omii europe
open middleware infrastructure institute

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**

Milestone M:JRA1.17 Evaluation of OGSA-BES with respect to its adoption in the middleware of the OMII – Europe partners

Due date of milestone: Jan 2007
Actual submission date: April 2007

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

Organisation name of lead contractor for this deliverable: INFN

Revision [1.0]

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)		
Dissemination Level		
PU	Public	**x**
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 Control Sheet

Document	**Title**: Evaluation of BES with respect to its adoption in the middleware of the OMII – Europe partners	
	ID: D:JRA1.17	
	Version: 1.0	**Status**: Final
	Available at: http://grid.pd.infn.it/omii/milestones:jra117	
	Software Tool: Microsoft Word / OpenOffice	
	File(s): 2007-04-10_Job_Submit_Milestone_JRA1_17	
Authorship	**Written by:**	Moreno Marzolla (INFN), Morris Riedel (FZJ)
	Contributors:	Shahbaz Memon (FZJ), Shiraz Memon (FZJ), Vivian Li (FLE),
	Reviewed by:	TBD
	Approved by:	TBD

Document Status Sheet

Version	Date	Status	Comments
0.1	09 March 2007	Draft	Initial version by Moreno (INFN)
0.2	28 March 2007	Draft	Morris/Shiraz/Shahbaz/Vivian adds UNICORE OGSA-BES and other evaluations
1.0	10 April 2007	Final	Morris/Moreno revise document with respect to current implementation experience

Executive Summary

This document examines the current status of the Basic Execution Service (BES) specification. We describe the aim and scope of BES and analyze the specification in order to consider its adoption in the Grid infrastructures of OMII-Europe partners. The BES specification, at the time this document is being written, is not finalized yet. This document refers to the most recent available version, which is currently draft version 33 [OGSA-BES].

This document is organized as follows: first we describe the job management interfaces used in gLite, UNICORE and Globus. We then describe the current BES specification; we finally give some remarks with respect to the adoption of BES in OMII-Europe infrastructures.

Table of Contents

1. Introduction

The OGSA - Basic Execution Service (BES) specification describes a Web Service interface for creation, monitoring and control of computational jobs. The meaning of "computational job" is quite broad: it could be a Host Operating System process, or even Web Services of parallel programs **[OGSA-BES]**. In the BES terminology, such a computational job is called *activity*. Activities can be described using the Job Submission Description Language **[JSDL]** notation.

The BES specification defined three different port-types:

BES-Factory: create, monitor and control a set of activities, and monitor BES attributes;

BES-Activity:create, monitor and control individual activities;

BES-Management: control the BES server itself. This port-type is intended to be used by system administrators.

Table 1 contains the namespace prefixes and related specifications in the context of OGSA-BES.

Prefix	Namespace
s11	http://schemas.xmlsoap.org/soap/envelope
xsd	http://www.w3.org/2001/XMLSchema
wsa	http://www.w3.org/2005/03/addressing
wse	http://schemas.xmlsoap.org/ws/2004/08/eventing
jsdl	http://schemas.ggf.org/jsdl/2005/11/jsdl
bes-mgmt	http://schemas.ggf.org/bes/2006/08/bes-management
bes-factory	http://schemas.ggf.org/bes/2006/08/bes-factory
bes-activity	http://schemas.ggf.org/bes/2006/08/bes-activity
bes-ext	http://schemas.ggf.org/bes/2006/08/bes-extensions

Table 1: Namespace prefixes and related specifications

2. The OGSA - BES specification in context

The OGSA - BES specification defines a standard Web service interface for creating, monitoring and controlling computational entities. The specification defines three WS port-types, which are shown in Table 2 with their corresponding operations.

BES-Management Port-type	
StopAcceptingNewActivities	Administrative operation: Requests that the BES service stops accepting new activities
StartAcceptingNewActivities	Administrative operation: Requests that the BES service starts accepting new activities
BES-Factory Port-type	
CreateActivity	Requests the creation of a new activity; in general, this operation performs the submission of a new computational job, which is immediately started
GetActivityStatus	Gets the status of a set of activities
TerminateActivities	Terminates a set of activities which have been previously created with CreateActivity
GetActivityDocuments	Requests the JSDL document for a set of activities
GetFactoryAttributeDocument	Requests the XML document containing the properties of this BES service
BES-Activity Port-type (optional)	
GetStatus	Requests the status of an activity
Terminate	Terminates an activity
GetDocument	Requests the JSDL document representing an activity
GetActivityAttributeDocument	Requests the XML document containing activity properties

Table 2: BES Port-types and operations

The specification requires that all OGSA - BES implementations must support a simple operation for retrieving all attributes in a single document (the GetFactoryAttributeDocument operation). However, the specification allows that a specific BES implementation may support other access mechanisms. In particular, an implementation may compose appropriate port-types—e.g., those defined in the WS-RF/WS-Notification, WS-Transfer/WS-Eventing, or WS-ResourceTransfer families of specifications—with the port-types defined in the BES specification.

The BES-Management port-type is used to control the OGSA - BES service itself. In the current specification, this port-type contains two operations which are used to stop the service from accepting new requests, and restart it respectively. This port-type should normally be used by the system administrators.

The BES-Factory port-type defines operations for creation and manipulation of activities and set of activities. Moreover, it contains an operation (GetFactoryAttributeDocument) for retrieving attribute information about the BES service itself. Such information contains, among others, the human-readable service name, the total number of activities currently active in the service, the EndPoint Reference (EPR) to activities currently active in the service, the number of contained resources accessible by the OGSA - BES and so on. BES uses the Job Submission Description Language (JSDL) **[JSDL]** specification to describe activities. Attributes are uniquely identified using WS-Addressing Endpoint References [WS-ADDR]. The BES CreateActivity operation returns an EPR, which can be used by clients to referto this activity.

