A JSDL Application Repository and Artefact Sharing Portal for Heterogeneous Grids and the NGS

David Meredith, d.j.Meredith@dl.ac.uk
(Science and Technology Research Council, Daresbury Laboratory, Warrington, WA4 4AD)
Mina Maniopoulou, A.Maniopoulou@rl.ac.uk
Andrew Richards, a.j.Richards@rl.ac.uk
(Science and Technology Research Council, Rutherford Appleton Laboratory, Didcot, OX11 0QX)
Mike Mineter, mjm@nesc.ac.uk
(National e-Science Centre, Edinburgh EH8 9AA)

Abstract

The portal application discussed in this paper (https://portal.ngs.ac.uk) facilitates sharing, storing and graphical authoring of middleware agnostic JSDL documents, and subsequent staging and job submission. Architecturally, the portal is not tied to a particular middleware or set of staging protocols, which supports extension and customisation for use in heterogeneous Grids. Users benefit by sharing application descriptions, and from the expertise and artefacts captured in JSDL (pre-published by domain-experts and NGS). We describe the portal from a software standards perspective, and provide user, resource administrator and Grid training critiques.

1.0 Introduction

The current heterogeneity of Grid middleware (e.g. Globus, WSRF, gLite, Condor) and data staging protocols on which Grids are built pose a great challenge for realising a truly pervasive and transparent Grid. Importantly, different middleware implementations adopt different schemes for the description of applications and their associated resources, and for their subsequent execution. Examples of different middleware specific application description schemes used within the Grid space include Globus Resource Specification Language for Globus 2.x Grids [1] (RSL), Job Description Documents for Globus GT4 WS-GRAM Grids [2] (JDD), Job Description Language for EGEE gLite Grids [3] (JDL), Open Grid Forum Job Submission Description Language [4] (OGF JSDL), and Condor Class-Ads [5]. Some different schemes available in the Grid space for the description of a single application and its resources are given in the Appendix (RSL, JDL, JSDL). A more detailed discussion regarding the different available schemes is given later. Often the heterogeneity of Grid middleware is apparent even within a single Grid organisation (this may not necessarily be disadvantageous to the user). An example includes the UK National Grid Service [6] (NGS), which currently supports Globus and applications described in RSL, and a modified version of the gLite middleware for the execution of applications described with JDL.

The provision of different application description formats and middleware does offer choice, but also adds complexity and hinders interoperability.

In addition to the different schemes used for the description and submission of applications, a number of different protocols are also relevant for the management and transfer of data (i.e. ‘cross protocol file staging’). Within the Grid space, this includes GsiFTP [7], SRB [8] (Storage Resource Broker), WebDav and (S)FTP for example. This ‘cross protocol’ file staging is especially relevant when considering heterogeneous Grids. In this scenario, either the middleware must support mediated data transfers between required data stores (i.e. ‘third party file transfers’), or a dedicated staging service must be devised to perform the cross protocol file staging (e.g. staging between an SRB vault and a GsiFTP server). These data staging requirements are particularly important when considering Service Orientated Architectures (SOA) style Grids. This is because SOA style services often implement temporary job execution environments or sandboxes for the purpose of file staging and job execution, rather than relying on the provision of permanent user accounts on compute resources to which data is stored, rather than temporarily staged.

The portal application described herein has been engineered with these concerns in mind (https://portal.ngs.ac.uk). The ongoing aim is

to provide an extendible and generic portal application for heterogeneous Grids, which describes applications and their resources using middleware agnostic JSDL (Job Submission Description Language). The portal application is based strongly on standards, and is not tied to a particular middleware or set of data transfer protocols. Consequently, the portal can be extended and customised to accommodate new middleware, and to interact with different data storage resources as required. For the portal, this has largely been facilitated with use of the OGF JSDL v1 specification, released under OGF Recommendation GFD.56 [9]. The JSDL specification is an XML Schema language used to describe the requirements of computational jobs for submission to resources in Grid environments. Importantly, JSDL does not mandate the use of particular staging protocols or middleware. In addition, a selection of important Java software engineering standards will be described.

