


EU project: RI031844-OMII-Europe

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:JRA1.9 Implementation of JSDL into OMII-Europe middleware together with the identified extensions

Due date of deliverable: Oct 2007
Actual submission date: Nov, 2007

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 | 2007-11-29 | First draft, many missing sections | Moreno Marzolla/INFN |
| 0.2 | 2007-12-18 | Second draft, filled in the CREAM-BES section. Added executive summary and introduction | Moreno Marzolla/INFN |
| 0.3 | 2008-01-23 | Added UNICORE 6 section | Sven van den Berghe/FLE |
| 0.4 | 2008-01-30 | Added references | Moreno Marzolla/INFN |




## Document Control Sheet

| | | |
|---|---|---|
| **Document** | **Title**: Implementation of JSDL into OMII-Europe middleware together with the identified extensions | |
| | **ID**: M:JRA1.9 | |
| | **Version**: 0.4 | **Status**: Draft |
| | **Available at**: | |
| | **Software Tool**: OpenOffice.org | |
| | **File(s):** OMII-EU-MJRA1.9.odt | |
| **Authorship** | **Written by:** | Moreno Marzolla (Editor) |
| | **Contributors:** | Sven van den Berghe<br>JRA1 Team |
| | **Reviewed by:** | TBD |
| | **Approved by:** | TBD |

## Document Status Sheet

| Version | Date | Status | Comments |
|---|---|---|---|
| 0.1 | 2007-11-29 | Draft | Initial CREAM-BES part (incomplete). Many missing sections |
| 0.2 | 2007-12-18 | Draft | Additions to the CREAM-BES part. Added executive summary and introduction. |
| 0.3 | 2008-01-23 | Draft | Added UNICORE 6 section |
| 0.4 | 2008-01-30 | Draft | Added references |

# Table of Contents

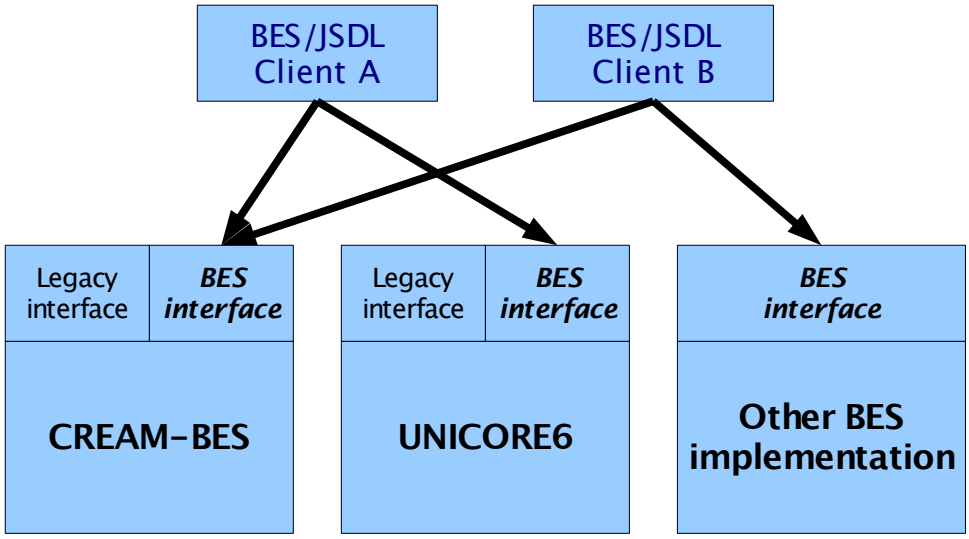
# 1 Executive Summary

This documents reports the implementation status and features of the JSDL support as found in OMII-Europe reengineered components. More specifically, in this document we considered the following job submission components re-engineered by OMII-Europe: CREAM-BES and UNICORE6. We describe the support for JSDL in these components; moreover, we highlight and motivate the implementation decisions, and discuss the current limitations and how these will be addressed in forthcoming software releases.

This document is both addressed at software developers, as this document is strongly focused on implementation details of JSDL in CREAM-BES and UNICORE6. These information are useful for those people writing client-side code to interact with the BES implementations.

# 2 Introduction to JSDL

The Job Submission Description Language (JSDL) [JSDL] is an XML-based 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. Drawing 1 below show the typical *interoperability* scenario for job submission using the BES/JSDL specifications. Note that different BES/JSDL compliant clients are able to submit jobs to any service exposing a BES interface [BES]; this is different from the typical *interoperation* approach between different Grid systems, which on the other hand would be based on ad-hoc, point-to-point adapters. Note that, in general, the same execution service may expose a "legacy" interface and a BES-compliant one. This is useful both for backward compatibility purposes, and because the BES/JSDL specifications allow for very basic functionalities only; for example, more complex job types (e.g., parametric jobs, or job collections with dependencies) are outside the scope of the current JSDL specification, while are supported by many execution services through their legacy interfaces.




*Drawing 1: Interoperability using BES/JSDL*

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 extension 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:

1. *Job Structure Elements*, which act as high-level containers of job description and requirements;

2. *Job Identity Elements*, which contains all elements that identify the job;
3. *Application Elements*, which describe among other things the name and version of the application to be executed;
4. *Resource Elements*, which describe the resources required by the job;
5. *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 POSIXApplication extension, which defines a schema describing an application executed on a POSIX compliant system.

