Project no: **RI031844-OMII-Europe**

Project acronym: **OMII-Europe**

Project title: **Open Middleware Infrastructure Institute for Europe**

Instrument: **Integrated Infrastructure Initiative**

Thematic Priority: **Communication network development**

M:JRA1.7 Definition of JSDL Extensions

Due date of deliverable: Oct 2006
Actual submission date: ???, 2006

Start date of project: **1 May 2006** Duration: **2 years**

Organisation name of lead contractor for this deliverable: INFN

Revision [0.4]

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)		
Dissemination Level		
PU	Public	**Public**
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Document Change History

Version	Date	Comment	Author/Partner
0.1	2006-09-22	First draft	Moreno Marzolla/INFN
0.2	2006-10-10	Second draft	Moreno Marzolla/INFN
0.3	2006-10-12	Third draft	Moreno Marzolla/INFN, with contribution from Vivian Li/FLE and Morris Riedel/FZJ
0.4	2006-10-13	Fourth Draft	Moreno Marzolla/INFN

Document Control Sheet

Document	**Title**: Definition of JSDL extensions	
	ID: M:JRA1 1.7	
	Version: 0.4	**Status**: Draft
	Available at: http://grid.pd.infn.it/omii/milestones:jra17	
	Software Tool: OpenOffice.org	
	File(s): OMII-Europe_JRA1_Job_Submission_Milestone_Oct_2006.doc	
Authorship	**Written by:**	Vivian Li Moreno Marzolla (Editor) Morris Riedel
	Contributors:	JRA1 Team
	Reviewed by:	TBD
	Approved by:	TBD

Document Status Sheet

Version	Date	Status	Comments
0.1	22 September 2006	Draft	Initial gLite part
0.2	10 October 2006	Draft	Added part on UNICORE (contributed by Morris Riedel/FZJ) and Globus
0.3	12 October 2006	Draft	Modified UNICORE part according to contribution from Vivian Li
0.4	13 October 2006	Draft	Minor cleanups

Table of Contents

1 Executive Summary

This document identifies the requirements and extensions for JSDL (the Job Submission Description Language) which are required by the gLite, UNICORE and Globus systems. JSDL currently provides basic capabilities for describing a generic "computational job", including its data and resource requirements. However, actual grid systems often provide more expressive job description notations which allow a larger number of job information characteristics to be described. The aim of this document is to describe which extensions to JSDL are needed by gLite, UNICORE and Globus in order to allow users to take advantages of the features of the individual systems.

After a brief introduction to the core JSDL concepts, we give a high-level description of the abovementioned grid systems, with particular emphasis on their current job description notation. Then, for each one we describe how JSDL maps to that job description notation, and identify which extensions to JSDL are necessary to the different systems. Finally, we provide XML Schemas for the identified extensions.

2 Introduction to JSDL

One of the most important functionalities offered by any Grid system of middleware is the possibility of submitting jobs, which will then be executed on suitable computational resources. While the exact notion of "job" usually varies from Grid to Grid, there are many common features which can be isolated. For example, a "job" usually consists of executing some executable program on a given processor; the program may operate on one or more input data files, and produce one or more output data files. Moreover, job requirements (minimum available memory, disk space, CPU speed) may be part of the job specification.

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

The Job Submission Description Language (JSDL) is an XML-basede notation for describing the requirements of computational jobs for submission to Grid environments. The JSDL notation is defined by means of a normative XML Schema that facilitate the expression of those requirements as a set of XML elements .

The JSDL specification is motivated by the need to achieve interoperability by different Grid job management systems; in fact, it is not uncommon that the same user community uses different Grid systems at the same time, each with its own notation for describing jobs. In this scenario, a common, standardized notation such as JSDL is clearly desirable.

The aim of JSDL is to provide a notation for describing the structure and requirements of individual jobs. Other, equally important, aspects of job submission and management are outside the scope of JSDL. For example, many Grid systems provide the notion of "structured job collections"; as an example, in the gLite framework, a DAG (Directed Acyclic Graph) can be used to represent workflows where multiple, independent jobs can be scheduled according to a set of user-defined inter-job dependencies. Most Grid systems have similar features; however, these are outside the scope of the JSDL specification.

While the JSDL specification is general enough to encompass the basic features of most Grids, there are many other specific features which are not present. The JSDL specification has an extensino mechanism by means it is possible to define extensions to JSDL, and add specific additional information: in fact, the JSDL specification allows arbitrary XML elements to be added in specific position, provided that the new XML elements have a different namespace than the JSDL one.

The JSDL specification defines the following element sets:

- *Job Structure Elements*, which act as high-level containers of job description and requirements;
- *Job Identity Elements*, which contains all elements that identify the job;
- *Application Elements*, which describe among other things the name and version of the application to be executed;
- *Resource Elements*, which describe the resources required by the job;

- *Data Staging Elements*, which describe the files which should be moved to or from the execution host, respectively before job start, or after the job completed.

These elements are organized according to the following XML pseudo schema

```
<JobDefinition>
 <JobDescription>
  <JobIdentification ... />?
  <Application ... />?
  <Resources ... />?
  <DataStaging ... />*
 </JobDescription>
 <xsd:any##other/>*
</JobDefinition>
```

As you can see, the <xsd:any##other/> element is used to denote extension points, that is, places in the XML schema where user-defined tags can be inserted. There are many of such extension points placed over the XML Schema for JSDL.

The JSDL specification also includes one mandatory extension, namely the POSIXApplicatino extension, which defines a schema describing an application executed on a POSIX compliant system.

3 Job Description in Grid Systems

In this section we analyze the current job description notation used in three major Grid systems and middlewares, namely gLite, Globus and UNICORE.

3.1 gLite

3.1.1 General Overview

In this section we describe the Job Description Language (JDL) used to describe jobs within the gLite infrastructure, and how the gLite JDL compares to the JSDL specification.

We start by illustrating the path usually traversed by a job in the gLite infrastructure. Figure 1 depicts the typical job submission scenario and the relevant gLite components involved.

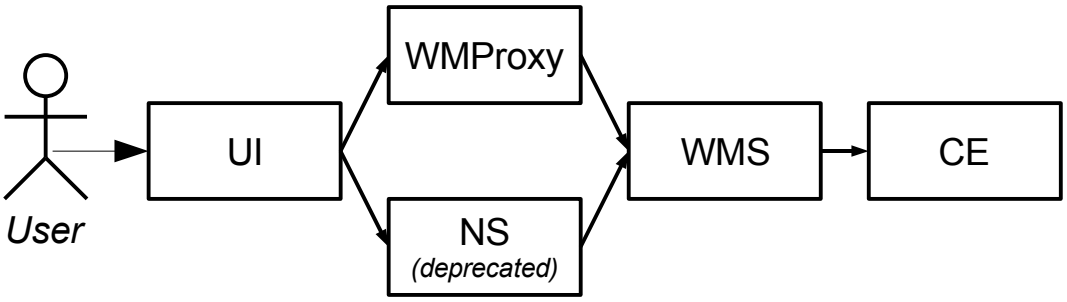


Illustration 1: Job submission path in gLite

Here, the user first prepares a Job, using a textual notation described in the next section. The job is then submitted using the User Interface (UI) component. The job is then transferred to the Network Server (NS) or the WMProxy; the first component was the traditional entry point for jobs; it is now considered deprecated, and the new entry point is the WMProxy, which exposes a Web Service-based interface. The WMProxy in turn transfers the job to the Workload Management System (WMS), which, among other things, performs the *matchmaking* phase. The WMS checks the requirements of the job, and selects a set of candidate Computing Elements (CEs) where the job could be run. The job is finally sent to one of these CEs and executed.