2.0 Core Functionalities

2.1 Application Description Repository

The NGS Applications repository is an open access portal used to describe and list applications and their associated artefacts. JSDL documents can be searched for under particular categories of interest as shown in Figure 1 (e.g. Tutorials / Examples, Chemistry). All necessary information required to execute a hosted application is captured in full by its associated JSDL. Users may freely browse the publicly available application descriptions without prior authentication, which only becomes necessary when attempting to interact with Grid resources (e.g. file browsing, job submission). The open access style of the portal was purposefully designed to help encourage Grid adoption, as user communities may immediately interact with the graphical user interface (GUI), browse applications, and provide feedback without prior certification (which is often perceived a 'barrier' to entry in Grid computing). Importantly, the open access functionality of the portal also allows users to consume the portal as a free-to-use JSDL GUI editor, even if applications are not actually executed with the portal.

2.2 GUI and JSDL Authoring

The portal GUI facilitates viewing, authoring and validating JSDL documents, including JSDL-POSIX extensions (Portable Operating System Interface) such as resource requirements, file systems, environment variables, candidate host lists, arguments and data staging elements. Typically, this information is already pre-configured for 'out of the box' application descriptions, and users need only tweak application descriptions as required (e.g. modify arguments). The JSDL XML is created by interaction with the portal GUI and can be directly accessed and modified as required. The portal validates the generated XML for correctness against the JSDL and JSDL POSIX extension schemas, and lists validation errors. The portal can also be used to provide additional (logical) validation constraints upon element values. This is because JSDL is loosely typed and does not impose strong validation constraints on the values of elements through advanced XSD restriction techniques such as xsd:patterns.

2.3 A JSDL and Artefact Sharing Framework

The portal supports the upload and subsequent validation and storing of JSDL documents. This allows user communities to distribute and share their application descriptions, related artefacts and associated expertise captured in pre-configured JSDL.

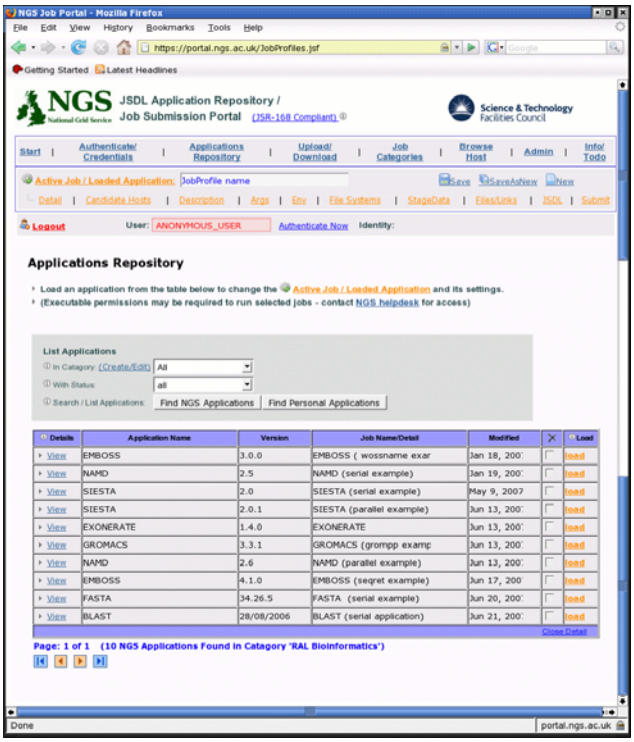

Figure 1. The applications repository page lists NGS and personal applications under configurable job categories (e.g. Bioinformatics, Tutorials/Examples, Chemistry)

2.4 Data Staging and Staging URIs

The portal can be used to record the URIs of required data by interacting with data storage resources (e.g. listing/browsing resources) and creating JSDL data staging elements (Figure 2). Responsibility for performing the staging is delegated to the underlying middleware or to a dedicated staging service. Architecturally, this is good design because the portal should not be responsible for undertaking core middleware functionality, and secondly because staging large amounts of data via the portal server would be problematic, especially in terms of concurrency. Of course, if the chosen middleware does not support staging between the required storage resources, this can be a limitation. Application artefacts can also be linked with the application description, including example standard input and output files, source code, and links to external Web pages. The key concept, which should be emphasised, is that the portal can be extended to support interaction with different data stores and middleware without having to change the underlying architecture or data model.