Through this document we assume that the reader is familiar with the XML notation. Whenever possible, we will use a (simplified) XPath notation to identify specific JSDL elements. For example, the XPath expression `//jsdl:Application/jsdl:ApplicationName` denotes all jsdl:ApplicationName elements, which are direct childs of a jsdl:Application element, which is a descendant of the root element of the JSDL document. As another example, the XPath expression `/jsdl:JobDefinition/jsdl:JobDescription/jsdl:JobIdentification/*` denotes all child elements of the `jsdl:JobIdentification` element, which itself is child of `jsdl:JobDescription`, which in turn is child of `jsdl:JobDefinition` which must be the root JSDL element.

# 3 Implementation of JSDL in CREAM-BES

This section describes the implementation of JSDL support in the CREAM-BES job execution service. We start with a high level description of the overall structure of CREAM-BES and we show how JSDL support has been added to the legacy CREAM service used in the gLite 3.1 middleware. We then describe which features of the JSDL specification are supported by CREAM-BES, and finally give some examples of valid JSDL documents which can actually be processed by the CREAM-BES server.

## *3.1 Introduction to CREAM-BES*

CREAM-BES is a BES-enabled version of the CREAM Computing Element [CREAM], being developed by the EGEE/EGEE2 project for inclusion in the gLite 3.1 middleware distribution. CREAM is a Java-based computing execution service, which exposes a WebService-based interface. The "legacy" CREAM interface is not BES, and its standard job description notation if based on Condor classads rather than JSDL. The reason is that the development of CREAM started before the standardization work which lead to BES and JSDL started. Moreover, the requirements for inclusion of CREAM in other gLite 3.1 middleware included support for the classad-based Job Description Language (JDL), for compatibility with the rest of the gLite 3.1 infrastructure.

One of the goals of the OMII-Europe project was to reengineer some of the existing job execution services to include support for the BES and JSDL specifications. As far as CREAM is concerned, we decided to add BES support on the top of the existing CREAM software, as to keep the code for the legacy service effectively separated with the BES/JSDL code (see Illustration 1).

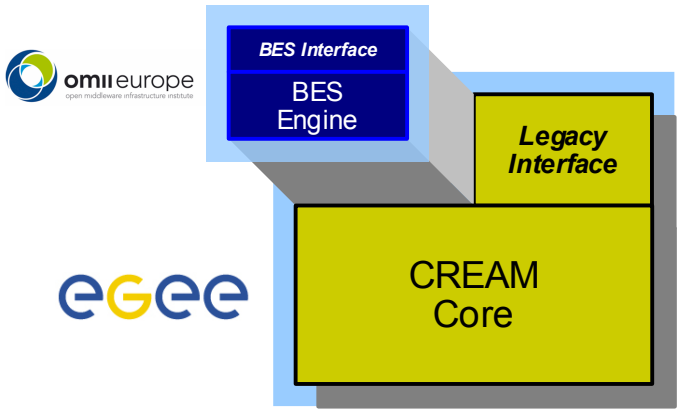



*Illustration 1: CREAM-BES structure*

Illustration 1 depicts the structure of the CREAM-BES server. As can be seen, CREAM-BES is made of two distinct components:

1. the legacy CREAM server; this is exactly the same software used in gLite 3.1, which needs to be installed according to the standard CREAM installation procedure (see the CREAM-BES installatino procedure described at http://grid.pd.infn.it/omii/cream-bes). The legacy CREAM (the part in yellow) is being developed by the EGEE-2 project for the gLite 3.1 middleware.

2. the BES interface with the corresponding implementation for CREAM. Implementation of the BES interface for CREAM (part in blue in the picture) is being developed by the OMII-Europe project. The implementation of the BES interface uses the CREAM core; note that the BES interface can coexist with the legacy CREAM one (i.e., both interfaces can be used at the same time).

There are some advantages in keeping the core CREAM physically separated from the BES interface and implementation. Specifically:

1. The BES/JSDL specifications evolve at a different rate than the development of the legacy CREAM service which is being included in gLite 3.1. It is thus necessary to decouple the development of the BES/JSDL interface from that of the CREAM core.
2. The CREAM-BES/JSDL interface does not require any modification on the CREAM core, so that the new interface does not introduce additional bugs, and does not interfere with the testing of CREAM as done by the EGEE-2 certification team.
3. Development of the BES/JSDL interface can make use of the latest CREAM version available. We use the ETICS [reference] system for building CREAM and CREAM-BES. The CREAM-BES ETICS component includes the latest version of the legacy CREAM as an external component. This means that every build of CREAM-BES uses the latest version of the CREAM core, so that CREAM-BES automatically gets all the bug fixes and improvements from the current CREAM release.

## 3.2 Building and Installing CREAM-BES

At the time of writing this document, the ETICS build and test facility is being upgraded to version 2.0, so instructions to build CREAM-BES are likely to change. Detailed and up-to-date building instruction of CREAM-BES can always be found in its wiki page: http://grid.pd.infn.it/omii/cream-bes/.

## 3.3 JSDL vs CREAM JDL

The job description notation used by the legacy interface of the CREAM server is the Job Description Language (JDL), which is a textual notation based on Condor classads (it should be noted that Condor classads also have an XML rendering, but this is not used by CREAM). The CREAM JDL contains

- 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"; 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. Details of the CREAM JDL have already been discussed in the Milestone document MJRA1.7 [MJRA1.7].