During execution, activities traverse a number of states. The basic state model is shown in Illustration 1:

- **Pending**: The service has created the activity, but the latter is not yet running on any computational resource
- **Running**: The activity is executing on some computational resource
- **Finished**: The activity successfully completed execution; this is a terminal state.
- **Terminated**: The activity has been terminated by calling the TerminateActivity OGSA - BES operation; this is a terminal state
- **Failed**: The activity has failed due to some error or failure.

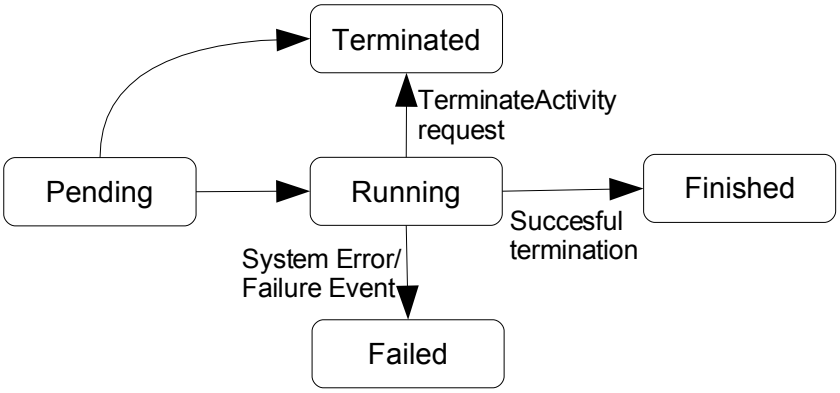

Illustration : BES Basic State Model

The BES specification allows OGSA - BES services to define additional, specialized states as refinement of the basic state model. In particular, the XML representation of the sub-states is performed by inserting the sub-state into the XML element representing each state, as follows:

```
<bes:ActivityStatus state="Running">
        <n00:Staging-In/>
</bes:ActivityStatus>
```

The above XML fragment represents an activity in the "Staging-In" sub-state of the "Running" status. Note that clients which are not aware of the existence of the sub-state may simply ignore the content of the <bes:ActivityStatus> element.

The BES specification already supports some extensions. The GetFactoryAttributesDocument operation returns an XML representation of the status and capabilities of the service. The BESFactory attributes are shown in Table 3.

Name	Type	Number	Data Type	Description
IsAcceptingNewActivities	BES	1	Boolean	True if BES is accepting new activities
CommonName	BES	0 or 1	String	Short human-readable name for BES
LongDescription	BES	0 or 1	String	Longer human-readable description.
TotalNumberOfActivities	BES	1	Integer	Number of activities current active in BES
ActivityReference	BES	≥0	EPR	EPRs to activities currently active in BES
TotalNumberOfContainedResources	BES	1	Integer	Number of contained resources accessible by the BES
ContainedResource	BES	≥0	anyType	Currently expected to be either of type BasicResourceAttributesDocumentType or FactoryResourceAttributesDocumentType.
NamingProfile	BES	≥1	URI	URI's of Naming profiles used by BES
BESExtension	BES	≥0	URI	URI's of supported BES extensions
LocalResourceManagerType	BES	1	URI	Resource's local resource manger type
ResourceName	BR	0 or 1	String	Resource's name
OperatingSystem	BR	0 or 1	String	Resource's operating system type
CPUArchitecture	BR	0 or 1	String	Resource's CPU architecture type
CPUcount	BR	0 or 1	DoubleDoubleI	Resource's CPU count
CPUSpeed	BR	0 or 1	DoubleDoubleI	Resource's CPU speed, in Hertz
PhysicalMemory	BR	0 or 1	DoubleDoubleI	Resource's physical memory size, in bytes
VirtualMemory	BR	0 or 1	DoubleDoubleI	Resource's virtual memory size, in bytes

Table 3: BESFactory Attributes

Note in particular the BESExtension attribute, which indicates what, if any, BES extensions are supported. Valid values include:

http://schemas.ggf.org/bes/2006/08/bes-extensions/IdempotentActivityIDLifetime

http://schemas.ggf.org/bes/2006/08/bes-extensions/SupportsSubscriptions

http://schemas.ggf.org/bes/2006/08/bes-extensions/SupportsLifetimes

Idempotent execution semantics. Idempotent execution refers to the ability of identify (and ignore) duplicate requests. This can be important for operations such as CreateActivity and TerminateActivities, which are not idempotent by definition, as they alter the overall state of the BES service. If idempotent execution is required, then the user is requested to put the following element to uniquely identify an activity to the BES using a client-generated identifier:

```
<bes-ext:IdempotentActivityID>
    wsa:AttributedURI
</bes-ext:IdempotentActivityID>
```

By default, the lifetime of the identifier should be equal to the lifetime of the activity. However, the activity requester may request a specific identifier lifetime with the following element:

```
<bes-ext:IdempotentActivityIDLifetime>
    xsd:dateTime
</bes-ext:IdempotentActivityIDLifetime>
```

Subscription to Notification Events. If a BES service allows clients to subscribe for status change notifications must implement the WS-Eventing or WS-Notification protocols. In order to receive notifications, the client must include one of the following subscription request elements as an extension of ActivityDocument in the input of CreateActivity:

```
<wsnt:Subscribe/> or <wse:Subscribe>
```

Lifetime Management. If a BES service implements the WS-ResourceLifetime operation, then clients can request the initial setting of the termination time resource property of the activity as follows:

```
<bes-ext:InitialTerminationTime>
    xsd:dateTime
</bes-ext:InitialTerminationTime>
```

WS-RF Basic Profile Rendering. BES services can be designed to be compliant with the OGSA WS-RF Basic Profile **[WSRF-BASE]** using composition of BES-specific port-types with other port-types defined in the WS-RF Basic Profile. In particular, implementations willing to be compliant with the OGSA WS-RF Basic Profile must implement, in addition to the BES-Management and BES-Factory port-types, the WS-ResourceProperties, WS-ResourceLifetime, and WS-BaseNotification port-types, in a manner compliant with the OGSA WS-RF Base Profile. Implementers must also ensure that all attributes given in the description of the attributes document appear as exactly named WS-ResourceProperties. That is, they must have the QName {http://schemas.ggf.org/bes/2006/08/bes-management}attr-name where attr-name is the name of the attribute used inside the attributes document types.