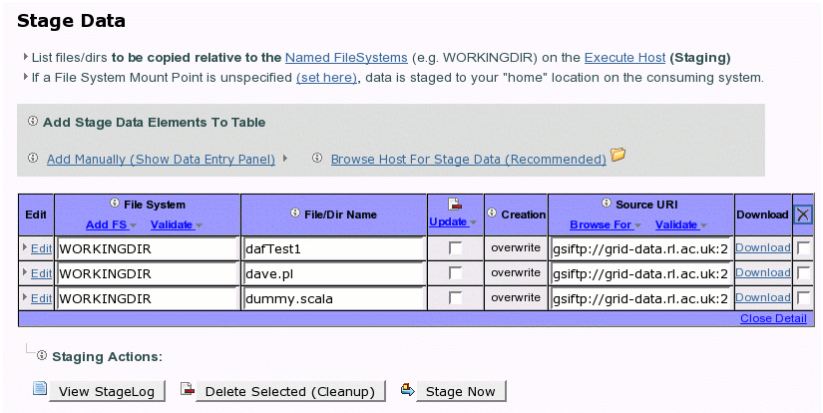

Figure 2. The data staging page can be used to configure JSDL data staging elements.

3.0 Job Submission and Monitoring (Translating JSDL to Middleware Specific Application Descriptions)

Job submission involves translating the agnostic JSDL into the middleware specific application description scheme for the chosen middleware. For example, given the JSDL application description shown in Appendix C, the RSL shown in Appendix A would be required for submission to Globus 2.x Grids, whilst the JDL shown in Appendix B would be required for submission to EGEE gLite Grids. To date, Globus 2.x is catered for in the portal. Work is also currently underway to fully support the creation of JDL scripts, and to support the OMII JSDL (Open Middleware Infrastructure Institute [10]) extensions for submission to GridSAM services [11].

Converting middleware agnostic JSDL application descriptions into middleware specific schemes can be problematic and is discussed here. This is because certain key attributes or elements may not be common or semantically equivalent across the different application descriptions. Complexity is also added when considering different middleware versioning (e.g. later versions of Globus support more RSL parameters). As an example, no equivalent JSDL elements exist for the following gLite JDL attributes, 'RetryCount,' 'MyProxyServer,' 'Parameter,' 'ParameterStart' and 'ParameterStep.' Most notably, JSDL does not yet support parametric style jobs with equivalent element support for the JDL 'Parameter…' attributes (it must be noted however, that early draft JSDL extensions for parametric style jobs are currently under development). The 'Parameter' attribute is used to specify the set of values that parametric attributes should assume, whilst the 'ParameterStart' attribute specifies the starting value for the variation, and the 'ParameterStep' defines the increment step. In addition, not all the attributes defined in the EGEE GLUE schema [12] for defining resources are represented by JSDL Resource and JSDL POSIX extensions. An example includes the 'other.GlueCEInfoLRMSType' for defining the batch queue system. Similarly, no JSDL elements capture the RSL 'jobType', 'gramMyJob', 'dryRun', 'two_phase' and 'restart' parameters (not all are given here). In these scenarios, the attributes/data that do not have semantic JSDL equivalents have to be accommodated within the portal (in both interface and database) and also in JSDL. JSDL can be augmented/extended with the use of different extension schemas that are designed to describe additional data requirements. Consequently, elements from the extension schema(s), which have different XML namespace definitions, can be added to the JSDL and validated as required. An example of this is illustrated in Appendix C, where the JSDL has been extended with elements from a gLite JDL extension Schema. This is required in order to accommodate JDL specific GLUE schema attributes, an example of which is shown in Appendix B (i.e. other.GlueCEInfoLRMSType == "PBS"). In order to accommodate the required GLUE attribute, the additional XML namespace definition is added to the root <JobDefinition> element (i.e. "xmlns:jdl=http://www.glite.org/jdl"). The JDL element can then be embedded into the JSDL in place of <xsd:any/> 'placeholder' elements, which are defined widely within the JSDL schema for the purpose of extension. This is shown in Appendix C with the addition of the following element: "<jdl:LRMSType>PBS</jdl:LRMSType>" (note the different "jdl" namespace prefix). JSDL also defines generic <jsdl:JobAnnotation> elements that can be used to capture additional information. This is also shown in Appendix C, where a JobAnnotation is used to capture a portal specific data extension, namely the job 'Category' with: "<jsdl:JobAnnotation>catagory:Tutorials / Examples</jsdl:JobAnnotation>."