## 3.4 Implementation of JSDL in CREAM-BES: Core elements

As recalled above, the legacy CREAM uses the JDL as the job description notation [JDL]. However, the CREAM core is not directly dependent on the JDL notation: in fact, the internal job representation used in the CREAM core is an `org.glite.ce.creamapij.Job` object, which is completely independent from the JDL representation. Basically, the `org.glite.ce.creamapij.Job` class contains a set of members which are basic types describing the list of input files to stage in, the list of output files to stage out, the executable which should be run by the job, the Virtual Organization (VO) the user running the job belongs to, and other information.

The JobRegister method of the legacy CREAM interface is the only method of CREAM which accepts a JDL as a parameter. This method, among other things, instantiates the appropriate `org.glite.ce.creamapij,Job` setting the value of its members to correct values as desumed by the parsing of the JDL document. The newly created `org.glite.ce.creamapij.Job` object is then passed along to classes of the CREAM core.

The same mechanism is used in CREAM-BES: the CreateActivity BES method receives as parameter a JSDL. CREAM-BES extract the relevant informations from the JSDL, and instantiates a `org.glite.ce.creamapij.Job` object to contain those informations. The Job object is then passed down to the CREAM core. Note that CREAM-BES does *not* convert the JSDL into a JDL; it merely fills in the needed information into a `org.glite.ce.creamapij.Job`object.

The following table shows the mapping between some members of the `org.glite.ce.creamapij.Job` class and JSDL elements. The CREAM-BES interface simply fills the member whose name is on the first column of the table with the value extracted from the JSDL elements whose name is on the third column.

| `org.glite.ce.creamapij.Job` member name | Meaning | Corresponding JSDL element |
|---|---|---|
| `private String arguments` | Arguments passed to the executable | `//jsdl-hpcpa:Argument` |
| `private Hashtable environment` | Values for environment variable | `//jsdl-hpcpa:Environment` |

| | | |
|---|---|---|
| `private String standardInput` | Name of the file where the standard input for the executable is fetched (this file is usually part of the staged in data) | `//jsdl-hpcpa:Input` |
| `private String standardOutput` | Name of the file where the standard output produced by the executable will be stored (this file is usually part of the staged out data) | `//jsdl-hpcpa:Output` |
| `private String standardError` | Name of the file where the standard error produced by the executable will be stored (this file is usually part of the staged out data) | `//jsdl-hpcpa:Error` |
| `private String executable` | Name of the executable which is run by this job | `//jsdl-hpcpa:Executable` |
| `private String VO` | Name of the Virtual Organization (VO) under which the user executes the job | `//jsdl:JobProject` |
| `private String[] InputFiles` | List of file names representing the input files used by the executable | `//jsdl:DataStaging/jsdl:Source/jsdl:URI` |
| `private String[] OutputFiles` | List of names representing the output files produced by the executable | `//jsdl:DataStaging/jsdl:Target/jsdl:URI` |
| `private String jdl` | The string describing the whole job | `//jsdl:JobDescription` |

Note that the org.glite.ce.creamapij.Job.jdl element is used by the legacy CREAM to hold the whole classad describing the job which the user submitted. However, this element is never actually used in the CREAM core, so basically can be set to an arbitrary String value. For this reason, CREAM-BES inserts here the XML representation of the JSDL which has been submitted, so that it is possible to retrieve back the JSDL in order to build the response the the `GetActivityDocuments` BES operation.

## 3.5 Implementation of JSDL in CREAM-BES: Resources Element

The Resources JSDL element describes the resource requirements of the job. CREAM-BES is able to check whether the requirements can be satisfied; to do so, CREAM-BES checks the capabilities of the attached batch queues. Such capabilities can be queries using the gLite infoprovider scripts, or by querying the OpenPegasus-based infoprovider. Unfortunately, not the infoprovider scripts do not report all the available capabilities of each batch queue; thus, there are some requirements which CREAM-BES is unable to satisfy, because it cannot verify the relevant parameter. For example, the

`IndividualNetworkBandwidth` JSDL element is used to specify the bandwidth requirement of each individual resource in bits per second. However, none of the currently available information provider report such information, thus CREAM-BES is unable to enforce that requirement.

The BES `CreateActivity` operation is used to submit a new job to the BES service. If the requirements for the job cannot be satisfied, the BES service raises an `UnsupportedFeatureFault` fault; the fault message contains the resource requirements which could not be satisfied. Note that CREAM-BES raises an UnsupportedFeatureFault if one of the following conditions hold:

- The JSDL contains a requirement which CREAM-BES knows about, and is unable to satisfy. For example, a JSDL may request that the job is to be executed on at least 10 CPUs, but less than that number are actually available.
- The JSDL contains a requirement which CREAM-BES does not know about. For example, a JSDL may request that the job runs on a resource with `IndividualNetworkBandwidth`not less than *x*. However, CREAM-BES does not know any information about the network bandwidth of each underlying resource, so that it cannot really guarantee that this requirement is met.