3. Job Management Interfaces in Grid Systems

In this section we analyze the the ongoing activity related to OGSA-BES-based job submission interfaces for gLite, Globus and UNICORE.

3.1 gLite

The job submission path for gLite is shown in Illustration 2 (for clarity we omit the *Network Server* component which is no longer used).

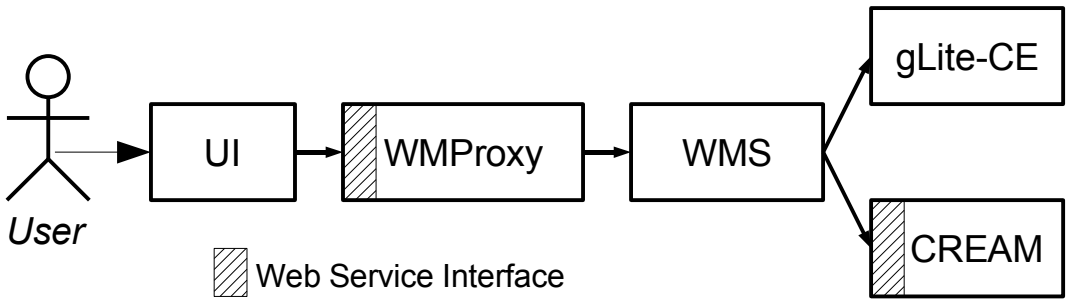

Illustration 2: Job Submission in gLite

Illustration 2: gLite Job submit

The user submits a job on the User Interface, which then transfers the request to the Workload Manager Proxy (WMProxy) [WMPROXY], which interacts with the Workload Management System (WMS). The WMS selects the appropriate Computing Element (CE) for the execution of the job, and finally sends the job to that CE. In gLite, two types of CEs are available: the Condor-based gLite-CE and the Web Service based CREAM (*Computing Resource Execution And Management*) CE **[CREAM]**. The shaded areas in Illustration 2 represent the Web Service interfaces available for job submission. Currently, the WMProxy/WMS (which should actually be considered as a single entity) and CREAM are the two components which expose Web Service interfaces for job submission and management.

The WMProxy interface differs from the CREAM CE one, the reason being that the WMS and the CE have been developed independently and the two components have different responsibilities. The WMS performs the *matchmaking* operation, that is, finds a match between the requirements of each job and the capabilities provided by the different CEs. Computing Elements interact directly with a batch system for execution of single jobs; matchmaking makes no sense on a CE, as once there, a job can only be executed.

Currently, jobs are described according to the Job Description Language (JDL) specification, which was already described in a previous OMII-EU document [OMII-EU-MJRA1.7]. The WMS—and hence, WMProxy—supports simple jobs, collections and Directed Acyclic Graphs (DAGs); DAGs are basically set of jobs with dependencies. The CE, on the other hand, only execute single jobs; support for collections and DAGs in CREAM is planned but not yet available.

To adhere to the SOA model, WMProxy has been designed and implemented as a Simple Object Access Protocol (SOAP) Web service. The interface is described through the Web Service Description Language (WSDL). The WSDL file was written following the Web Services Interoperability Basic Profile (WS-I Basic Profile). This profile defines a set of Web Services specifications that promote interoperability. The WMProxy service runs in an Apache container extended with Fast CGI and Grid Site modules.

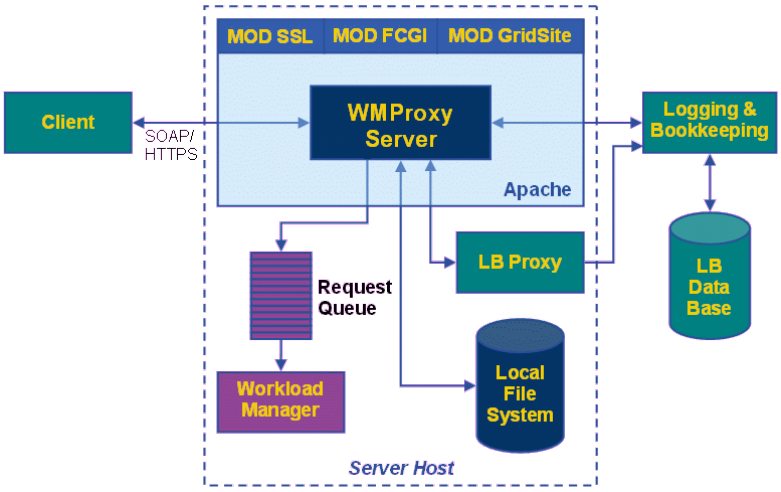

Illustration : WMProxy integration

The integration of the WMProxy within the WMS is shown in Illustration 3. WMProxy provides a core module performing validation, conversion, environment preparation and information logging for each incoming request, before delivering it to the **Workload Manager (WM)**. WM is the core component of the WMS taking the appropriate actions to satisfy incoming requests. Communication between the WMProxy and the WM occurs through a thread-safe, file-system based queue. The LBProxy, which is used as a state storage of active jobs, is the only external service with which the WMProxy interacts. The LBProxy service provides an optimized access to the **Logging and Bookkeeping Service (LB)**. Other relevant information about processed jobs/requests are stored in the local file system in a reserved area managed by the WMS.