In practice, different methods can be used to do the conversion between the different application descriptions. Since JSDL is standard XML Schema, it can be readily compiled using Schema binding and validation tools such as JAX-B [13] or XMLBeans [14] to create a utility API for working with associated XML instance documents. In doing this, it becomes fairly easy to manually convert between these descriptions using the generated API. However, this assumes the developer has a thorough knowledge of the different schemes and of the correct conversion semantics. In practice, it should be more simplistic to rely on tools that have been designed specifically for these conversions, such as dedicated XSLT transforms. However, in practice extreme care should be taken, as this can also be problematic. For example, an XSLT transform was used to convert between the JSDL shown in Appendix C and the JDL shown in Appendix B. However, the transform failed to interpret the JSDL '<FileSystem/>' elements and their nested '<MountPoint/>' path sub-

elements. These elements are used to reference path information in JSDL wherever 'filesystemName' attribute are used from within other elements (e.g. within '<DataStaging>', '<Input>', '<Output>', '<Error>' elements). Consequently, because the XSLT failed to interpret these elements and their references correctly, the mount point path information is dropped in the resulting JDL attributes. This is shown in Appendix B where the JDL does not interpret the full path information of certain files (e.g. the 'cpi.in,' 'cpi.out,' 'cpi.err' attributes). From experience in developing Grid applications, (hidden) errors such as these can be very problematic.

4.0 Viewpoints / Critiques

In the following sections, we present different viewpoints from parties who have been involved with development, testing and use of the portal (software engineer, resource administrator, Grid-trainer, user).

4.1 Software Engineering Viewpoint (Architecture and Standards)

A number of software engineering standards and design patterns for n-tier server application development have been observed and will be described/evaluated here. The application architecture includes a stateless transactional service facade, a DAO layer and a Web layer with concurrency controller. In addition, a set of standalone business logic objects have been designed and are managed using dependency injection / inversion of control (i.e. DI / IOC). Each layer depends only on the layer immediately below (e.g. service layer objects never invoke the web layer while DAO objects never invoke the service layer). An important decision was to support portal deployment to lightweight Java Web containers (i.e. Servlet containers including Tomcat [15] and Jetty [16]), and not limit deployment to fully featured (heavyweight) J2EE application servers (e.g. JBoss [17] and Glassfish [18]). This was decided in order to support the distribution of the portal to the largest possible community. Java/JEE functionality (e.g. declarative transaction demarcation, DI/IOC) was implemented using the Spring J2EE application framework [19].

4.1.1 Web and Controller Layer

The Web and model-view controller layer (MVC) had been implemented using Java Server Faces (JSF) technology [20], the Java standard for building server application interfaces. Importantly, vanilla JSF allows applications to be hosted as plain Web applications, or as portlet within JSR-168 [21] portal containers (e.g. Gridsphere [22], uPortal [23]). In doing this, the portal can be readily integrated into institutional portals. A limitation that was encountered during development, is that JSF and JSP page development are designer un-friendly and can be tedious. The API does not integrate well with visual Web interface design tools for the design of page layouts (e.g. what you see is what you get Web page authoring). In summary, JSF is regarded by the authors as a powerful API, but can be complex.

4.1.2 Persistence, DAO and Service Layers

The persistence layer has been implemented using the EJB 3.0 Java Persistence API (JPA – JSR-220) [24] and Hibernate 3.2 [25] as the chosen object-to-relational mapping (ORM) provider. These APIs allow a relational domain model to be captured as a set or reusable object entities. The entities can be mapped to most relational database management servers (RDBMS) available today. Persistent objects are retrieved using the standard EJB-QL (Enterprise Java Bean Query Language), which translates the database agnostic query into the underlying RDBMS SQL variant. An extremely useful functionality that was found during development is capability to detach and serialise persistent objects, allowing them to be passed up to different application tiers (e.g. the Web tier for modification and subsequent re-attachment for persisting to the database). This deprecates the previous (and common) transfer object anti-pattern [26]. The transactional service layer was implemented using the Spring IOC container. Spring facilitates the injection and propagation of the JPA persistence context for long running transactions. In addition, Spring allows transactions to be demarked declaratively rather than programmatically using aspects (i.e. the new paradigm of aspect orientated programming – AOP). In doing this, if a DAO operation fails, the SQLException exception is translated, captured and handled by the aspect to ensure that DAO interfaces do not provide a leaky abstraction. We found this extremely useful, especially for alleviating the development of tedious and error prone database exception handling code (i.e. repeated try / catch / finally boilerplate statements). The authors regard this functionality as extremely flexible and powerful. In summary, a