The following table describes the status of CREAM-BES support for the JSDL Requirements elements

| JSDL Element | Yes | No | Comments |
|---|---|---|---|
| `JobIdentification` | X | | This attribute is not used by CREAM-BES |
| `JobName` | X | | This attribute is not used by CREAM-BES |
| `JobAnnotation` | X | | This attribute is not used by CREAM-BES |
| `JobProject` | X | | If defined in the JSDL, CREAM-BES will use it as the Virtual Organization (VO) name for the job |
| `Application` | X | | This attribute is not used by CREAM-BES |
| `ApplicationName` | X | | This attribute is not used by CREAM-BES |
| `ApplicationVersion` | X | | This attribute is not used by CREAM-BES |
| `Resources` | X | | |
| `CandidateHosts` | | X | CREAM-BES is unable to dispatch a job on a specific execution host ;it can only dispatch the job to a specific batch queue |
| `HostName` | | X | |
| `FileSystem` | | X | |
| `MountPoint` | | X | |
| `MountSource` | | X | |
| `DiskSpace` | | X | |
| `FileSystemType` | | X | |

| JSDL Element | Yes | No | Comments |
|---|---|---|---|
| ExclusiveExecution | | X | Partially supported. CREAM-BES does not support exclusive execution of a process on a worker node. Thus, if jsdl:ExclusiveExecution is requested to be true, CREAM-BES will raise an UnsupportedFeatureFault fault |
| OperatingSystem | X | | |
| OperatingSystemType | X | | |
| OperatingSystemName | X | | |
| OperatingSystemVersion | X | | |
| CPUArchitecture | X | | |
| CPUArchitectureName | X | | |
| IndividualCPUSpeed | X | | |
| IndividualCPUTime | X | | |
| IndividualCPUCount | X | | |
| IndividualNetworkBandwidth | | X | |
| IndividualPhysicalMemory | X | | |
| IndividualVirtualMemory | X | | |
| IndividualDiskSpace | | X | |
| TotalCPUTime | | X | |
| TotalCPUCount | X | | |
| TotalPhysicalMemory | | X | |
| TotalVirtualMemory | | X | |
| TotalDiskSpace | | X | |
| TotalResourceCount | | X | |
| DataStaging | X | | |
| FileName | X | | |
| FilesystemName | | X | |
| CreationFlag | | X | |
| DeleteOnTermination | | X | |
| Source | X | | |
| Target | X | | |
| POSIXApplication | | X | |

| JSDL Element | Yes | No | Comments |
|---|---|---|---|
| Executable | | X | |
| Argument | | X | |
| Input | | X | |
| Output | | X | |
| Error | | X | |
| WorkingDirectory | | X | |
| Environment | | X | |
| WallTimeLimit | | X | |
| FileSizeLimit | | X | |
| CoreDumpLimit | | X | |
| DataSegmentLimit | | X | |
| LockedMemoryLimit | | X | |
| MemoryLimit | | X | |
| OpenDescriptorsLimit | | X | |
| PipeSizeLimit | | X | |
| StackSizeLimit | | X | |
| CPUTimeLimit | | X | |
| ProcessCountLimit | | X | |
| VirtualMemoryLimit | | X | |
| CPUTimeLimit | | X | |
| ProcessCountLimit | | X | |
| VirtualMemoryLimit | | X | |
| ThreadCountLimit | | X | |
| UserName | | X | |
| GroupName | | X | |
| HPCProfileApplication | X | | |
| Executable | X | | |
| Argument | X | | |
| Input | X | | |
| Output | X | | |

| JSDL Element | Yes | No | Comments |
|---|---|---|---|
| `Error` | X | | |
| `WorkingDirectory` | | X | CREAM-BES always executes jobs in a temporary directory chosen by the system |
| `Environment` | X | | |
| `UserName` | | X | CREAM-BES always executes jobs choosing an appropriate user ID based on local mapping |

It should be noted that the lack of support for some resource requirements in CREAM-BES is not related to any intrinsic limitation of CREAM-BES itself; in fact it is related to the information provider used by CREAM-BES to gather resource information. However, since CREAM-BES allows information providers to be "plugged-in" using a standard interface, it is foreseen that when more advanced providers will be available, CREAM-BES will support a wider range of resource requirements.

## 3.6 Implementation of JSDL in CREAM-BES: Data Staging

The JSDL specification defines a `DataStaging` element, which is a complex XML type describing which files must be moved to and from the execution host. Files are staged in (moved to the execution host) before the job starts execution, and are staged out (moved from the execution host to a suitable location) after the job terminates.

In the legacy JDL there is the concept of *input* and *output* sandboxes. The input sandbox is the list of files which need to be staged in, while the output sandbox is the list of files which need to be staged out. CREAM-BES builds these lists by parsing the content of the `DataStaging` JSDL element, as follows:

- The content of all `//jsdl:DataStaging/jsdl:FileName` elements such that there exists a `jsdl:Source` sibling, is inserted into the input sandbox list;
- The content of all `//jsdl:DataStaging/jsdl:FileName` elements such that there exists a `jsdl:Target` sibling, is inserted into the output sandbox list.

Support for the `DataStaging` element in CREAM-BES has the following limitations:

- The `FileSystemName` sub-element is not supported and is currently ignored; it should be noted that a BES implementation should NOT silently ignore any element it does not support or understand. The currect behaviour here should be to raise an appropriate fault if the `FileSystemName` sub-element is used. This is true also for the other JSDL elements mentioned below.
- The `CreationFlag` sub-element is not supported and is currently ignored;
- The `DeleteOnTermination` sub-element is not supported and is currently ignored.
- The only URI schema which is currently supported is plain ftp; if username/password authentication is used, username and password can be embedded into the URI according to the standard [Reference] (e.g.: `ftp://user:passwd@foo.bar.com/file.dat`). While this approach

is dangerous from a security point of view, it is the only approach which can safely be implemented at the moment in an interoperable manner, due to the lack of a standard credential delegation mechanisms to be implemented in BES implementations.