A full description of the WSDL interface for the WMProxy service is given in reference [WMP-WSDL].

The CREAM service is also based on a Web Service interface, which provides the operations described in Table 4.

Operation	Description
GetInfo	Returns a set of informations related to the CREAM service itself
JobRegister	Registers a new job for execution. This operation allocates some CREAM internal structures to hold job informations, but the job executable is not started
JobSuspend	Temporarily suspends execution of a running job or list of jobs
JobResume	Resumes execution of a suspended job or list of jobs
JobLease	Renew the lease for a job or list of jobs. Should the lease expire, the CREAM service kills the job and purges tall its local storage
JobList	The the list of the job IDs for all non-purged jobs
JobStart	Starts execution of a job (or list of jobs) which has been previously registered
JobPurge	Kills a job (or list of jobs), and purges all its local storage space
JobInfo	Get detailed information on a job or list of jobs
JobStatus	Get the status information on a job or list of jobs
JobProxyRenew	Renews the user proxy credentials for a given job or list of jobs

Operation	Description
getCEMonURL	Gets the URL of the CEMon service associated with this CREAM service
EnableAcceptJobSubmissions	Enables submissions of new jobs on the CREAM service
DisableAcceptJobSubmissions	Disables submissions of new jobs on the CREAM service
DoesAcceptJobSubmissions	Queries whether the CREAM service is accepting job submissions

Table 4: CREAM operations

3.2 Globus

In this section, we briefly describe the job submission and management interface of the Globus system. 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. 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. The following activities are the main client activities around a WS GRAM job to be a partially ordered sequence [GT4]:

- **Creation of Job**: A 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).
- **Optional Staging Credentials**: Optionally, the client may request staging activities to occur before or after the job.
- **Optional Job Credential**: Optionally, the client may request that a credential be stored into the user account for use by the job process.
- **Optional Credential Refresh**: Optionally, credentials delegated for use with staging, transfer, or job processes may be refreshed using the Delegation service interface.
- **Optional Hold of Cleanup for Streaming Output**: 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.
- **ManagedJob Destruction**: Under nearly all circumstances, ManagedJob resources will be eventually destroyed after job cleanup has completed.

The WS GRAM protocol is centered around the creation of a stateful ManagedJob resource using the ManagedJobFactory *createManagedJob()* operation. A simple batch job may involve nothing more than this initial client creation step, with all other job life cycle steps occurring automatically in the server. A number of optional protocol elements are available for more complex scenarios.

`DelegationFactory::requestSecurityToken`	This (optional) step allows a client to delegate credentials that will be required for correct operation of WS GRAM, RFT, or the user's job process. Such credentials are only used when referenced in the subsequent job request and under the condition that WS GRAM or RFT is configured to make use of the DelegationFactory, respectively.
`Delegation::refresh`	This (optional) step allows a client to update the credentials already established for use with the previous requestSecurityToken step.

`ManagedJobFactory::getResourceProperty` and `getMultipleResourceProperties`	These (optional) steps allow a client to retrieve information about the scheduler and the jobs associated with a particular factory resource before or after job creation. The delegationFactoryEndpoint and stagingDelegationFactoryEndpoint resource properties are two examples of information that may need to be obtained before job creation.
`ManagedJobFactory::createManagedJob`	This required step establishes the stateful ManagedJob resource which implements the job processing described in the input request.
`ManagedJob::release`	This (optional) step allows the ManagedJob to continue through a state in its life cycle where it was previously held or scheduled to be held according to details of the original job request.
`ManagedJob::setTerminationTime`	This (optional) step allows the client to reschedule automatic termination to be different than was originally set during creation of the ManagedJob resource.
`ManagedJob::destroy`	This (optional) step allows the client to explicitly abort a job and destroy the ManagedJob resource in the event that the scheduled automatic termination time is not adequate. If the job has already completed (i.e. is in the Done or Failed state), this will simply destroy the resource associated with the job. If the job has not completed, appropriate steps will be taken to purge the job process from the scheduler and perform clean up operations before setting the job state to Failed.
`ManagedJob::subscribe`	This (optional) step allows a client to subscribe for notifications of status (and particularly life cycle status) of the ManagedJob resource. For responsiveness, it is possible to establish an initial subscription in the createManagedJob() operation without an additional round-trip communication to the newly created job.
`ManagedJob::getResourceProperty` and `getMultipleResourceProperties`	These (optional) steps allow a client to query the status (and particularly life cycle status) of the ManagedJob resource.

The WS-GRAM software architecture is shown in Illustration 4. Note that the most recent version of WS-GRAM (namely, GT 4.2 WS-GRAM, documented at http://www.globus.org/toolkit/docs/development/4.2-drafts/execution/wsgram/user/index.html#wsgram-user-jsdl), includes support for JSDL job descriptions.