combination of EJB 3.0 JPA and Spring is highly recommended by the authors when developing applications for Java standard edition and light weight containers (Tomcat, Jetty).

4.2 Resource Administrator Viewpoints (Contributed by NGS Resource Administrators)

From a resource administrator's viewpoint, the portal is "quite straightforward to use," provided he/she knows where an application is installed including its dependencies. Creating job-submission templates via the portal then becomes routine. The architecture of the portal is an important feature as it allows for semi-transparent changes in grid infrastructure. For example, within the NGS, permanent user accounts are currently provided, but the present status is being revised, with the consideration of providing temporary staging accounts. Since the portal facilitates file staging, this allows such changes to take place without affecting the portal. Importantly, this would minimise the effect on "our" end-user, and also allows resource administrators to make (semi) transparent infrastructure changes. As a future enhancement-suggestion, we suggest improvements to the JSDL 'publishing' model, giving the administrator (or even user) finer grained controls on who can access and view application templates. Currently, this can only be achieved by e-mailing and then uploading JSDL documents between relevant parties, which is not the most functional solution for resource administration. In addition, more relevant documentation could save an administrator time and effort. The Application Developer Training Course held by the UK National e-Science Centre is a very good opportunity for system administrators to get first hand experience on the portal and to make suggestions.

4.3 Portal and Training Viewpoints (Contributed by TOE-Nesc Training Team)

At the time of writing (early April 2007) the portal is now becoming a routine part of NGS training. Commonly given courses are categorized as 'Induction' or 'Application Developer.' The former introduces concepts of the NGS and provides experience of using the core software stack. The application developer courses are suited to those seeking to develop and deploy new applications, especially as VOMS is becoming established and an increasing focus centres on ways to share an application across a community. The NGS Portal fits this profile of training as follows:

4.3.1 Induction Courses: These courses now include use of the portal and introduce JSDL concepts. In induction courses, the current routine mode of interacting with the NGS is presented: GSISSH onto an access machine; Globus commands to submit, monitor and manage jobs. The portal is useful as an exemplar of a) how more user-friendly higher level tools can be built upon basic services (basic services that for the less computer-oriented can be daunting), and b) how standards are facilitating development of these higher level tools. However the portal is not primarily seen as a tool only for new users of the NGS, and it is not seen as a component that is only useful for training. The portal is being presented as being useful in its own right, as a service that will for many be a routine part of a researcher's way to benefit from the NGS. This is especially the relevant for those users who wish to be able to submit and manage jobs from browsers, without having to invoke GSISSH.

4.3.2 Application Developer Courses: These courses include how application developers can deploy their application via the NGS portal. Future modules are likely to include the re-use of NGS JSR-168 portlets in a particular VO's portal.

4.4. User Viewpoints (Collated from NGS User Feedback)

For the most part, users are only concerned with knowing how to run their applications on a Grid just as they would on their own workstations. It was regarded as unreasonable for all users to be expected to learn middleware specific commands (or even become Linux/Unix users in some cases). Such processes are both time and energy consuming, and it was felt that this does not allow users to concentrate solely on their research. Feedback suggests the portal helps solve this problem by abstracting the user from the complexity of learning middleware and Linux/Unix commands. The alternative method for job-submission on the NGS, i.e. the GSISSH terminal, which requires users to log in and submit jobs using Globus Toolkit commands, was regarded as certainly necessary for 'power users', i.e. those users who require shell interactivity (e.g. for compiling applications and for advanced requirements). However, it was felt that this is not suitable for those who are not as comfortable with Linux/Unix and Grid/Globus commands. It was commented upon that there is not much additional overhead in using the portal compared to running applications on desktop computers. In

addition, it was appreciated that the portal saves jobs, which automatically provides a history for monitoring and/or archiving purposes. In addition, prospective users can access and navigate through the portal, investigate its functionality and browse through the applications installed without having to authenticate in advance. Users consider this a particularly nice feature.