## 3.7 Examples

In this section we give some examples of valid JSDL documents which are processed by CREAM-BES.

### 3.7.1 A simple "sleep 30" application

Let us consider the following JSDL:

```
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:JobDefinition xmlns="http://www.example.org/"
                    xmlns:jsdl="http://schema.ggf.org/jsdl/2005/11/jsdl"
                    xmlns:jsdl-posix="http://schemas.ggf.org/jsdl/2005/11/jsdl-posix"
                    xmlns:jsdl-hpcpa="http://schemas.ggf.org/jsdl/2006/07/jsdl-hpcpa"
                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <jsdl:JobDescription>

    <!-- JobIdentification block -->
    <jsdl:JobIdentification>
      <jsdl:JobName>Simple Test Job</jsdl:JobName>
      <jsdl:Description>A simple job which just sleeps for 20 seconds</jsdl:Description>
    </jsdl:JobIdentification>

    <!-- Application block -->
    <jsdl:Application>
      <jsdl:ApplicationName>sleep</jsdl:ApplicationName>
      <jsdl-hpcpa:HPCProfileApplication>
        <jsdl-hpcpa:Executable>/bin/sleep</jsdl-hpcpa:Executable>
        <jsdl-hpcpa:Argument>20</jsdl-hpcpa:Argument>
      </jsdl-hpcpa:HPCProfileApplication>
    </jsdl:Application>

    <!-- Resources block -->
    <jsdl:Resources>
      <jsdl:TotalCPUCount>
        <jsdl:LowerBoundedRange>1.0</jsdl:LowerBoundedRange>
      </jsdl:TotalCPUCount>
    </jsdl:Resources>

  </jsdl:JobDescription>
</jsdl:JobDefinition>
```

This JSDL defines a simple application (the values of the XML elements in `/jsdl:JobDefinition/jsdl:JobDescription/jsdl:JobIdentification/*` are ignored by CREAM-BES). The `//jsdl:Application/jsdl:ApplicationName` element denotes a string which represents a human-readable name of the application. It may contain whatever value, as it is used for informative purposes only. The `//jsdl-hpcpa:Executable`element contains the complete pathname of the executable command which this job runs, while the `//jsdl-hpcpa:Argument`elements contain any parameter which should be passed via command-line to the executable. Note that there may be multiple jsdl-hpcpa:Argument elements, so that it is possible to pass multiple command-line options to the executable.

The JSDL resources block defines a simple requirement for the job, namely, that the job requires *at least* one CPU to be executed. This is expressed using the `jsdl:TotalCPUCount` element, which according to the JSDL specification, describes the total number of CPUs required by this job submission the `//jsdl:TotalCPUCount/jsdl:LowerBoundedRange` element states that this requirement specifies a lower bound.

### 3.7.2 An application using data staging

Let us consider the following JSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:JobDefinition xmlns="http://www.example.org/"
                    xmlns:jsdl="http://schema.ggf.org/jsdl/2005/11/jsdl"
                    xmlns:jsdl-posix="http://schemas.ggf.org/jsdl/2005/11/jsdl-posix"
                    xmlns:jsdl-hpcpa="http://schemas.ggf.org/jsdl/2006/07/jsdl-hpcpa"
                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <jsdl:JobDescription>

    <!-- JobIdentification block -->
    <jsdl:JobIdentification>
      <jsdl:JobName>Image conversion job</jsdl:JobName>
      <jsdl:Description>
        This job converts an image using ImageMagick. It is used to
        check whether data staging actually works.
      </jsdl:Description>
    </jsdl:JobIdentification>

    <!-- Application block -->
    <jsdl:Application>
      <jsdl:ApplicationName>convert</jsdl:ApplicationName>
      <jsdl-hpcpa:HPCProfileApplication>
        <jsdl-hpcpa:Executable>/usr/bin/convert</jsdl-hpcpa:Executable>
        <jsdl-hpcpa:Argument>-format</jsdl-hpcpa:Argument>
        <jsdl-hpcpa:Argument>png</jsdl-hpcpa:Argument>
        <jsdl-hpcpa:Argument>-scale</jsdl-hpcpa:Argument>
        <jsdl-hpcpa:Argument>30&#37;</jsdl-hpcpa:Argument>
        <jsdl-hpcpa:Argument>-grayscale</jsdl-hpcpa:Argument>
        <jsdl-hpcpa:Argument>fish.jpg</jsdl-hpcpa:Argument>
        <jsdl-hpcpa:Argument>fish_small.png</jsdl-hpcpa:Argument>
      </jsdl-hpcpa:HPCProfileApplication>
    </jsdl:Application>

    <!-- Resources block -->

    <!--
         We need (at least) one CPU to execute the job.  This
         requirement is not really meaningful, it is here just to show
         how requirements work in JSDL
    -->
    <jsdl:Resources>
      <jsdl:TotalCPUCount>
        <jsdl:LowerBoundedRange>1.0</jsdl:LowerBoundedRange>
      </jsdl:TotalCPUCount>
    </jsdl:Resources>