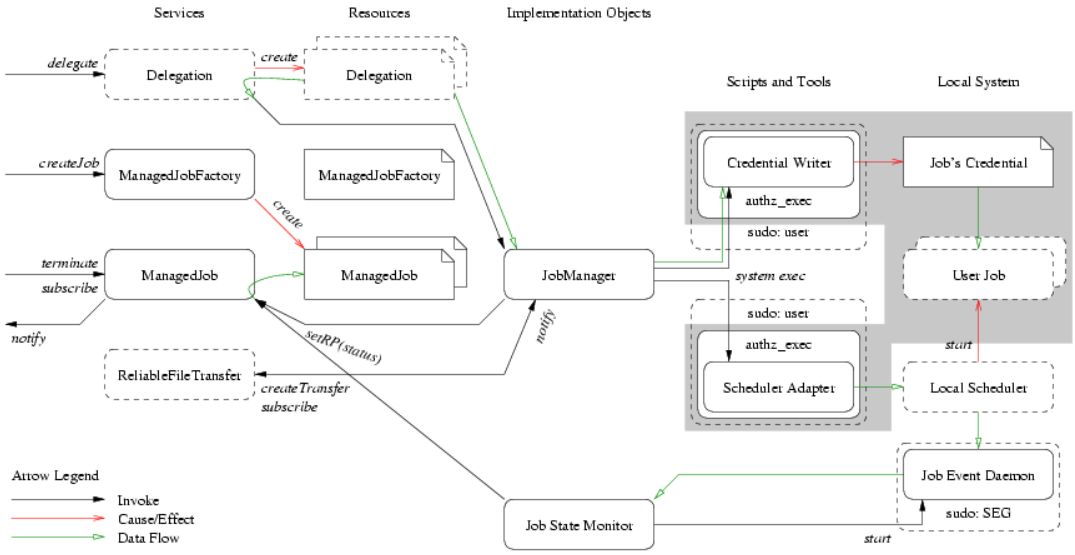

Illustration 4: WS-GRAM Software Architecture

3.3 UNICORE

This section focusses on the augmentation of an OGSA-BES interface for UNICORE, including a mapping of the UNICORE internal state-model and the state-mode standardized by OGSA-BES.

3.3.1 Deployment of OGSA-BES services in UNICORE

In recent years, the UNICORE 5 Grid system evolved to a full-grown and well tested Grid middleware system that is used in daily production at supercomputing centers and other research facilities worldwide. Furthermore, UNICORE serves as a solid basis in many European and International research projects (e.g. Chemomentum **[CHEMOMENTUM]**) that use basic UNICORE components to implement advanced features and support scientific applications from a growing range of domains **[UNICOREPLUS]**. More recently, the first prototype of the Web services-based UNICORE 6 evolved that is based on emerging standard technologies such as WS-RF. The adoption of standards into Grid middleware systems provides interoperability among the different systems and thus makes the change from one middleware to another more easy and transparent to the scientists so that they can concentrate on their scientific workflows instead of Grid middleware evaluations. In the context of standards, UNICORE 5 used proprietary protocols such as the UNICORE Protocol Layer (UPL) **[UNICOREPROJECTS]** and proprietary job descriptions named as Abstract Job Objects (AJOs) **[UNICOREPROJECTS]**. The new Web services-based UNICORE 6, on the other hand, provides a WS-RF compliant interface layer that consists of several basic services for job management and file transfer collectively named as the UNICORE Atomic Services (UAS) **[STANDARDIZATIONPROCESSES]** developed during the European UNIGRIDS project **[UNIGRIDS].** Furthermore, the execution backend of UNICORE 6 was significantly improved **[XNJS]** in order to execute emerging standard compliant job descriptions based on JSDL. In addition the UNICORE Gateway **[GATEWAY]** was re-developed to authenticate Web service message exchanges between Grid clients and the UNICORE middleware. Note that one UNICORE Gateway can provide a single point of entry to multiple UNICORE Grid middleware installations. The UNICORE Gateway checks whether a request from an end-user uses a certificate that is signed by a trusted CA and whether it is valid and not revoked.

Illustration 5 shows the basic Web service-based architecture of UNICORE 6. It consists of a WS-RF hosting environment for Grid services based on Jetty **[JETTY]** and the XFire SOAP stack **[XFIRE]**. Furthermore, it consists of an execution backend and the Target System Interface (TSI) that interacts with the Resource Management System (RMS) (e.g. Torque, LoadLeveler) on a computational resource. Before the standardization of the OMII – Europe project begun, The Target System Factory (TSF) was used to create an instance of the Target System Service (TSS) of the UNICORE Grid middleware and thus implements the WS-RF factory pattern **[WSRF-TC]**. The TSS provides a proprietary interface to submit JSDL-compliant job descriptions to the UNICORE Grid middleware, while the Job Management Service (JMS) can be used to control and monitor the job afterwards. In addition, the Storage Management Service (SMS) and File Transfer Service (FTS) can be used for staging job related files in and out of the UNICORE middleware. The JSDL based job description is parsed and interpreted by the enhanced Network Job Supervisor (NJS) **[XNJS]** that also performs the authorization of users by using the enhanced UNICORE User Database (UUDB). After successful authorization, the rather abstract job definition is translated into non-abstract job descriptions, a process named as incarnation, by using the Incarnation Database (IDB) at the NJS. Finally the job is forwarded via the TSI to the RMS for scheduling on the HPC resource (e.g. supercomputer or cluster).