A number of improvements have also been suggested. Firstly it was felt that the technical 'jargon' used within the portal (and indeed within the Grid space as a whole) confuses users and the portal page titles should be more descriptive. Another feature that that should be addressed is support for parametric style jobs. At present, the portal only allows single job submissions. Consecutive (i.e. parametric) and/or interrelated jobs can only be submitted if an application is wrapped within in a submission script. Users have requested this functionality be added to the portal. In addition, it was felt that the portal GUI could reflect a particular application more closely. For example, if the name of an input file is specified as a command line argument, there should be no need for the standard input field to be shown in the portal GUI. Users also requested functionality to collate information on Grid resources, including installed libraries and modules. To sum up, it was generally considered that the portal helps integrate the users to the grid, and that it provides extra functionality without excessive overhead. Its user-friendliness can and should be improved further and there is space for enhancing its functionality, but it has been described as an invaluable tool for grid users within the NGS.

5.0 Conclusion and Future Developments

The NGS applications repository portal is aimed to encourage a wider Grid audience by providing a high level of abstraction. This is achieved with the use of middleware agnostic JSDL application descriptions for job description and execution. The portal has been identified for open source support by OMII under a free BSD licence. Future developments include support for new middleware providers, especially for OMII GridSAM instances and EGEE gLite, and for file staging from different data storage resources. Of particular importance, is the support for parametric style jobs, which could be facilitated in the portal with the release of the JSDL parametric schema extensions. The applications repository can be used by users (and administrators) to publish and share pre-configured application descriptions and to provide access to associated resources for 'out of the box' execution. In addition, the portal can be used by resource-administrators to provide 'recipes' or 'templates' for describing the setup of particular resources (e.g. application examples with pre-configured environment setup, compilers, listed modules etc). The portal has become an integral component in courses led by the Nesc Training Outreach Education (TOE) team for Grid training throughout the UK. Importantly, the portal provides a framework for demonstrating key Grid concepts with the use of specially designed, portal hosted teaching applications and examples. New users begin submitting Grid jobs with minimal effort.

6.0 Acknowledgements
Computing resources provided by STFC's e-Science facility.
UK National Grid Service.
National e-Science Centre Training and Outreach Education team.

7.0 References

[1]GT 2.4: The Globus Resource Specification Language RSL v1.0.
http://www.globus.org/toolkit/docs/2.4/gram/rsl_spec1.html
[2]Usage Scenarios for Jobs Specified in JDD
http://www.globus.org/toolkit/docs/development/4.1.1/execution/gram4/user/index.html#gram4-user-usagescenarios-jdd
[3] gLite – Lightweight Middleware for Grid computing. http://glite.web.cern.ch/glite/
[4] Job Submission Description Language3. https://forge.gridforum.org/projects/jsdl-wg/
[5] Condor, High Throughput Computing. http://www.cs.wisc.edu/condor/
[6] NGS, UK National Grid Service. http://www.grid-support.ac.uk/
[7] Grid FTP. http://www.globus.org/grid_software/data/gridftp.php
[8] Storage Resrouce Broker. http://www.sdsc.edu/srb/index.php/Main_Page
[9] OGF GFD 56. http://www.gridforum.org/documents/GFD.56.pdf
[10] Open Middleware Infrastructure Institute. http://www.omii.ac.uk/