Each one of these components may alter the job description it receives before passing it to the next component. Thus, the job description received by the last component in the chain (CE) is not necessarily identical to the one submitted by the user to the UI. Additional, component-specific information are automatically added during the preocessing of the request at each step.

3.1.2 JDL specification

The current job specification for gLite is based on the Job Description Language (JDL) . JDL is a textual description based on the Classad notation; in its simplest form, it contains the following information:

- The name of the executable program;
- The program argument(s);
- The input files needed by the program;
- The output files produced by the program.

The gLite JDL supports the following kind of jobs:

- **Job** (a simple job);
- **DAG** (a Directed Acyclic Graph of dependent jobs);
- **Collection** (a set of independent jobs).

Here is an example of a simple job:

```
[
  Type="Job";
  JobType="Normal";
  VirtualOrganisation="EGEE";
  executable="/bin/echo";
  arguments="Hello World!";
]
```

This JDL represents a job of type "Normal" (more on job types later); the job executes the command `/bin/echo` with arguments `Hello World!`. This specific job does not require any input file, nor produces any output file (apart from the standard output). The `VirtualOrganisation` tag is used to define the Virtual Organisation (VO) associated with the job.

A more complex example of job follows (this example is taken from):

```
[
  JobType = "normal";
  Executable = "jobExecutor";
  Arguments = "1 100";
  Environment = {"PATH=$PATH:/usr/local/bin",
                 "LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/lib"};
  InputSandbox = {"/home/glite/scripts/jobExecutor ",
                  "/home/glite/scripts/envcheck.sh"};
  StdOutput = "je.out";
  StdError = "je.err";
  OutputSandbox = {"je.out","je.err","je-1-100.out"};
  OutputSandboxDestURI = {"je.out",
                          "je.err",
                          "https://trinity.datamat.it:7443/run/je-1-100.out"};
  Rank = -other.GlueCEStateEstimatedResponseTime;
```

```
   Requirements = RegExp(".*lal\.in2p3\.fr.*", other.GlueCEUniqueID) &&
                  (other.GlueCEPolicyMaxCPUTime >= 100);
   // If no resource is available retry matchmaking until 2006-05-26 18:30:00
   ExpiryTime = 1148729400;
   Prologue = "envcheck.sh";
   RetryCount = 2;
   ShallowRetryCount = 4;
   MyProxyServer = "kuiken.nikhef.nl";
   VirtualOrganisation = "infngrid";
   PerusalFileEnable = true;
]
```

Here we see a number of attributes which can be used to specify requirements for the job, or which can be used to tune the behavior of the gLite infrastructure. For example, the `InputSandbox` and `OutputSandbox` attributes specify the set of input and output files used by the job. In particular, the `InputSandbox` attribute represents a list of files which are needed by the job for running. The `OutputSandbox` attribute is used to specify the set of files produced by the job, after it has completed. The `RetryCount` attribute is used to define the maximum number of job resubmissions which should be attempted after the job failed for some reason not related to the job itself (that is, the number of resubmissions cuased by failure of gLite components).

One interesting thing to note is that the gLite JDL attributes can be roughly partitioned in two classes:

1. Attributes used to describe the job itself (e.g., executable name, input and output sandbox);
2. Attributes used to describe how the job is to be processed by the gLite infrastructure (e.g., `RetryCount`, `ShallowRetryCount`).

JDL attributes falling in class 1 above should be considered as candidates for a "JSDL gLite extension", as they are related to the job, rather than to the way job are processed by the gLite infrastructure.

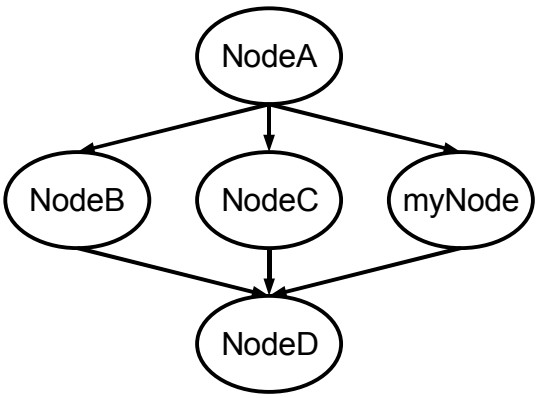
gLite currently supports the following types of jobs (these only apply to "Normal" jobs, and not to "DAG" nor "collections"):

- *Normal*: a simple batch job
- *Interactive*: a job whose standard streams are forwarded to the submitting client
- *MPICH*: a parallel application using MPICH-P4 implementation of MPI
- *Partitionable*: a job that can be thought as composed by a set of independent steps/iterations, i.e. a set of independent sub-jobs, each one taking care of a step or of a sub-set of steps, and which can be executed in parallel
- *Checkpointable*: a job able to save its state, so that the job execution can be suspended, and resumed later, starting from the same point where it was first stopped
- *Parametric*: a job whose JDL contains parametric attributes (e.g. Arguments, StdInput etc.) whose values can be made vary in order to obtain submission of several instances of similar jobs only differing for the value of the parameterized attributes

Finally, we give an example of DAG:

```
[
  Type = "dag";
  VirtualOrganisation = "EGEE";
  MyProxyServer = "skurut.cesnet.cz";
  HLRLocation = "eth.to.infn.it:5562:/O=CESNET/O=INFN To/CN=Andrea Guarise";
  InputSandbox = { "/tmp/foo/*.exe",
                   "/home/gliteuser/bar",
                   "gsiftp://neo.datamat.it:5678/tmp/cms_sim.exe ",
                   "file:///tmp/myconf" };
  InputSandboxBaseURI = "gsiftp://matrix.datamat.it:5432/tmp";
  Rank = - other.GlueHostEstimatedTraversalTime;
  Requirements = other.GlueCEStateStatus == "Production";
  max_nodes_running = 5;
  nodes = [
    nodeA = [ description = [ JobType = "Normal"; Executable = "a.exe"; InputSandbox =
{ "/home/data/myfile.txt", root.InputSandbox }; ]; ];
    mynode = [ description = [ JobType = "Normal"; Executable = "b.exe"; Arguments = "1 2
3"; RetryCount = 3; Requirements = other.GlueCEInfoTotalCPUs > 2; Rank =
other.GlueCEStateFreeCPUs; OutputSandbox = {"myoutput.txt", "myerror.txt" };
OutputSandboxDestURI = "gsiftp://neo.datamat.it:5432/tmp"; ]; ];
    nodeD = [ description = [ JobType = "Checkpointable"; Executable = "b.exe"; Arguments =
"1 2 3"; RetryCount = 3; InputSandbox = { "file:///home/pippo",
root.nodes.mynode.description.OutputSandbox[0] }; ]; ];
    nodeC = [ file = "/home/test/c.jdl"; ];
    nodeB = [ file = "foo.jdl"; node_retry_count = 2; ];
  ];
  dependencies = { { nodeA, nodeB },
                   { nodeA, nodeC },
                   {nodeA, mynode },
                   { { nodeB, nodeC, mynode }, nodeD }
  };
];
```

The above DAG represents a set of five jobs (named `nodeA`, `mynode`, `nodeB`, `nodeC`, `nodeD`), with the dependencies shown in the following graph:



3.2 Unicore

3.2.1 General Overview

UNICORE stands for Uniform Interface to Computing Resources, is a vertically integrated Grid Computing Environment, that provides a seamless, secure, and intuitive access to distributed resources, regardless the differences in "hardware architectures, vendor specific operating systems, incompatible batch systems, different application environments, historically grown computer center practices, naming convections, file system structures, and security policies". The current release of UNICORE 6 consists of 5 core components: UnicoreGS(the Web services implementation), SOAP enabled Gateway, NJS, TSI, and UUDB.