    <!--
         Data staging elements
    -->
    <jsdl:DataStaging>
      <jsdl:FileName>fish.jpg</jsdl:FileName>
      <jsdl:Source>
        <jsdl:URI>ftp://test:testpwd@host.mydomain.org/home/test/fish.jpg</jsdl:URI>
      </jsdl:Source>
      <jsdl:CreationFlag>overwrite</jsdl:CreationFlag>
      <jsdl:DeleteOnTermination>true</jsdl:DeleteOnTermination>
    </jsdl:DataStaging>
    <jsdl:DataStaging>
      <jsdl:FileName>fish_small.png</jsdl:FileName>
      <jsdl:Target>
        <jsdl:URI>ftp://test:testpwd@host.mydomain.org/home/test/fish_small.png</jsdl:URI>
      </jsdl:Target>
      <jsdl:CreationFlag>overwrite</jsdl:CreationFlag>
      <jsdl:DeleteOnTermination>true</jsdl:DeleteOnTermination>
    </jsdl:DataStaging>
  </jsdl:JobDescription>
</jsdl:JobDefinition>
```

This JSDL invokes the following command (see the `//jsdl-hpcpa:HPCProfileApplication` block):

```
/usr/bin/convert -format png -scale 30% -grayscale fish.jpg fish_small.jpg
```

Note the use of the % entity to denote the percent sign (%); this is necessary as the percent sign has a special meaning in XML documents. Note also that each individual piece of the command line arguments ('-format', 'png' and so on) is defined in a single `//jsdl-hpcpa:HPCProfileApplication/Argument` element.
The input and output files fish.jpg and fish_small.jpg are staged from/to a remote location using the FTP protocol with username/password authentication. In particular, the following block:

```
<jsdl:DataStaging>
  <jsdl:FileName>fish.jpg</jsdl:FileName>
  <jsdl:Source>
    <jsdl:URI>ftp://test:testpwd@host.mydomain.org/home/test/fish.jpg</jsdl:URI>
  </jsdl:Source>
  <jsdl:CreationFlag>overwrite</jsdl:CreationFlag>
  <jsdl:DeleteOnTermination>true</jsdl:DeleteOnTermination>
</jsdl:DataStaging>
```

requires the CREAM-BES service to fetch (stage in) an external file from location `host.mydomain.org/home/test/fish.jpg` using username “test” and password “testpwd” to authenticate with the remote FTP server. The name of the (local) copy of this file is specified using the `//jsdl:DataStaging/jsdl:FileName` element, and in this case is “fish.jpg”. Of course it is possible to stage in a file giving it a different local name than the original remote name.

Similarly, the block