[11] GridSAM – Grid Job Submission and Monitoring Service. http://gridsam.sourceforge.net/2.0.1/index.html
[12] EGEE GLUE Schema. http://glueschema.forge.cnaf.infn.it/
[13] JAXB – Java API for XML Binding and Validation. https://jaxb.dev.java.net/
[14] Apache XMLBeans. http://xmlbeans.apache.org/
[15] Apache Tomcat. http://tomcat.apache.org/
[16] Jetty Web Server. http://www.mortbay.org/
[17] JBoss Application Server. http://labs.jboss.com/portal/
[18] Glassfish Application Server. https://glassfish.dev.java.net/
[19] Spring Application Framework. http://www.springframework.org/
[20] Java Server Faces Technology. http://java.sun.com/javaee/javaserverfaces/
[21] JSR 168 Portlet Specification. http://jcp.org/aboutJava/communityprocess/final/jsr168/
[22] Gridsphere Portal Framework. http://www.gridsphere.org
[23] uPoral. http://www.uportal.org/
[24] EJB3 Java Persitence API. http://java.sun.com/products/ejb/
[25] Hibernate. Relational Persistence for Java and .NET. http://www.hibernate.org/
[26] The Gang of Four Group (GOF). http://www.industriallogic.com/patterns/ili_nyc_gof.html

8.0 Appendix

Some different schemes available in the Grid space for the description of applications and their resources (each example describes the same application).

a) RSL, used to describe applications in Globus 2 Grids. b) JDL, used to describe applications in gLite Grids. c) JSDL, which is not tied to a particular middleware. In all examples, the executable ('cpi') is staged from to the consuming system. The JDL 'InputSandBox' and 'OutputSandBox' parameters describe files to be staged between a local workstation and the consuming system while JSDL defines <jsdl:DataStaging> elements with either </jsdl:Source> or </jsdl:Target> sub elements. The RSL does not capture staging data, which (usually) has to be staged separately using Globus gsiftp commands. Refer to the text for discussion.

A) Globus RSL (Resource Specification Language)

```
&(executable=$(GLOBUSRUN_GASS_URL)/home/ngs0153/cpi) (arguments= 30 fileA) (jobType=mpi)
(environment = (NGSMODULES mpich-gm/1.2.5..10-intel8.1:intel/fce/9.1.032) (TMP /tmp)) (count = 4)
(hostCount = 8) (minMemory = 512) (maxWallTime = 3)  (directory=/home/ngs0153)
(stdin=/home/ngs0153/cpi.in) (stdout=/home/ngs0153/cpi.out) (stderr=/home/ngs0153/cpi.err)
```

B) gLite JDL (Job Description Language)

```
Type = "Job";
JobType = "Normal";
RetryCount = 3;
Executable ="/home/ngs0153/cpi";
Arguments = "30 fileA";
VirtualOrganisation = "myGridVOproject";
StdInput = "cpi.in";
StdOutput = "cpi.out";
StdError = "cpi.err";
InputSandbox = { "gsiftp://grid-data.rl.ac.uk:2811/home/ngs0153/cpi", "gsiftp://grid-
data2.dl.ac.uk:2811/myhome/fileA" };
InputSandboxDestFileName = { "cpi", "fileA" };
OutputSandbox = { "cpi.out" };
OutputSandboxDestURI = { "gsiftp://mygridhome.dl.ac.uk:2811/myhome" };
DeleteOnTermination = { "fileA" };
Environment = { "NGSMODULES=mpich-gm/1.2.5..10-intel8.1:intel/fce/9.1.032", "TMP=/tmp" };
Requirements = ( other.GlueCEInfoLRMSType == "PBS" ) && ( member( GlueCEInfoHostName, {"grid-
data.rl.ac.uk:2119" , "mygrid-resource.dl.ac.uk:2119" } ) ) && ( GlueHostProcessorModel == "Intel" );
Rank = -other.GlueCEStateEstimatedResponseTime;
```