As shown in Figure 2, the UnicoreGS provides five services that are conceptually named as the UniGrids Atomic Services (UAS). In particular, the UAS define the mandatory functionality for system, file, and job management within Grids and potentially in other infrastructures such as pervasive computing environments. Furthermore, the UAS were used to contribute to the Execution Services Interface (ESI) document created by Dave Snelling (UNICORE) and Ian Foster (Globus) and thus contributed to the standardization process of the OGSA - Basic Execution Services (OGSA-BES) specification of OGF.

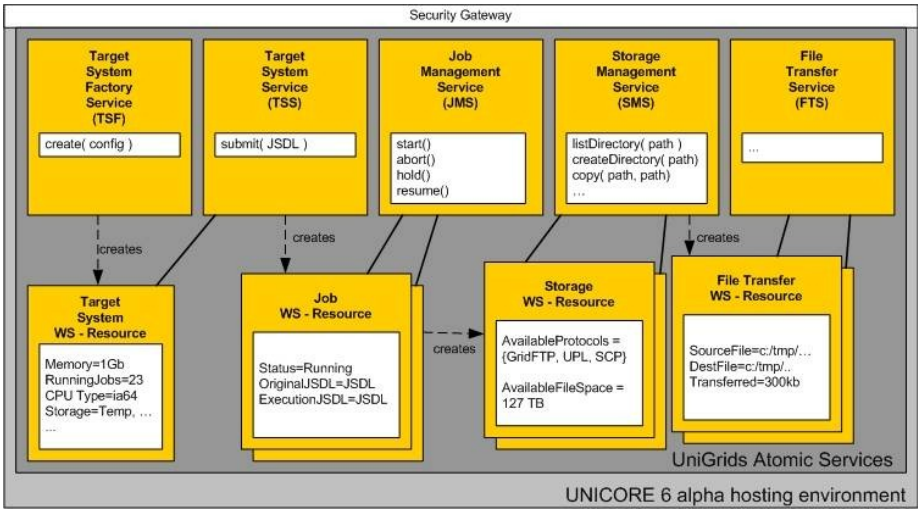


Illustration 2: Overall Unicore Structure

More precisely, the UAS consists of several stateful Web services that allow for the submission and management of computational jobs, easy access to storage resources, and file transfers between systems. UnicoreGS consists not only of a WS-RF implementation, but also of a hosting environment that is capable of hosting WS-RF compliant services and their stateful WS-Resources. The implementation of the services and the hosting environment base upon the Simple Object Access Protocol (SOAP) engine Apache eXtensible Interaction System (AXIS) in version 1.0, and a developed WS-RF and WS-N library in Java. Furthermore, additional handlers for the integration in AXIS were developed to support WS-Addressing compliant message headers.

Note that the execution backend on the target system is currently realized by an Network Job Supervisor (NJS) and Target System Interface (TSI) of the production UNICORE Grid system. Hence, existing well-tested technology is interfaced with WS-* technologies to provide standardized access, instead of proprietary protocols such as the UNICORE 5 Protocol Layer (UPL).

Within UNICORE 6 alpha, the production UNICORE Gateway component(? This is not true!) is enhanced to allow for SOAP messages that address WS-Resources and hosted Grid services. In addition, the Gateway is extensible to other Protocols, e.g. for dedicated streaming services or parallel HTTP that is currently being developed by the European NEXTGrid project. In particular, the default channel for the new Gateway dispatches incoming SOAP messages that understand WS-Addressing headers in the SOAP headers. Also, the new Gateway can use the request URL information from the HTTP transport layer to dispatch simple SOAP messages which do not include WS-Addressing headers and is thus quite similar to standard Web mechanisms, e.g. URL-Rewriting techniques.

For large data transfers, streamed data or for interactive applications requiring fast response times and low latencies, the characteristics of SOAP over HTTPS are usually unsatisfactory compared to application-specific proprietary protocols. Therefore, the new Gateway also provides extensions for streaming data without using SOAP and HTTP. The security within UNICORE 6 base upon X.509 certificates, a standard by IETF, but a de-facto standard within Grid environments. More precisely, every request to a service hosted within UNICORE 6 can be realized using transport-level security with X.509 credentials. However, the authentication of a user is checked by the usage of a WS-based Unicore User DataBase service (WS-UUDB) that checks if the certificates are signed by a Certification Authority (CA) that is being trusted. Also, the WS-UUDB provides authorization by mapping X.509 certificates to explicit xlogins and user accounts on the physical resources (e.g. supercomputer). Because of the integration of the Explicit Trust Delegation mechanisms into the UNICORE server components, also UNICORE 6 alpha provides a secure delegation of tasks on the behalf of the user to different Grid sites. Furthermore, the WS-UUDB service can be possibly shared within a virtual organizations and thus allows easier management of authentication and authorization issues in large virtual organizations. This means one centrally managed UUDB could exist that stores the needed certificates and CA information for all users of Grid middleware systems within a virtual organization.

3.2.2 Job Submission in Unicore

The submission and management of computational jobs are handled by the Target System Service (TSS) and the Job Management Service (JMS) respectively. the creation of the TSS is done through a Target System Factory (TSF) in a WS-RF factory pattern, which is defined as any service that is capable of bringing a WS-Resource into existence. The Target System WS-Resource models a physical Grid resource such as a supercomputer or cluster and exposes its state via WS-Resource Properties (WS-RP). These states can reach from the current CPU load, and available physical memory, to the total number of running jobs, processor information, and provided application-specific software. Hence, the Target System WS-Resource can be easily used for monitoring the current status of a system or to provide up-to-date information for remote brokering services. In addition, the TSS provides access to the Target System WS-Resource and defines a submit() operation. To submit computational jobs to through UnicoreGS, this operation can be used by adding as a parameter a job description compliant with the OGF standard Job Submission Description Language (JSDL). Internally, the JSDL job descriptions will be mapped to UNICORE proprietary Abstract Job Objects

and the real execution will be delegated to the UNICORE server components, in particular the NJS and the underlying TSI.

Note that JSDL is a small subset of AJO, but studies into more powerful orchestration and workflow languages such as Business Process and Execution Language (BPEL) already started. However, the submitted JSDL will be stored as state of a Job WS-Resource that is internally created by the submit() operation and represents a model of the executed job of the target system. The Job WS-Resource exposes also state properties such as the status of the job (e.g. running, queued, or executing), and the working directory of the job. Hence, this information can be used when using Secure Shell (SSH) connections to examine the job output in the working directory during execution, or the start of application-specific software that analyses the execution of the job, for instance debuggers or performance testing tools. The Job WS-Resources can be accessed via the JMS which enables job control operations upon submitted jobs, for instance hold(), resume(), or abort(). Also, this Job WS - Resources provide a sophisticating base for exposing their state to other monitoring services in a standard compliant way and thus allow innovations of easier developments of monitoring services.

Apart from the above job submission function, UnicoreGS also provides service that are related to storage and file management. A Storage Management Service (SMS) can be used to access Storage WS-Resources that are automatically created by the TSS at startup of UnicoreGS. These Storage WS-Resources expose properties like the underlying filesystem or a list of supported protocols, which indicate the protocols that can be used for file transfers, for instance GridFTP, Secure Copy (SCP) or UPL. UPL as a protocol indicates that UNICORE 6 is designed to allow interaction with UNICORE 5 systems if system administrators decide not to change their proprietary Grid middleware to service-oriented middleware. However, the import(), copy() and export() operations of the SMS can be used to create File Transfer WS-Resources and to initiate the correspondend operations on the storage devices. Desirable features of huge file transfers include, in addition to security, monitoring capabilities of the amount of transferred bytes. This is realized by exposing this information within a File Transfer WS-Resource, along with further information such as the source or destination of files.

