mandates a set of well-known Uniform Resource Identifiers (URI) to define a common set of job states, namely *pending*, *scheduled*, *running*, *suspended*, *done* and *exit*. The set of states might not be supported by all job submission service, for example, DRM that doesn't support checkpointing might not provide the *suspended* state. Moreover, ways of advertising the

supported state transition is not defined in the current design. The meaning of the job state supported by different job submission service is essential to enable autonomous monitoring in the future.

Apart from inspecting job status, clients can use the monitoring port to initiate output transfer from the output sandbox. Client can request portion of the files to be transfered using the mechanism already defined in the JDML document, or be overridden with alternative method and address. This allows the output to be underspecified at submission time and modified at runtime. Apart from the transfer mechanism defined in JDML, the monitoring port type allows data transfer using SOAP attachment.

3   A Generic WS-JDML Web Service

The WS-JDML web service port types have been prototyped as part of the ICENI project to demonstrate the functionalities on top of existing DRMs. The implementation is built on the Java J2EE 1.4 platform showcasing many capabilities. By architecting on the J2EE standard APIs and deployment model, the system is widely deployable on J2EE platforms provided by different vendors allowing deployers to have a wider choice on the performance guarantee and the cost. The current prototype implements the WS-JDML web service interface and a generic backend layer that can communicate with a variety of DRM systems, namely secured shell execution and condor at the time of writing. Figure 1 depicts the current architecture.

In the web service layer, the JAX-RPC compliant Java implementation of the submission and monitoring port types are responsible to verify the JDML input against the schemas and supported set of job terms. They are also responsible to authorise requests based on the WS-Security[2] artifacts carried in the SOAP headers. When a new job arrives, the job request is persisted to acknowledge receipt and pass on to the next stage in the pipeline.

At the core of the implementation, the Java Message Queue (JMS) is responsible for persisting the input jobs in a FIFO fashion. The queue is transactionally managed by the container, therefore subsequent failure in the pipeline would ensure changes in the job state be rollbacked and replayed at the next opportunity. The JobBean serves as the long-term representation of the job state and other information conveyed back to the client during the lifetime of the job.

The JobConsumerBean removes a job request from the queue and initiates the job submission process. The current implementation relies on the abstraction of a *JDMLTransformer* and *DRMConnector* pair for each underlying DRM. The JDMLTrans-

former implementation is responsible for transforming a JDML document into the native format of the DRM and a set of *Action*s to be performed by the DRMConnector. The set of actions include file transfer and initialisation of the environments. The *DRMConnector* obtains the actions and the native script from the transformation pipeline, and connects to the DRM system using platform-specific means, such as secured shell, Java API, etc.. Depending on the monitoring facilities provided by the underlying DRM systems, job state update is currently obtained by pulling the DRM using a *TimerBean* periodically.

In order to handle the input and output sandboxes, the current implementation allows file to be transmitted using SOAP attachments, and other traditional means of file transfers, such as grid-ftp, sftp, etc.. depending on the set of configured file transfer components available as extensible *Action*s.

4   Discussion

The WS-JDML port types defined have several open areas to be examined;

– *Job State Transition* - The ability to describe the job state transition supported by a job submission service is essential for autonomous monitoring agents to act on the reported state of the job.
– *Notification* - The current interface only supports a pull-approach in retrieving job status. A more scalable approach would be to notify a nominated sink service to receive state change event. Upcoming Web Service specifications, such as WS-Notification might be able to address this in a standardised manner.
– *Job Term Semantics* - Definition of job term is documented using natural language specification. The lack of formal model to describe job term semantic makes transformation error-prone and open to ambiguity. By annotating job term definition with ontological metadata in different job description language, one can confidently reason and transform between job description.