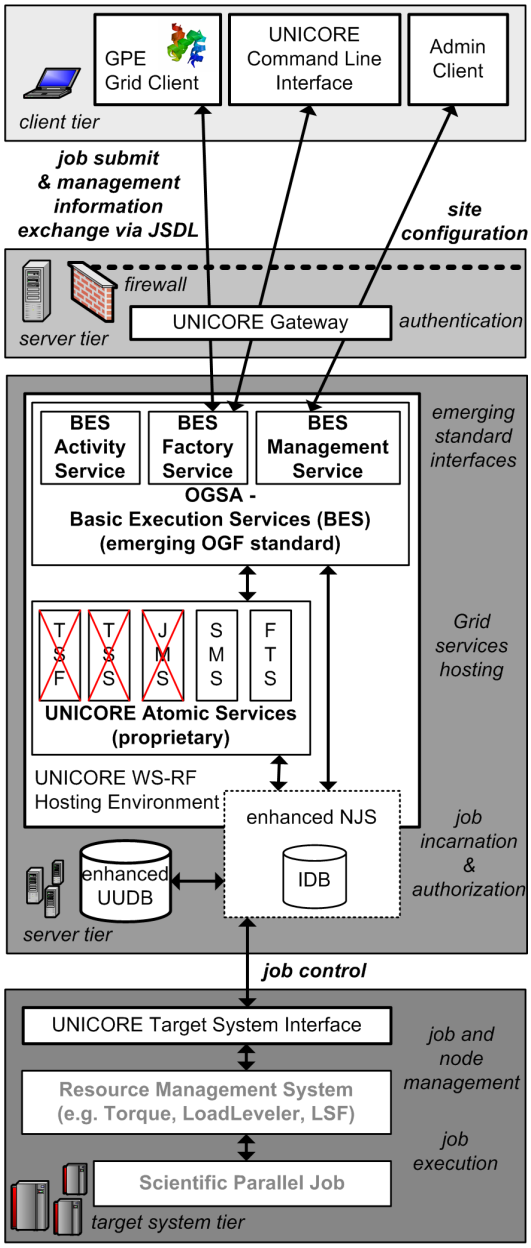

Illustration 5: UNICORE 6 with a deployed OGSA – BES services. In OMII – Europe the proprietary UNICORE Atomic Services are partly replaced by the standardized OGSA – BES services of the Open Grid Forum.

UNICORE 6 can be used with a wide variety of clients that follow the emerging standard guidelines of OGF and OASIS. For instance, the GPE client suite **[GPE]** provides three different clients that can be used to access resources managed by the UNICORE Grid middleware. While the UAS layer is still rather proprietary, the JRA1 activities of OMII - Europe currently augment UNICORE 6 with OGF standards such as the OGSA Basic Execution Services (BES) **[OGSA-BES],** OGSA Database Access and Integration Services (DAIS) **[WS-DAIS]** or OGSA Resource Usage Services (RUS) **[OGSA-RUS]** in order to provide standard compliant interfaces that ensure interoperability. In the context of the JRA1 job submission activity, the proprietary UAS are partly replaced by several standardized OGSA – BES services as shown in Illustration 5. In more detail, the TSS and TSF is replaced by the BES Factory that provides an operation that is capable of submitting JSDL documents named as *CreateActivity(JSDL)* operation. This leads to the creation of a job WS-resource that represents the computational jobs described by this JSDL. Furthermore, the JMS is replaced by the BES Activity service to control and monitor this job, for instance by using the *Terminate()* operation. For more details about the supported operations and properties of the interfaces please refer to Section 2 of this document.

Finally, the OGSA-BES augmentation for UNICORE also includes the BES Management service. This is a new service and improves the functionality of UNICORE in terms of administration. Therefore, an administration client as shown in Illustration 5 can be used to access the BES Management service and thus control whether new jobs can be submitted to UNICORE or not. Therefore the operations *StopAcceptingNewActivities()* and *StartAcceptingNewActivities()* are supported by UNICORE.

3.3.2 UNICORE Activity State Model

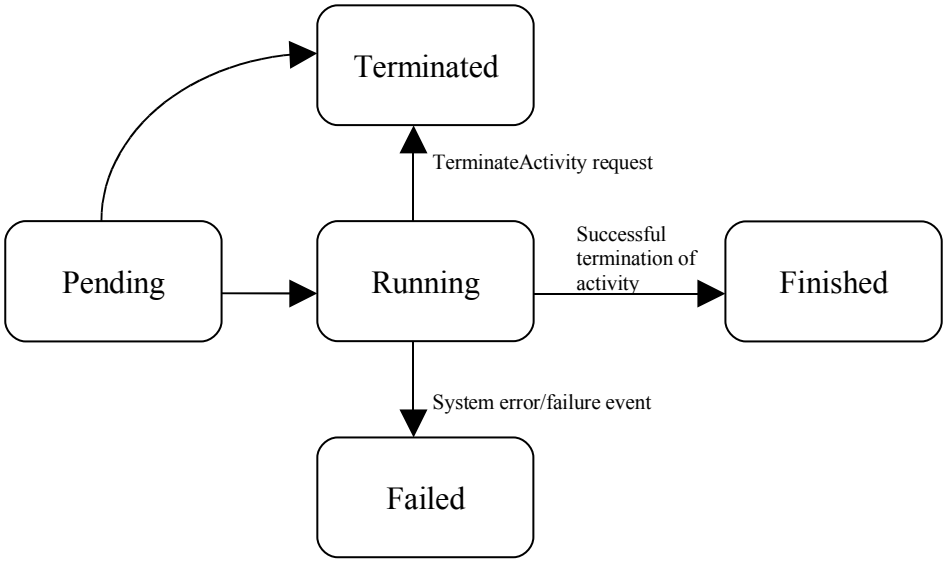

Illustration 6: BES Activity Basic State Model

In any batch or execution activities are the major entities to get privileges of being submitted, executed or cancelled. Deriving robust and simplified model for activities is challenging when activities comprises of complex tasks. Each activity dynamicaly encapsulates concrete states and each state in turn corresponds to the level of progress the computational job has approached. OGSA - BES defines a basic state model (see Illustration 6) that is specifically designed for a simplified way of monitoring job statuses. Thus it can be adapted in most of the other Grid middleware systems or job management systems and can be easily conceivable. The developed UNICORE state model (see Illustration 7) is designed to include data-staging and therefore regarded as *data-staging profile* in BES terms.

OGSA – BES activities in UNICORE are passed through five different states from the job submission till the job is finished or job termination. These states are named as *Pending, Staging-in, Executing, Staging-out* and *Finished*. In addition, to signify exceptions during the job processing, there are the states *Failed* and *Terminated*. The states are listed within Illustration 6.

In more detail, the states within the OGSA-BES implementation of UNICORE are used to represent a job status as follows:

Pending
Activity is created by a service but not yet instantiated by any UNICORE resource or at least enabled execution to start on particular resource.