3.3 Globus

The WS GRAM software implements a solution to the job-management problem, providing Web services interfaces consistent with the WSRF model. This solution is specific to operating systems following the Unix programming and security model.

WS GRAM combines job-management services and local system adapters with other service components of GT 4.0 in order to support job execution with coordinated file staging.

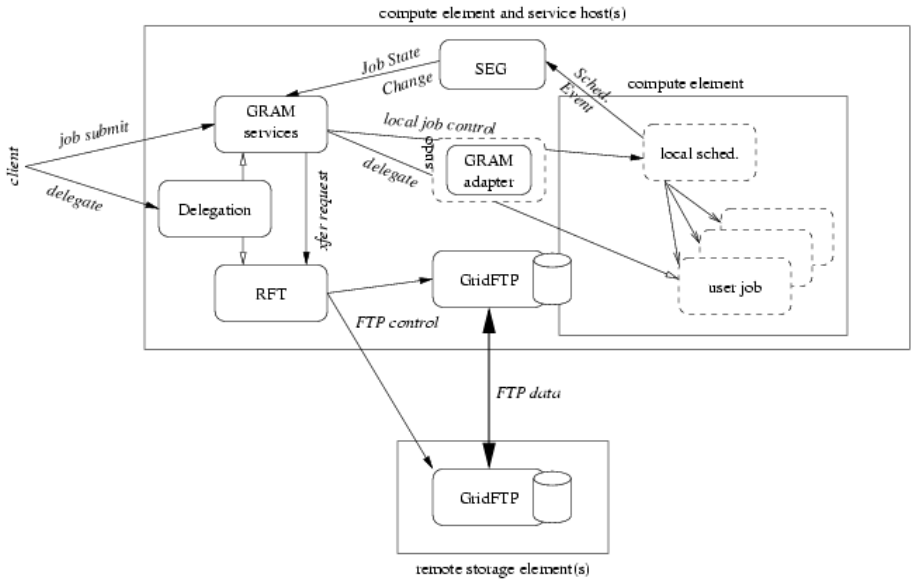


Illustration 3: GRAM Job Submission

The heart of the WS GRAM service architecture is a set of Web services designed to be hosted in the Globus Toolkit's WSRF core hosting environment. Note, these services, described below, make use of platform-specific callouts to other software components.

The components in the WS GRAM solution are organized to support a range of optional features that together address different usage scenarios. These scenarios are explored in depth in terms of protocol exchanges in the Protocol Variations section. However, at a high level we can consider the main client activities around a WS GRAM job to be a partially ordered sequence.

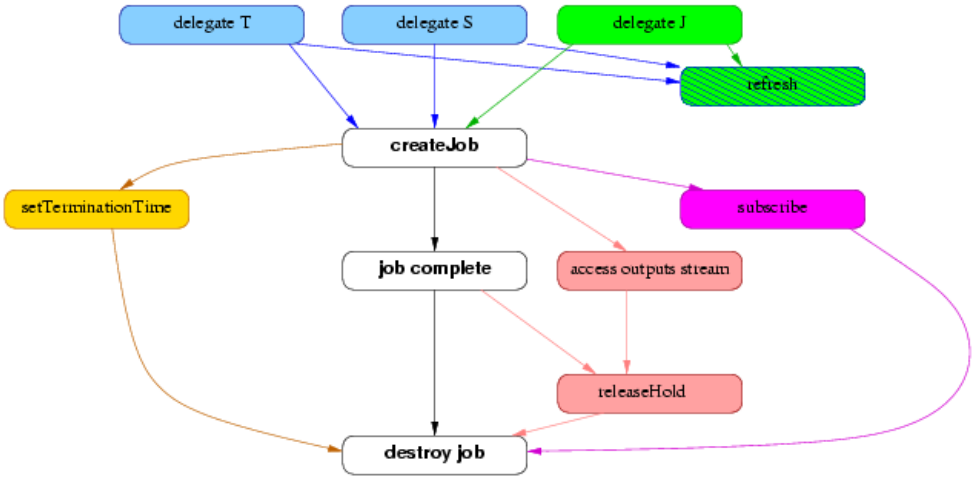


Illustration 4: GRAM Client Activities

The main component of the WS GRAM service model is the ManagedJob resource created by a ManagedJobFactory::createManagedJob invocation. A meaningful WS GRAM client MUST create a job that will then go through a life cycle where it eventually completes execution and the resource is eventually destroyed (the core black-and-white nodes in the high-level picture).

Optionally, the client MAY request staging activities to occur before or after the job. If these are requested in the create call, suitable delegated credential EPRs MUST be passed in as part of the creation input, meaning that delegation operations MUST be performed sometime before createManagedJob when staging is enabled (the light-blue delegation nodes in the high-level picture). Two credential fields must be initialized: the staging and transfer credentials, which may refer to distinct credentials or may both refer to the same credential. The staging credential gives WS GRAM the right to interact with the RFT service, while the transfer credential gives RFT the right to interact with GridFTP servers.

Optionally, the client MAY request that a credential be stored into the user account for use by the job process. When this is requested in the create call, a suitable delegated credential EPR is passed as part of the creation input. As for staging, the credential MUST have been delegated before the job is created (the green nodes in the picture).

Optionally, credentials delegated for use with staging, transfer, or job processes may be refreshed using the Delegation service interface. This operation may be performed on any valid Delegation EPR (the blue/green striped node in the picture).

If the client wishes to directly access output files written by the job (as opposed to waiting for the stage-out step to transfer files from the job host), the client should request that the file cleanup process be held until released. This gives the client an opportunity to fetch all remaining/buffered data after the job completes but before the output files are deleted. (See the pink nodes in the high-level picture).

The cleanup hold and release are not necessary if the client will not be accessing files that are scheduled for cleanup in the job request, either because the client is not accessing any files or because the files it is accessing will remain on the job host after ManagedJob termination.

Under nearly all circumstances, ManagedJob resources will be eventually destroyed after job cleanup has completed. Clients may hasten this step via an explicit destroy request or by manipulation of the scheduled termination time. Most system administrators will set a default and maximum ManagedJob linger time after which automatic purging of completed ManagedJob resources will occur.

4 Requirements for JSDL extensions

4.1 gLite-specific Requirements

Comparing the gLite JDL specification with JSDL we observe that some information which are defined in the JDL, and which are needed for the correct description of gLite jobs, have no equivalent JSDL attribute. For those elements, JSDL extensions have been identified. Moreover, there are also JSDL attributes which do have a mapping to JDL ones, but while the former are optional (i.e., the minimum multiplicity is set to zero), the latter are not (i.e., the minimum multiplicity is one).

We identified two classes of JDL attributes: those indicating how the gLite WMS has to handle the job, and those really describing the job itself. Attributes characterizing the job could be good candidates for a first simple JSDL 'gLite extension', while the others (WMS-related) should be treated as new arguments of the job submission service operations, and thus outside the scope of this document.