The implementation is an on-going project seeking to provide more comprehensive support to different DRMs and file transfer protocols. The scalability of the system can be characterised by the J2EE containers in use. Distributed Java Message Queue implementation is common-place, therefore the job consumer layer can be vertically distributed and horizontally scaled. Fault-tolerance is ensured by the transactional capability of the container and
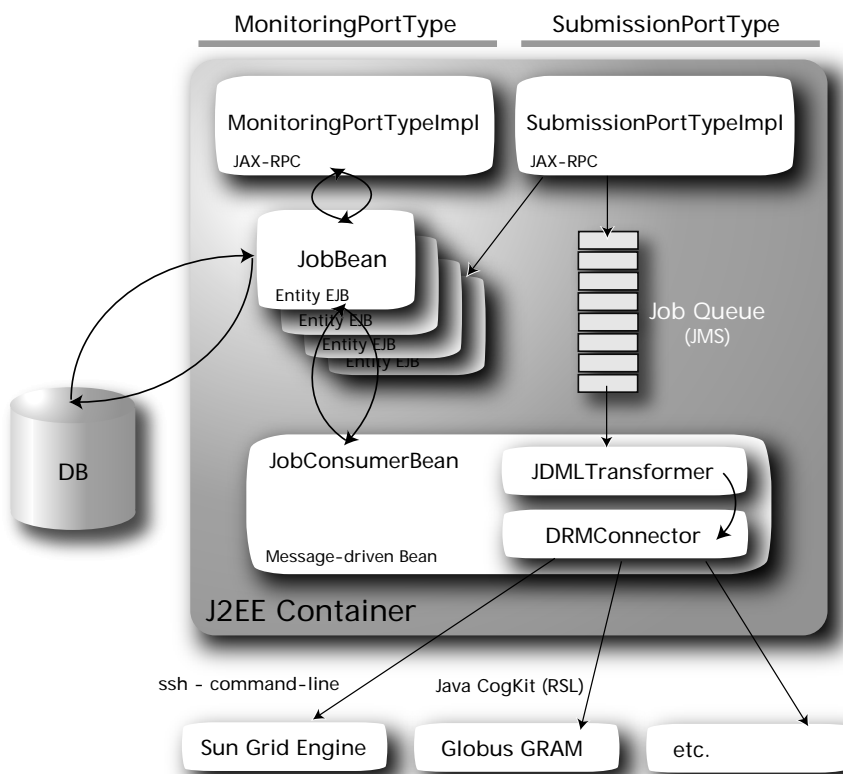
Fig. 1: A Generic WS-JDML Web Service in J2EE

the long-term persistence of states in database. Although once the job entered into the DRM layer, the treatment of job failure depends on the recovery facility provided by underlying DRM.

The implementation has employed WS-Security to secure message exchange between the client and the service. However, the translation of X509 credential carried in the WS-Security layer to the DRM-specific credential is a subject for future work. A possible direction would be to adopt a credential delegation scheme where the WS-JDML service acts as a gateway to submit job to backend DRM on behalf of the client. However at the time of writing there is no existing standard in the context of WS-Security to enable secured delegation of identity.

5   Conclusion

This paper has discussed an experimental specification of a Job Submission and Monitoring Web Service. The port type definition utilises exist-

ing Web Service standards to achieve transport-neutral addressing and message-level security. The experimental definition is accompanied by an exploratory implementation building on industry-supported Web Service tools and platforms. The generic implementation can be extended to support a rich set of DRMs and data transfer mechanisms.

References

1. Project SGE. `http://wwws.sun.com/software/gridware/`.
2. Web Services Security: SOAP Message Security v1.0, OASIS Standard. `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-se%curity-1.0.pdf`.
3. I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Open Grid Service Infrastructure WG, Global Grid Forum, June 2002.
4. Web Services Resource Framework. `http://www.globus.org/wsrf/`.

5. Distributed Resource Management Application API Working Group. `http://www.drmaa.org/`.

6. Job Submission Description Language Working Group. `https://forge.gridforum.org/projects/jsdl-wg`.

7. Imperial College e-Science Network Infrastructure Project. `http://www.lesc.ic.ac.uk/iceni/`.

8. Platform LSF. `http://www.platform.com/products/LSF`.

9. Condor Team. Condor Project Homepage. `http://www.cs.wisc.edu/condor`.

10. The ClassAd Language Reference Manual Version 2.1. `http://www.cs.wisc.edu/condor/classad/refman//`.

11. The DataGrid Project. `http://www.eu-datagrid.org/`.

12. The Globus Project. `http://www.globus.org/`.

13. Gregor von Laszewski, Ian Foster, Jarek Gawor, Warren Smith, and Steve Tuecke. CoG Kits: A Bridge between Commodity Distributed Computing and High-Performance Grids. In *ACM Java Grande 2000 Conference*, pages 97–106, San Francisco, CA, 3-5 June 2000.