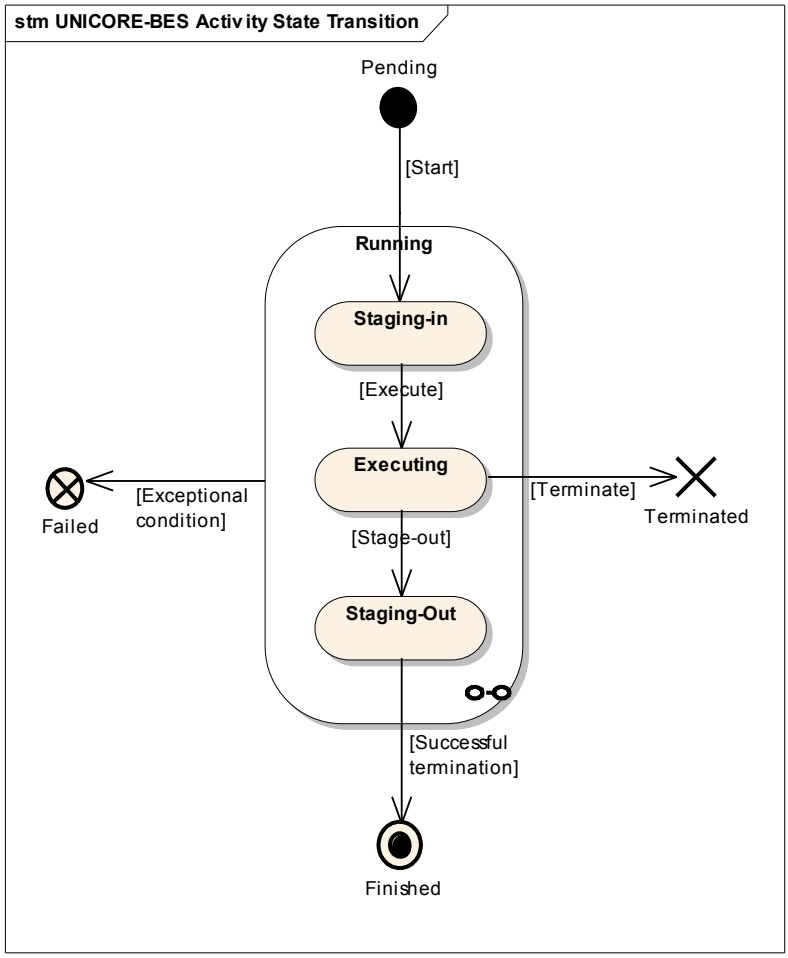

Illustration 7: UNICORE Activity State Model

Staging-In
This state represents a specialized state of the '*Running' state* as shown in Illustration 7. Thus *Running* is itself a smaller state machine that encapsulates substates. So this state indicates that the activity in which users data being loaded into some storage before starting any execution.

Executing
Substate of '*Running'* represents the UNICORE job (activity in OGSA-BES terms) currently being processed on a computational resource.

Staging-Out
Also one of the sub-states of '*Running'* that indicates that the data is being transferred after execution of the job into a remote location.

Finished
It represents that the job is successfully completed.

Terminated
This state is also called an aborted state. The client job can be terminated on a terminate request by an end-user. This request might not necessarily be terminate by the client who creates particular job, instead an administrator may uses the admin client to abort jobs.

Failed
This state may be the result of un-successful execution of the computational job. It is being due to some exceptional reason, for instance time-out or resource unavailable.

4. Evaluation and Discussions

The OGSA-BES interfaces are currently adopted by all Grid middleware systems. From our current experience, the following issues have been observed so far.

4.1 Emerging standards specification

The BES specification is still under development, although the currently available version (draft 33) is mostly ready for entering the *public comment* phase. Implementing an evolving specification is always risky, as changes in the specification may arise unexpectedly during the standardization process. For an OGSA-BES Supercomputing 2006 interoperation demo, for instance, it was decided to temporarily "freeze" the draft version 26 of the specification so that implementers could have the time to implement that version of BES. However, we expect that the BES specification will be finalized shortly, and within the timeframe of the OMII-Europe project, so that we will able to provide a stable implementation by month 20 (as requested by MJRA1.10 milestone: "OMII-Europe supports BES plus required extensions"). Currently the middleware platforms all adopt the most recent version that is draft 33. The interoperability of these implementations is tested within the JRA3 Task 2 activity during the deployment of the implemenations in the multi-platform infrastructure.

4.2 Lack of a standard security profile in relation to OGSA-BES

The BES itself is concerned with job submission and management, so security considerations have been kept out of the specification. However, it is obvious that actual deployment of BES-enabled services will need a strong authorization/authentication mechanism. To provide an example, within the Supercomputing 2006 interoperation demo, the services were required to implement the WS Security/Username Token specification [WS-SEC] over HTTP/SSL encrypted connections. This simple approach was only an interim solution for the demo itself, and is not suitable for large-scale deployment. The OMII-Europe project has a task, in particular JRA3 Task 1 - Common Security Infrastructure Infrastructure, which goal is to define a core set of security features that will enable users to make use of multiple middlewares with a single underlying security infrastructure. The primary focus will be on interoperability between Globus, gLite, and UNICORE, and eventually will result in the definition of a common security technology "profile", that is a document that describes a set of technologies and standards, or subparts of these, that will and can be supported by all the "compliant" middlewaresand baseline technologies.

4.3 Support for WS-RF technology standard is sketched, but not clearly defined

The BES specification states in its introduction that implementations may support other resource models and related access mechanisms, in particular by composing appropriate port-types from the WS-RF/WS-Notification [WSRF-BASE], WS-Transfer/WS-Eventing or WS-ResouceTransfer families of specifications. However, details on how BES-compliant services are expected to do that in a commonly agreed way are not given. The rendering of BES activity properties within the WS-RF Basic Profile is sketched in Appendix I of the BES specification [OGSA-BES], but that is not intended to be normative. It is advisable that more normative details are given with respect to the optional WS-RF-* binding of the BES specification. This, for instance, can be realized by the OGSA-BES group by providing one core specification of OGSA-BES in abstract IDL. In addition, separate rendering documents that define the concrete mappings of the abstract OGSA-BES functionality to the correspondend models WS-RF, WS-Transfer, and others could be defined. However, since the process of the specification is already quiete far in the OGF editorial process it is not assumed that they will change their style of the document.