```
<jsdl:DataStaging>
  <jsdl:FileName>fish_small.png</jsdl:FileName>
  <jsdl:Target>
    <jsdl:URI>ftp://test:testpwd@host.mydomain.org/home/test/fish_small.png</jsdl:URI>
  </jsdl:Target>
  <jsdl:CreationFlag>overwrite</jsdl:CreationFlag>
  <jsdl:DeleteOnTermination>true</jsdl:DeleteOnTermination>
</jsdl:DataStaging>
```

requests that the local file with name “fish_small.png” (which must be produced after the application terminates) has to be staged out to the remote location `host.mydomain.org/home/test/fish_small.png`, using again username “test” and password “testpwd” to authenticate with the remote FTP server.

# 4 Implementation of JSDL in UNICORE6

In addition to the work described here there is also work currently underway to develop an OGSA-BES Plugin for the UNICORE Rich Client Platform based on Eclipse.

## *4.1 BES and UNICORE6*

The OGSA-Basic Execution Services (BES) specification describes a Web services interface for the creation, monitoring and control of computational jobs. In BES computational jobs are called "activities". BES activities can be described using JSDL. This section provides an overview of the integration of OGSA-BES interfaces into the UNICORE 6 Grid middleware [UNICORE], in addition the ByteIO interfaces for data access are also described.

A study of the the OGSA-BES interface reveals that it is similar to the Target System Service and Job Management Service of the UAS of UNICORE 6. While the UAS does have WS-RF compliant message exchanges and submit operation takes a JSDL document as parameter, the syntax of the UAS operations is proprietary. The OGSA-BES interfaces offer UNICORE 6 a standardized syntax for Web service operations dealing with job control and management which can be internally implemented on top of the same XNJS as the UAS. The Web service layer is well isolated from the lower-level execution engine. All OGSA-BES invocations are transmitted to the server-side within the SOAP body, except the pieces of information within the SOAP header that are used to address specific job WS-Resource instances and also the security tokens. As shown in Illustration 2, the integrated OGSA-BES UNICORE 6 interface has three services: Activity, Factory and Management services.

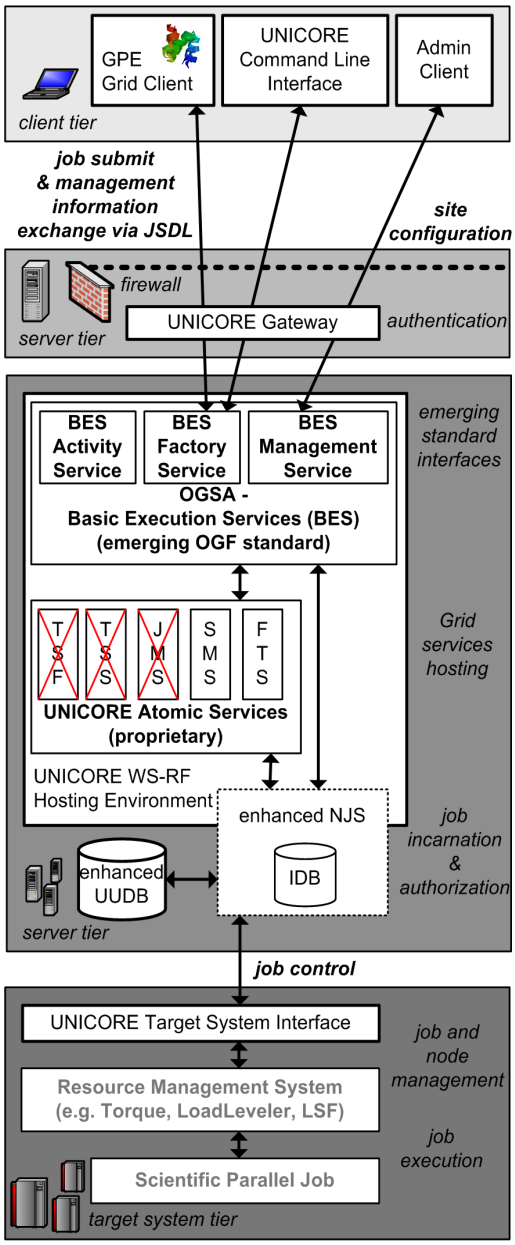



*Illustration 2: UNICORE 6 with a deployed OGSA–BES services. In OMII–Europe the proprietary UNICORE Atomic Services are partly replaced by OGSA–BES services.*

The BES Factory Service is responsible for the creation and control of a set of activities (described using JSDL). In the UNICORE 6 implementation, this Web service interface uses the newly developed and enhanced NJS as execution backend that provide significant performance and scalability improvements over UNICORE 5. The CreateActivity(JSDL) operation of the BES-Factory Service leads to the creation of a job resource within the NJS that represents the computational job described by the JSDL. At the Web service level, this resource can be controlled and monitored by using the BES Activity Service operations, e.g. by using the `Terminate()` or

`GetStatus()` operations. In addition, the OGSA-BES implementation for UNICORE includes a BES Management Service. This service improves the functionality of UNICORE in terms of remote administration. An administration client, for instance, can be used to access this service and control whether new jobs can be submitted to UNICORE or not. To support this, the BES operations `StopAcceptingNewActivities()` and `StartAcceptingNewActivities()` are supported by UNICORE 6.

UNICORE 6 is also compliant with the HPC-Profile. This profile specifies the usage of the OGSA-BES interface in conjunction with certain extensions to JSDL. UNICORE 6 implements HPC Application as an extension to JSDL that is used to describe an executable running as an operating system process. Basically it shares much in common with the JSDL POSIXApplication, but removes some of the features that present barriers to interoperability by using the XML element jsdl-hpcpa:Executable and other similar elements.

### 4.1.1 Data Access and File Transfer

During the implementation of the OGSA-BES interface (which includes JSDL) it became clear that BES covers less of the extent of job execution functionality than the UAS. In most job management use cases, data needs to be staged-in before execution and after the job execution the results need to be staged-out to a dedicated (permanent) location. Therefore, file transfers and storage management have a close relationship to every job management system on computational Grids or e-Science infrastructures. The OGSA-BES specification has left this concern to implementers. But both file transfer and storage management are needed if complete JSDL descriptions are to be supported. Therefore, the OGSA-BES implementation within UNICORE 6 also provides a file transfer service (FTS). The FTS provides Streamable ByteIO (SByteIO) and Random ByteIO (RByteIO) data access mechanisms to support job submissions that rely on staged data. These specifications are standardized by the OGF and thus lay the foundation for data staging capabilities in interoperability scenarios and cross-Grid use cases.

Local and remote transfer mechanisms are supported. With local transfer an end-user uploads job data to the target machine prior to job submission. This data moved by the execution engine for further processing and job submission at the TSI. Transfers via remote storage use a dedicated machine for managing file transfer and data storage. The end-user uploads data to a third arty and as part of the job description specifies this location in the JSDL.

To sum up, the data transfer and access mechanisms are realized using the FTS and Storage Management Service (SMS) (see Illustration 2). FTS is the actual service exposing the ByteIO interface specification and it is responsible for staging data in and out during job submission and execution. The SMS is thus a factory that creates different FTS instances for different data transfers. The SMS is also used to manage the storage of data on all the tiers such as client tier, server tier, or even on a remote data server.