The following table describes the JDL attributes which are not related to the job itself, but are instead specific of the gLite infrastructure used to submit and execute the job. The following JDL attributes are not suited as JSDL extensions

JDL Attribute Name	*Description*
`ExpiryTime`	A job for which no compatible CEs have been found during the matchmaking phase is hold in the WMS Task Queue for a certain time (currently it is 1 day from job submission) so that it can be subjected again to matchmaking from time to time until a compatible CE is found. If no match is found after 1 day the job is aborted. The `ExpiryTime` attribute is an integer representing the date and time (in seconds since epoch) until the job has to be considered valid by the WMS
`RetryCount`	This attribute is an integer representing the maximum number of deep job re-submissions to be done in case of failure due to some grid component (i.e. not to the job itself). Job resubmission is defined *deep* when the user s job has started running on the WN and then the job itself or the WMS JobWrapper has failed. It is instead defined *shallow* when the WMS JobWrapper has failed before starting the actual user s job.
`ShallowRetryCount`	The `ShallowRetryCount` attribute is an integer representing the maximum number of shallow job resubmissions to be done in case of failure due to some grid component (i.e. not to the job itself).
`MyProxyServer`	The `MyProxyServer` attribute specifies the hostname of a MyProxy server where the user has registered her/his long-term proxy certificate. A MyProxy server can be used to store a long-lived user certificate which can be used by the WMS to renew the lifetime of a standard user certificate proxy (usually valid only for 12 hours). Long-running jobs may run into this limit and fail due to expiration of the user proxy. To avoid this from happening, the user can store a certificate on a MyProxy server by issuing

HLRLocation	The HLRLocation attribute is a string representing the user Home Location Register address in the format <host fqdn>:<port>:[<X509contact string>] HLR is the service responsible for managing the economic transactions and the accounts of user and resources. The presence of the HLRLocation attribute in the JDL enables accounting for the job, i.e. on the CE, while the job runs, a sensor monitors the resource usage and when the job is done those data (usage records) are sent to the HLR specified through the HLRLocation attribute the HLR computes the job cost according to the usage records and to the resource price and then debits the user account
JobProvenance	The JobProvenance attribute is a string representing the endpoint URI of the Job Provenance service where data about the job have to be stored, e.g. JobProvenance = "https://lindir.ics.muni.cz:10001" The specification of this attribute in the job description makes the WMS feed the Job Provenance service with the job sandbox files. This attribute is not mandatory. A default value for this attribute can be specified in the WMProxy
FuzzyRank	The FuzzyRank attribute is a Boolean attribute that enables *fuzzyness* in the ranking computation. In other words if this attribute is set to true, it forces the matchmaking algorithm to adopt a stochastic selection criteria while searching for the best matching CE

Table 1: gLite JDL attributes not related to the Job description

4.1.1 gLite JSDL extensions

We show in Table 2 the JDL attributes which are used to describe a job in gLite, and thus are candidate for being included as "gLite extensions for JSDL". We specify the JDL Attribute name with the corresponding Description; more information can be found in the JDL specification . Note that the only mandatory element in Table 2 is the "VirtualOrganization" tag. All other elements are optional.

JDL Attribute name	*Description*
Prologue	The Prologue attribute is a string representing the executable/script name of the *prologue*. The *prologue* is an executable run within the WMS job wrapper before the user job is started. It can be used for purposes ranging from application-specific checks that the job environment has been correctly set on the WN to actions like data transfers, database updates or MPI pre script. If the prologue fails the job wrapper terminates and the job is considered for resubmission
Epilogue	The Epilogue attribute is a string representing the executable/script name of the *epilogue*. The *epilogue* is an executable/script run within the WMS job wrapper after the user job completion. It can be used for purposes ranging from application-specific checks that the job performed correctly to actions like data transfers, database updates or MPI post script

DataRequirements	The DataRequirements attribute is a list of classads representing the data requirements for a job. Each classad in the list contains three attributes: InputData, DataCatalogType and DataCatalog, representing respectively the list of input data needed by the job, the type of data catalog that has to be targeted to resolve logical names to physical names and lastly the URI of the data catalog if this is not the VO default one (endpoint known through service discovery or configuration). The form of this attribute allows users to target experiment-specific catalogs for their jobs and to mix different input data types supported by different data catalogs in the same job description. Note that it is possible to specify more than once the same catalog type e.g. for using the VO default
OutputSE	The OutputSE attribute is a string representing the URI of the Storage Element where the user wants to store the output data. Once specified, this attribute is used by the RB to find a CE being "close" to this SE and schedule the job there
UserTags	The UserTags attribute is a classad attribute that allows the user to specify user-defined key, value pairs (where the value must be a string) that are logged at submission time to the LB and are associated to the job in the LB database. The specified user tags can be then used to build conditions when querying the LB for the status of submitted jobs
VirtualOrganisation	The VirtualOrganisation attribute is a string representing the name of the VO the submitting user is currently working for. The value for this attribute has to match with the VO specified within the credentials issued by VOMS or with the possible options provided by the WMProxy client commands. This element MUST be specified in the JDL.
Requirements	The Requirements attribute is a boolean ClassAd which describes the requirements for a job on resources. To have a job executed on a given Computing Element (CE), the Requirements classad MUST evaluate to true on that CE.

Table 2: gLite JDL attributes related with job description

4.1.2 Mapping between JSDL and JDL attributes

We now describe the mapping between JSDL and JDL attributes; Table 3 shows which JSDL attributes correspond to JDL attributes. For the JDL elements described in Table 2 we define additional JSDL elements in the jsdl-glite namespace. Other information which are required by gLite to process a job are taken from JSDL or JSDL-Posix elements as shown. The "Required" column indicates whether the JSDL attribute MUST be present in the JSDL in order to be correctly processed by the gLite infrastructure.

According to the JSDL specification, every JSDL element allows additional attributes or sub-elements, the only requirement being that those additional attributes/elements MUST be in a different namespace than the normative namespace defined for JSDL. For gLite, we define the namespace jsdl-glite="http://schemas.glite.org/jsdl/2006/1/jsdl-glite".

JSDL element	JDL attribute	Required?	Notes
jsdl:JobProject	VirtualOrganisation	Yes	
jsdl-posix:Executable	Executable	Yes	
jsdl-posix:Argument	Arguments	No	
jsdl-posix:Environment	Environment	No	
jsdl-posix:Input	StdInput	No	
jsdl-posix:Output	StdOutput	No	
jsdl-posix:Error	StdError	No	
jsdl:Resources	Requirements	No	
jsdl:DataStaging	InputSandbox	No	Only when jsdl:Source child is present
jsdl:DataStaging	OutputSandbox	No	Only when jsdl:Target child is present
jsdl:DataStaging	DataRequirements	No	Only when has jdl:name attribute; gLite extension
jsdl-glite:DataAccessProtocol	DataAccessProtocol	No	gLite extension
jsdl-glite:Prologue	Prologue	No	gLite extension
jsdl-glite:Epilogue	Epilogue	No	gLite extension
jsdl-glite:UserTags	UserTags	No	gLite extension
jsdl-glite:OutputSE	OutputSE	No	gLite extension

Table 3: Mapping between JSDL and JDL attributes

From Table 3 we observe that, in order to be processed by gLite, a JSDL must define suitable values for the jsdl:jobProject and jsdl-posix:Executable attributes, although these elements are optional in the JSDL schema (i.e., they have minimum multiplicity equal to zero).

The JSDL provides a way for specifying job requirements, using the jsdl:Resources element. The JSDL specification describes a number of sub-elements which may be present in the jsdl:Resources element. The jsdl:Resources element maps to the Requirements JDL attribute; however, the content of jsdl:Resources must be translated into the corresponding boolean ClassAd, containing GlueSchema attributes which refer to individual resource names.