C) OGF JSDL (Job Submission Description Language)

```
<jsdl:JobDefinition xmlns:jsdl=http://schemas.ggf.org/jsdl/2005/11/jsdl
                xmlns:jsdl1=http://schemas.ggf.org/jsdl/2005/11/jsdl-posix xmlns:jdl=http://www.glite.org/jdl >
 <jsdl:JobDescription>
  <jsdl:JobIdentification>
   <jsdl:JobName>MyC MPI C CODE</jsdl:JobName>
```

```
    <jsdl:JobProject>myGridVOproject</jsdl:JobProject>
    <jsdl:Description>The code calculates Pi</jsdl:Description>
    <jsdl:JobAnnotation>catagory:Tutorials / Examples</jsdl:JobAnnotation>
   </jsdl:JobIdentification>
   <jsdl:Application>
    <jsdl:ApplicationName>CPI</jsdl:ApplicationName>
    <jsdl:ApplicationVersion>1.0</jsdl:ApplicationVersion>
    <jsdl1:POSIXApplication>
     <jsdl1:Executable>/home/ngs0153/cpi</jsdl1:Executable>
     <jsdl1:Argument>30</jsdl1:Argument>
     <jsdl1:Argument>fileA</jsdl1:Argument>
     <jsdl1:Input filesystemName="WORKINGDIR">cpi.in</jsdl1:Input>
     <jsdl1:Output filesystemName="WORKINGDIR">cpi.out</jsdl1:Output>
     <jsdl1:Error filesystemName="WORKINGDIR">cpi.err</jsdl1:Error>
     <jsdl1:WorkingDirectory>/home/ngs0153</jsdl1:WorkingDirectory>
     <jsdl1:Environment name="NGSMODULES">
          mpich-gm/1.2.5..10-intel8.1:intel/fce/9.1.032</jsdl1:Environment>
     <jsdl1:Environment name="TMP">/tmp</jsdl1:Environment>
     <jsdl1:WallTimeLimit>180</jsdl1:WallTimeLimit>
     <jsdl1:ProcessCountLimit>4</jsdl1:ProcessCountLimit>
    </jsdl1:POSIXApplication>
   </jsdl:Application>
   <jsdl:Resources>
    <jdl:LRMSType>PBS</jdl:LRMSType>
    <jsdl:CandidateHosts>
     <jsdl:HostName>grid-data.rl.ac.uk:2119</jsdl:HostName>
     <jsdl:HostName>mygrid-resource.dl.ac.uk:2119</jsdl:HostName>
    </jsdl:CandidateHosts>
    <jsdl:FileSystem name="WORKINGDIR">
     <jsdl:FileSystemType>normal</jsdl:FileSystemType>
     <jsdl:Description>The working job directory</jsdl:Description>
     <jsdl:MountPoint>/home/ngs0153</jsdl:MountPoint>
    </jsdl:FileSystem>
    <jsdl:TotalPhysicalMemory>
     <jsdl:LowerBoundedRange>5.36870912E8</jsdl:LowerBoundedRange>
    </jsdl:TotalPhysicalMemory>
    <jsdl:TotalResourceCount>
     <jsdl:Exact>8</jsdl:Exact>
    </jsdl:TotalResourceCount>
     <jsdl:CPUArchitecture><jsdl:CPUArchitectureName>Intel</jsdl:CPUArchitectureName>
    </jsdl:CPUArchitecture>
   </jsdl:Resources>
   <jsdl:DataStaging>
    <jsdl:FileName>cpi</jsdl:FileName>
    <jsdl:FilesystemName>WORKINGDIR</jsdl:FilesystemName>
    <jsdl:CreationFlag>overwrite</jsdl:CreationFlag>
    <jsdl:DeleteOnTermination>false</jsdl:DeleteOnTermination>
    <jsdl:Source><jsdl:URI>gsiftp://grid-data.rl.ac.uk:2811/home/ngs0153/cpi</jsdl:URI></jsdl:Source>
   </jsdl:DataStaging>
   <jsdl:DataStaging>
    <jsdl:FileName>fileA</jsdl:FileName>
    <jsdl:FilesystemName>WORKINGDIR</jsdl:FilesystemName>
    <jsdl:CreationFlag>overwrite</jsdl:CreationFlag>
    <jsdl:DeleteOnTermination>true</jsdl:DeleteOnTermination>
    <jsdl:Source><jsdl:URI>gsiftp://grid-data2.dl.ac.uk:2811/myhome/fileA</jsdl:URI> </jsdl:Source>
   </jsdl:DataStaging>
   <jsdl:DataStaging>
    <jsdl:FileName>cpi.out</jsdl:FileName>
    <jsdl:FilesystemName>WORKINGDIR</jsdl:FilesystemName>
    <jsdl:CreationFlag>overwrite</jsdl:CreationFlag>
     <jsdl:Target><jsdl:URI>gsiftp://mygridhome.dl.ac.uk:2811/myhome</jsdl:URI></jsdl:Target>
   </jsdl:DataStaging>
  </jsdl:JobDescription>
 </jsdl:JobDefinition>
```