In summary, UNICORE 6 provides a standardized job submission interface (OGSA-BES) that works with standardized job descriptions (JSDL) that may use standardized data access mechanisms (RByteIO/SByteIO) for staging data.

## 4.2 Building and Installing UNICORE6-BES

As of version 6.X the BES interface is included as part of the standard UNICORE 6 release and can be installed using the instructions contained in any release package. the latest UNICORE 6 release is installed on the OMII Europe evaluation sites.

## 4.3 Implementation of JSDL in UNICORE 6

UNICORE 6 uses JSDL as its native job description language. The following table indicates the support in UNICORE 6 for the JSDL elements..

| JSDL Element | Yes | No | Comments |
|---|---|---|---|
| JobIdentification | X | | |
| JobName | X | | |
| JobAnnotation | | X | |
| JobProject | | X | |
| Application | X | | |
| ApplicationName | X | | |
| ApplicationVersion | X | | |
| Resources | X | | |
| CandidateHosts | | X | |
| HostName | | X | |
| FileSystem | | X | |
| MountPoint | | X | |
| MountSource | | X | |
| DiskSpace | | X | |
| FileSystemType | | X | |
| ExclusiveExecution | | X | |
| OperatingSystem | | X | |
| OperatingSystemType | | X | |
| OperatingSystemName | | X | |
| OperatingSystemVersion | | X | |
| CPUArchitecture | | X | |
| CPUArchitectureName | | X | |
| IndividualCPUSpeed | | X | |

| JSDL Element | Yes | No | Comments |
|---|---|---|---|
| IndividualCPUTime | X | | |
| IndividualCPUCount | X | | |
| IndividualNetworkBandwidth | | X | |
| IndividualPhysicalMemory | X | | |
| IndividualVirtualMemory | | X | |
| IndividualDiskSpace | | X | |
| TotalCPUTime | | X | |
| TotalCPUCount | | X | |
| TotalPhysicalMemory | | X | |
| TotalVirtualMemory | | X | |
| TotalDiskSpace | | X | |
| TotalResourceCount | X | | |
| DataStaging | X | | |
| FileName | X | | |
| FilesystemName | X | | |
| CreationFlag | | X | |
| DeleteOnTermination | | X | |
| Source | X | | |
| Target | X | | |
| POSIXApplication | X | | |
| Executable | X | | |
| Argument | X | | array |
| Input | X | | |
| Output | X | | |
| Error | X | | |
| WorkingDirectory | | X | UNICORE always executes jobs in a temporary directory chosen by the system |
| Environment | X | | array |
| WallTimeLimit | | X | |
| FileSizeLimit | | X | |
| CoreDumpLimit | | X | |

| JSDL Element | Yes | No | Comments |
|---|---|---|---|
| DataSegmentLimit | | X | |
| LockedMemoryLimit | | X | |
| MemoryLimit | | X | |
| OpenDescriptorsLimit | | X | |
| PipeSizeLimit | | X | |
| StackSizeLimit | | X | |
| CPUTimeLimit | | X | |
| ProcessCountLimit | | X | |
| VirtualMemoryLimit | | X | |
| CPUTimeLimit | | X | |
| ProcessCountLimit | | X | |
| VirtualMemoryLimit | | X | |
| ThreadCountLimit | | X | |
| UserName | X | | |
| GroupName | | X | |
| HPCProfileApplication | X | | Used like POSIXApplication |
| Executable | X | | |
| Argument | X | | |
| Input | X | | |
| Output | X | | |
| Error | X | | |
| WorkingDirectory | | X | UNICORE always executes jobs in a temporary directory chosen by the system |
| Environment | X | | |
| UserName | X | | |

The JSDL specification as well as the HPCProfileApplication extension does not cover all necessary details for HPC-based job submissions (e.g. number of processes per host) and therefore the UNICORE 6 community is considering proprietary JSDL extensions for these.

# 5 Conclusions

In this document we have described the implementation details of JSDL into CREAM-BES and UNICORE6. CREAM-BES is the BES/JSDL-enabled extension of the legacy CREAM execution service which is being developed for the gLite 3.1 middleware.

The core JSDL elements matched quite well with the existing functionalities provided by the native job description notations used by UNICORE and CREAM; thus, for simple jobs, JSDL provides most of the features which were available in the legacy notations. However, there are also features provided by JSDL which cannot be easily implemented without rewriting significant amounts of existing code. In particular, JSDL allows users to specify several attributes for the execution environment on which the job will be executed (e.g., filesystem layout, local user ID, network bandwidth requirements etc.) which cannot be directly controlled by UNICORE or CREAM execution services. Such features are thus currently not supported in UNICORE6 and CREAM-BES.

On the other hand, we observed that many existing execution services (including UNICORE and CREAM) provide additional functionalities which are not covered by the JSDL specification; the most notable one is the ability to describe and execute job collections with dependencies described as DAGs (Directed Acyclic Graphs). Since it is quite natural to leverage the Grid power to execute such “complex” kind of jobs, we are currently investigating solutions to overcome these limitations of JSDL. Most likely, these limitations will be addressed by developing appropriate JSDL extensions (using the standard JSDL extension mechanisms).

# 6 References

[JSDL-spec] A. Savva (editor), *Job Submission Description Language (JSDL) Specification, Version 1.0*, GGF November 2005, available at http://www.gridforum.org/documents/GFD.56.pdf

[BES-spec] I. Foster et al., OGSA Basic Execution Service, Version 1.0, August 2007. available at http://www.ogf.org/documents/GFD108.pdf

[UNICORE] A. Streit, D. Erwin, T. Lippert, D. Mallmann, R. Menday, M. Rambadt, M. Riedel, M. Romberg, B. Schuller, and P. Wieder. UNICORE - From Project Results to Production Grids. In L. Grandinetti, editor, Grid Computing: The New Frontiers of High Performance Processing, Advances in Parallel Computing 14, pages 357–376. Elsevier

[CREAM] C. Aiftimiei et al., Job Submission and Management Through Web Services: the Experience with the CREAM Service, to appear in proc. Intl. Conf. Computing in High Energy and Nuclear Physics (CHEP'07).

[JDL] M. Sgaravatto, *CREAM Job Description Language Attributes Specification for the EGEE Middleware*, document identifier EGEE-JRA1-TEC-592336, available online at https://edms.cern.ch/document/502336

[MJRA1.7] Defitions of JSDL extensions, Milestone Document MJRA1.7, OMII-Europe project, 2006