Table 4 shows the mapping between some jsdl:Resources sub-elements and GlueSchema attribute names . jsdl:Resources sub-elements not appearing on the table do not have any corresponding GlueSchema attribute, and thus are ignored by gLite.

JSDL element name	GlueSchema attribute	Notes
jsdl:CandidateHosts	GlueCEInfoHostName	
jsdl:CPUArchitecture	GlueHostProcessorModel	

jsdl:IndividualPhysicalMemory	GlueHostMainMemorySize	
jsdl:IndividualVirtualMemory	GlueHostVirtualMemorySize	
jsdl:IndividualCPUSpeed	GlueHostProcessorSpeed	
jsdl:IndividualCPUTime	GlueCEMaxCpuTime	
jsdl:OperatingSystem	GlueHostOperatingSystemName	
jsdl:OperatingSystemVersion	GlueHostOperatingSystemVersion	
jsdl-glite:LRMSType	other.GlueCEInfoLRMSType	gLite extension

Table 4: Mapping between JSDL and GlueSchema elements.

4.2 Unicore-specific requirements

4.2.1 About UNICORE JOB and AJO

As mentioned above, a UNICORE job is realized in an abstract representation of a job group – a job that may recursively contain other job groups and/or tasks, which is called the Abstract Job Object (AJO), they are stored at user's workstation as a serialized Java object and/or in XML format. Tasks contained in a job group are incarnated into a batch job to be executed on a system at the site or into an action, like a file transfer to a storage space. Child jobs groups are transferred to the appropriate site to be incarnated and executed there. The user may specify temporal dependencies between the entities contained in a job group.

The Java AJO class library is the basis for the modelling of UNICORE jobs and for the protocol between the UNICORE clients and the servers. It contains the following components:

1 Basic UNICORE protocol layer (UPL) for Client<->Gateway and Gateway<->Server communication on top of SSL

2 Resource object hierarchy for the specification of resource requests (need certain resources) and available resources (can provide certain resources)

3 Object hierarchy of abstract actions containing the definition of all supported tasks – user and system tasks – as well as for abstract UNICORE (sub-)jobs.

4.2.2 JSDL to AJO mapping

In Unigrids project, JSDL is formally adopted as a normative way to submit jobs through job submission Web services interface, since it is defined by GGF JSDL-WG and is a specification for an abstract standard about job submission that is independent to language bindings. The mapping between these two concepts as illustrated as follows:

JSDL	*AJO*
JobDescription{0,n}	AbstractAction.ActionGroup.AbstractJob
JobIdentification{0,1}	

JSDL	*AJO*
JobName{0,1}	AbstractAction.ActionGroup.AbstractJob.name
Description{0,1}	
JobAnnotation{0,n}	AbstractAction.notes
JobProject{0,n}	Utility.UserAttributesConverter.project
Application{0,1}	
ApplicationName{0,1}	Resource.CapabilityResource.SoftwareResource.Application | AbstractAction.ActionGroup.AbstractJob.JSDLImpl | application.name
ApplicationVersion{0,1}	Resource.CapabilityResource.SoftwareResource.Application | AbstractAction.ActionGroup.AbstractJob.JSDLImpl | application.version | AbstractAction.ActionGroup.AbstractJob.JSDLImpl
Description{0,1}	application.description | userApplication.description
POSIXApplication{0,1}	
Executable{0,1}	userApplication.executable
Argument{0,n}	application.arguments | userApplication.arguments
Input{0,1}	application.stdin | userApplication.stdin
Output{0,1}	application.stdout | userApplication.stdout
Error{0,1}	application.stderr | userApplication.stderr
WorkingDirectory{0,1}	AbstractAction.ActionGroup.AbstractJob.JSDLImpl
Environment{0,n}	environmentVariable

JSDL	*AJO*
WallTimeLimit{0,1}	Resource.CapabilityResource.Limit limit & type.Wall_time AbstractAction.ActionGroup.AbstractJob. JSDLImpl resources.wall_time_limit Resource.CapabilityResource.Limit
FileSizeLimit{0,1}	limit & type.File_size
CoreDumpLimit{0,1}	limit & type.Core_dump
DataSegmentLimit{0,1}	limit & type.Data_segment
LockedMemoryLimit{0,1}	limit & type.Locked_memory
MemoryLimit{0,1}	limit & type.Memory_rss
OpenDescriptorsLimit{0,1}	limit & type.Open_descriptors
PipeSizeLimit{0,1}	limit & type.Pipe_size
StackSizeLimit{0,1}	limit & type.Stack_size
CPUTimeLimit{0,1}	limit & type.CPU_time AbstractAction.ActionGroup.AbstractJob. JSDLImpl resources.cpu_time_limit
ProcessCountLimit{0,1}	limit & type.Process_count
VirtualMemoryLimit{0,1}	limit & type.Virtual_memory | AbstractAction.ActionGroup.AbstractJob. JSDLImpl resources.virtual_memory_limit
ThreadCountLimit{0,1}	limit & type.Thread_count
UserName{0,1}	Utility.UserAttributesConverter.xlogin
GroupName{0,1}	
Resources{0,1}	
CandidateHosts{0,1}	
HostName{1,n}	Vsite
FileSystem{0,n}	Resource.CapacityResource.Storage.Pathe dStorage AlternativeUspace | Home | Root | StorageServer | Temp| USpace

JSDL	*AJO*
Description{0,1}	
MountPoint{0,1}	
MountSource{0,1}	
DiskSpace{0,1}	
FileSystemType{0,1}	
ExclusiveExecution{0,1}	
OperatingSystem{0,1}	
OperatingSystemType{0,1}	
OperatingSystemName{1,1}	
OperatingSystemVersion{0,1}	
CPUArchitecture{0,1}	
CPUArchitectureName{1,1}	
IndividualCPUSpeed{0,1}	
IndividualCPUTime{0,1}	AbstractAction.ActionGroup.AbstractJob.JSDLImpl resources.cpu_count
IndividualCPUCount{0,1}	
IndividualNetworkBandwidth{0,1}	Resource.CapacityResource.PerformanceResource.Network performance
IndividualPhysicalMemory{0,1}	
IndividualVirtualMemory{0,1}	AbstractAction.ActionGroup.AbstractJob.JSDLImpl resources.virtual_memory_limit Resource.CapacityResource Memory.request
IndividualDiskSpace{0,1}	
TotalCPUTime{0,1}	
TotalCPUCount{0,1}	AbstractAction.ActionGroup.AbstractJob.JSDLImpl resources.cpu_count

JSDL	*AJO*
TotalPhysicalMemory{0,1}	
TotalVirtualMemory{0,1}	AbstractAction.ActionGroup.AbstractJob.JSDLImpl resources.virtual_memory_limit | Resource.CapacityResource.Memory.request
TotalDiskSpace{0,1}	
TotalResourceCount{0,1}	Resource.CapacityResource.Node.request
DataStaging{0,n}	
FileName{1,1}	AbstractAction.ActionGroup.AbstractJob.JSDLImpl stageInFile.local_name | stageOutFile.local_name
FileSystemName{0,1}	stageInFile.local_file_system_name | stageOutFile.local_file_system_name
CreationFlag{1,1}	stageInFile.overwrite | stageOutFile.overwrite
DeleteOnTermination{0,1}	
Source{0,1}	
URI{1,1}	stageInFile.vsite | stageInFile.file_system_name | | stageInFile.file_name
Target{0,1}	
URI{1,1}	stageOutFile.vsite | stageOutFile.file_system_name | stageOutFile.file_name

Table 5: JSDL to AJO mapping

Through this map, most of the conversions can be easily achieved internally by a library called JSDL2AJO.jar, however, there are two major issues that the conversion is struggling with: dependency and resource description.

The AJO has the capability of workflow and dependency in between sub jobs and parent job with sub jobs, but in the current version of JSDL, apart from some standard functions, there is no way that the specification can fully accomplish this mission.

JSDL has only defined Posix Application as an application resource, in UNICORE case, it is quite often that other applications like POV-Ray and shallow can only find their place inside <Application> tag as xsd:any. It will be very convenient if the JSDL extension can at least define a set of application type that will be used quit often, if the full schema for that application type is too heavy for the current JSDL working group.

4.3 Globus-specific requirements

JSDL is an accepted GGF standard specification for describing grid jobs. Since JSDL became a standard, many groups have started to use it. To accommodate this emerging standard, we propose to extend our factory web service interface with an operation that takes a JSDL document, while leaving the original createManagedJob operation the same. This will allow users to better plan when they take on the transition from the current WS GRAM job description schema to JSDL.

Then general plan is to add new operations and resource properties required to support JSDL to the current WS GRAM service. Resulting in a single WS GRAM service that is capable of processing a job specified using the current JDD schema as well as a job specified using the JSDL schema. Clients written before these JSDL enhancements will be able to interoperate with the new WS GRAM service that supports both the current job description schema and JSDL.

The gram clients: globusrun-ws and gramJob java API will be updated to identify the job as either JDD or JSDL and use the matching WS GRAM service operations. The globusrun-ws job wrapper scripts (globus-job-*-ws) will not be altered and will continue to specify job's in JDD format.

5 Concluding Statement

In this document we proposed JSDL extensions to satisfy the requirements of the gLite, Globus and UNICORE Grid systems. We started by analyzing the current job description notations used within these systems; then, we described how each job description notation can be mapped into a JSDL specification. From the features of the original notation which are missing in JSDL we derived new JSDL elements, which can be qualified as part of Grid specific JSDL extensions.

6 References

[RFC 2119] Bradner, S. *Key words for use in RFCs to Indicate Requirement Levels*. Internet Engineering Task Force, RFC 2119, March 1997. Available at http://www.ietf.org/rfc/rfc2119.txt

[JDL] *Job Description Language Attributes Specification for the Glite Middleware (Submission Through WMProxy Service)*, EGEE Document EGEE-JRA1-TEC-590869-JDLAttributes-v0-7, https://edms.cern.ch/document/590869/1

[JSDL] A. Savva (Editor), *Job Submission Description Language (JSDL) Specification*, Version 1.0, http://www.gridforum.org/documents/GFD.56.pdf

[GLUE] GLUE Schema Specification, Version 1.2, Final Specification, http://glueschema.forge.cnaf.infn.it/uploads/Spec/GLUEInfoModel_1_2_final.pdf

[GRAM-JDL] *WS GRAM: Job Description Schema* http://www.globus.org/toolkit/docs/4.0/execution/wsgram/schemas/gram_job_description.html

[WS-GRAM] *GT 4.0 WS GRAM Approach,* http://www.globus.org/toolkit/docs/4.0/execution/key/WS_GRAM_Approach.html

[XML Schema1] XML Schema—Part 1: Structures, W3C Recommendation 28 October 2004, http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/

[XML Schema2] XML Schema—Part 2: Datatypes, Second Edition, W3C Recommendation 28 October 2004, http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/

7 Appendix

7.1 XML schema for gLite extension

```
<?xml version="1.0" encoding="UTF-8"?>
<schema
    targetNamespace="http://schemas.glite.org/jsdl/2006/1/glite-jsdl"
    xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:jsdl-glite="http://schemas.glite.org/jsdl/2006/1/glite-jsdl"
    xmlns:jsdl="http://schemas.ggf.org/jsdl/2005/11/jsdl"
    xmlns:jsdl-posix="http://schemas.ggf.org/jsdl/2005/11/jsdl-posix">
    <import namespace="http://schemas.ggf.org/jsdl/2005/11/jsdl"
           schemaLocation="jsdl-schema.xsd"></import>
    <import namespace="http://schemas.ggf.org/jsdl/2005/11/jsdl-posix"
           schemaLocation="jsdl-posix-schema.xsd"></import>
    <element name="DataAccessProtocol"
           type="jsdl-glite:DataAccessProtocol_type">
    </element>

    <element name="Prologue" type="string"></element>

    <element name="Epilogue" type="string"></element>

    <simpleType name="DataAccessProtocol_type">
           <restriction base="string">
                  <enumeration value="gsiftp"></enumeration>
                  <enumeration value="nfs"></enumeration>
                  <enumeration value="afs"></enumeration>
                  <enumeration value="rfio"></enumeration>
                  <enumeration value="gsirfio"></enumeration>
                  <enumeration value="dcap"></enumeration>
                  <enumeration value="gsidcap"></enumeration>
                  <enumeration value="root"></enumeration>
                  <enumeration value="https"></enumeration>
                  <enumeration value="other"></enumeration>
           </restriction>
    </simpleType>

    <element name="UserTags" type="string"></element>

    <element name="LRMSType" type="normalizedString"></element>

    <element name="OutputSE" type="anyURI"></element>

    <element name="DataCatalogEndPoint" type="anyURI"></element>
</schema>
```

7.2 XML schema for Globus extension

7.2.1 jsdl-gram.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  Copyright 1999-2006 University of Chicago

  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

  http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License.
-->
<xsd:schema
    targetNamespace="http://schemas.globus.org/gram/2006/07/jsdl-gram"
    xmlns:tns="http://schemas.globus.org/gram/2006/07/jsdl-gram"
    xmlns:jsdl="http://schemas.ggf.org/jsdl/2005/11/jsdl"
    xmlns:rft="http://www.globus.org/namespaces/2004/10/rft"
    xmlns:types="http://www.globus.org/namespaces/2004/10/gram/job/types"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
    xmlns:wsnt="http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.2-
draft-01.xsd"
    elementFormDefault="qualified"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <xsd:import
        namespace="http://schemas.ggf.org/jsdl/2005/11/jsdl"
        schemaLocation="../../gram/4.2/jsdl.xsd"/>

    <xsd:import
        namespace="http://www.globus.org/namespaces/2004/10/gram/job/types"
        schemaLocation="../../gram/4.0/managed_job_types.xsd"/>


    <xsd:import
        namespace="http://www.globus.org/namespaces/2004/10/rft"
        schemaLocation="../../transfer/reliable/reliable_transfer_types.xsd"/>

    <xsd:import
        namespace="http://schemas.xmlsoap.org/ws/2004/03/addressing"
        schemaLocation="../../ws/addressing/WS-Addressing.xsd"/>

    <xsd:import
        namespace="http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.2-
draft-01.xsd"
        schemaLocation="../../wsrf/notification/WS-BaseN.xsd"/>

     <xsd:simpleType name="SoftwareEnvironmentManagerEnumeration">
            <xsd:restriction base="xsd:string">
                   <xsd:enumeration value="SoftEnv"/>
                   <xsd:enumeration value="Modules"/>
                   <xsd:enumeration value="other"/>
            </xsd:restriction>
     </xsd:simpleType>
```

```
<xsd:simpleType name="JobTypeEnumeration">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="single"/>
        <xsd:enumeration value="multiple"/>
        <xsd:enumeration value="array"/>
        <xsd:enumeration value="mpi"/>
        <xsd:enumeration value="condor"/>
        <xsd:enumeration value="other"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="SubjobDescriptions_Type">
    <xsd:sequence>
        <xsd:element ref="jsdl:JobDescription" maxOccurs="unbounded"/>

        <xsd:any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other" processContents="lax"/>
</xsd:complexType>

<xsd:complexType name="Holds_Type">
    <xsd:sequence>
        <xsd:element ref="tns:State" minOccurs="0" maxOccurs="unbounded"/>

        <xsd:any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other" processContents="lax"/>
</xsd:complexType>

<xsd:complexType name="Resources_Type">
    <xsd:sequence>
        <xsd:element ref="jsdl:FileSystem"
            minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element ref="jsdl:Resources" maxOccurs="unbounded"/>

               <xsd:any namespace="##other" processContents="lax"
            minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:anyAttribute namespace="##other" processContents="lax"/>
</xsd:complexType>

<xsd:complexType name="HostAttributes_Type">
    <xsd:sequence>
        <xsd:element ref="tns:Attribute" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="SoftwareEnvironment_Type">
    <xsd:sequence>
        <xsd:element ref="tns:Key" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="manager"
                   type="tns:SoftwareEnvironmentManagerEnumeration"
                   use="required"/>
    <xsd:anyAttribute namespace="##other" processContents="lax"/>
</xsd:complexType>


<!-- Holds_Type elements -->
<xsd:element name="State"              type="types:StateEnumeration"/>
```

```
<!-- jsdl-gram:HostAttributes elements -->
<xsd:element name="Attribute"           type="xsd:string"/>

<!-- tns:EnvironmentSetup_Type elements -->
<xsd:element name="Key"                 type="xsd:string"/>


<!-- jsdl:JobDescription extension elements -->

<xsd:element name="Resources" type="tns:Resources_Type">
    <xsd:annotation>
        <xsd:documentation>
            A replacement for jsdl:Resources that allows more complex
            composition of resource requirements by allowing for an array
            of jsdl:Resources elements.
        </xsd:documentation>
    </xsd:annotation>
</xsd:element>

<xsd:element name="JobType" type="tns:JobTypeEnumeration">
    <xsd:annotation>
        <xsd:documentation>
            This specifies how the resource manager should start the job.

            single
            ------
            Submit the job with a reservation of N nodes and leave
            responsibility for utilizing the nodes up to the job process.

            multiple
            --------
            The submission method depends on whether the compute resource is
            an SMP machine or a cluster.
            If the resource is an SMP machine, reserve M processors and
            start N copies of the executable via a single local job
            submission.
            If the resource is a cluster, reserve M hosts and submit
            a script job that starts N copies of the executable on the
            reserved hosts in a round robin fashion.

            array
            -----
            Submit N copies of the job.

            mpi
            ---
            Submit an appropriate mpirun or mpiexec job that starts N copies
            of the executable via MPI.

            condor
            ------
            Submit a 'standard' universe job (Condor resource manager only).

            other
            -----
            A user-defined method of starting the job.
        </xsd:documentation>
    </xsd:annotation>
</xsd:element>

<xsd:element name="Queue" type="xsd:string">
    <xsd:annotation>
```

```
            <xsd:documentation>
                The name of the queue that the job should be submitted to.
            </xsd:documentation>
        </xsd:annotation>
    </xsd:element>

    <xsd:element name="Holds" type="tns:Holds_Type">
        <xsd:annotation>
            <xsd:documentation>
                A list of states on which pre-holds should be placed upon
                submission.
            </xsd:documentation>
        </xsd:annotation>
    </xsd:element>

    <xsd:element name="AuthorizationSubject" type="xsd:string">
        <xsd:annotation>
            <xsd:documentation>
                The certificate subject that will be expected from each subjob
                in the multijob.
            </xsd:documentation>
        </xsd:annotation>
    </xsd:element>

    <xsd:element name="SubjobDescriptions" type="tns:SubjobDescriptions_Type">
        <xsd:annotation>
            <xsd:documentation>
                A list of jsdl:JobDescription elements that represent subjobs of
                a multijob.
            </xsd:documentation>
        </xsd:annotation>
    </xsd:element>

    <!-- jsdl:Resources extension elements -->

    <xsd:element name="SoftwareEnvironment" type="tns:SoftwareEnvironment_Type">
        <xsd:annotation>
            <xsd:documentation>
                A list of keys that each execution host must use to setup the
                software environment via the specified software environment
                manager.
            </xsd:documentation>
        </xsd:annotation>
    </xsd:element>

    <xsd:element name="HostAttributes" type="tns:HostAttributes_Type">
        <xsd:annotation>
            <xsd:documentation>
                A list of names for attributes that a candidate execution host
                must have.
            </xsd:documentation>
        </xsd:annotation>
    </xsd:element>

    <!-- jsdl:DataStaging_Type extension elements -->

    <xsd:element name="ExecutionHostStaging" type="xsd:boolean">
        <xsd:annotation>
            <xsd:documentation>
                Enable second-tier staging from the service host to the
```

```
                execution host(s).
            </xsd:documentation>
        </xsd:annotation>
    </xsd:element>

    <!-- jsdl:Application_Type extension elements -->

    <xsd:element name="IndividualProcessCount" type="xsd:int">
        <xsd:annotation>
            <xsd:documentation>
                The number of processes to spawn per allocated resource.
                See jsdl:Resources's TotalResourceCount element.
            </xsd:documentation>
        </xsd:annotation>
    </xsd:element>

    <xsd:element name="TotalProcessCount" type="xsd:int">
        <xsd:annotation>
            <xsd:documentation>
                The number of processes to spawn in total accross all allocated
                resources.
            </xsd:documentation>
        </xsd:annotation>
    </xsd:element>
</xsd:schema>
```

7.2.2 jsdl-rft.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  Copyright 1999-2006 University of Chicago

  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

  http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License.
-->
<xsd:schema
    targetNamespace="http://schemas.globus.org/gram/2006/07/jsdl-rft"
    xmlns:tns="http://schemas.globus.org/gram/2006/07/jsdl-rft"
    xmlns:rft="http://www.globus.org/namespaces/2004/10/rft"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
    elementFormDefault="qualified"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <xsd:import
        namespace="http://www.globus.org/namespaces/2004/10/rft"
        schemaLocation="../../transfer/reliable/reliable_transfer_types.xsd"/>
```

```
    <xsd:import
        namespace="http://schemas.xmlsoap.org/ws/2004/03/addressing"
        schemaLocation="../../ws/addressing/WS-Addressing.xsd"/>

    <!-- jsdl:JobDescription extension elements -->
    <xsd:element name="StageInConcurrency"  type="xsd:int"/>
    <xsd:element name="StageOutConcurrency" type="xsd:int"/>
    <xsd:element name="DeleteConcurrency"   type="xsd:int"/>

    <!-- jsdl:DataStaging extension elements -->
    <xsd:element name="FinishBy"            type="xsd:dateTime"/>
    <xsd:element name="Attempts"            type="xsd:int"/>
    <xsd:element name="Size"                type="xsd:long"/>
    <xsd:element name="Binary"              type="xsd:boolean"/>
    <xsd:element name="BlockSize"           type="xsd:int"/>
    <xsd:element name="TCPBufferSize"       type="xsd:int"/>
    <xsd:element name="ThirdPartyTransfer"  type="xsd:boolean"/>
    <xsd:element name="ParallelStreams"     type="xsd:int"/>
    <xsd:element name="DataChannelAuthn"    type="xsd:boolean"/>
    <xsd:element name="SubjectName"         type="xsd:string"/>
    <xsd:element name="DestinationSubjectName"  type="xsd:string"/>
    <xsd:element name="SourceSubjectName"   type="xsd:string"/>
    <xsd:element name="IgnoreFilePermError" type="xsd:boolean"/>
</xsd:schema>
```