4.4 Missing functionality with respect to data staging for computation

During the adoption of OGSA-BES (which implies JSDL) within the project it becomes clear that OGSA-BES has some issues. We provide an example in the context of the UNICORE Grid middleware and its adoption of OGSA-BES. In particular, within the EuropeanUniGrids project **[UNIGRIDS]**, the UNICORE community has developed the UNICORE Atomic Services (UAS) as desribed above. Although the project ended in July 2006, its defined uniform interfaces to Grid Services, the UAS, lead to a major impact on the Grid community since it was the first time that UNICORE and Globus developers worked together on a standard execution interface to their systems. In more detail, both communities jointly developed the Execution Services Interface (ESI) **[ESI]** taking Globus GRAM execution management requirements and requirements of the UAS of UNICORE into account. The ESI specification in turn was given as a comprehensive input into the OGSA - BES working group of OGF. The idea was that the OGSA-BES specification should be revised by taking the ESI specification into account that contents requirements of the UNICORE and Globus community.

To provide an example, OGSA-BES only focusses on simple job executions that are described by JSDL documents. The UAS on the other hand cover this functionality, but also provides an interface for storage and filetransfer, named as Storage Management Service and File Transfer Service. Both are shown in Illustration 5. Both are still needed in conjuntion with OGSA-BES in order to provide the functionality that is needed for JSDL-based job executions that rely on staged data. The following JSDL document provides a typical example of a job execution that works with data staging. In the shown simple example, a *test.txt* file is used for computation (cat of it) and should be staged into the job workspace before job execution. In this context, the JSDL document specifies an URI to the corresponding service that is able to perform the data staging.

```
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:JobDefinition xmlns:p="http://schemas.ggf.org/jsdl/2005/11/jsdl"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.ggf.org/jsdl/2005/11/jsdl jsdl.xsd
">

  <jsdl:JobDescription>
    <jsdl:Application>
      <jsdl:ApplicationName>Cat</jsdl:ApplicationName>
      <jsdl:ApplicationVersion>1.0</jsdl:ApplicationVersion>
    </jsdl:Application>

    <jsdl:DataStaging>
      <jsdl:FileName>/infile</jsdl:FileName>
      <jsdl:CreationFlag>overwrite</jsdl:CreationFlag>
      <jsdl:Source>
        <jsdl:URI>RBYTEIO:http://127.0.0.1:7777/services/StorageManage
ment?
        res=default_bes_storage#test.txt
        </jsdl:URI>
      </jsdl:Source>
    </jsdl:DataStaging>

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

In this particular example, the JSDL job description is used to submit a job to UNICORE through the OGSA-BES interface. The JSDL is referencing the StorageManagementService in order to start a FileTransferService operation that is used to transfer the file into the working directory of UNICORE (see also Illustration 5). Hence, it becomes clear that basic job execution should include simple data staging capabilities that are not provided as a concrete interface in the OGSA-BES specification.

To conclude, data staging is an important feature which is needed for job execution, since many jobs rely on input data. However, since this issue is outside the scope of OGSA-BES, a new OGF group has been created that will eventually cover the functionality of the File Transfer Service of the UAS, named as the OGSA - Data Movement Interface (DMI) working group [OGSA-DMI]. Nevertheless, a concrete interface to storages such as the Storage Management Service of the UAS is still mssing, but there is already some work in the Grid Storage Management (GSM-WG) [GSM-WG] that might be useful as well as implementations in the context of storage resource managers (SRM) and storage resource brokers (SRB). This raises a demand for the adoption of the OGSA-DMI specification and storage managers by BES-compliant services in general, and by the middleware platforms of OMII – Europe in particular. Hence, it is strongly recommended by this JRA1 – job submit activity that OMII – Europe 2 will try to augment the major Grid middleware platforms with the upcoming OGSA – DMI interface developed in OGF as well as support for storage resource managers/brokers.

4.5 BES Management portType

The functionality of the BES Management portType provides less functionality for a complete management of a service container in the context of job executionl. However since it is not covered by any specification before it provides additional functionality that covers at least the basic job submission itself with *StartAcceptingNewActivities()* and *StopAcceptingNewActivities()*.

5. Conclusions

In this document we described the current status of job submission interfaces in the gLite, UNICORE and Globus Grid middleware systems. We analyzed the latest draft of the BES specification, describing its main features and our experience in implementing it within the major Grid middleware Systems within OMII – Europe. Our experience with the implementation of the BES specification was quite positive, even if there are some general remarks. Some of the early prototypes developed within OMII – Europe in this activity was used in the Supercomputing 2006 interoperation demonstrations. These demonstrations show how many different OGSA - BES implementations were interoperable through OGSA-BES on a very low level. However, in the context of JRA3 – Task 2 within this project, these interoperation efforts will be improved in terms of real interoperability.

While some issues need to be addressed (as described in the previous section), we consider those issues to be relatively minor with respect to the advantage of having a working, standard interface for job submission and management across different Grid systems. However, the adoption of OGSA-BES is not enough to address the basic requirements, especially in terms of data staging. Therefore, we state a clear demand for the integration of OGSA-DMI interfaces into the Grid middleware systems within OMII – Europe 2. Furthermore, storage management is required that can be covered by storage resource managers and brokers. Once implementations of OGSA-BES and OGSA-DMI are deployed within the Grid middleware platforms, an important set of standardized interfaces for job submission is accomplished and supported by storage managers.

6. References

[CHEMOMENTUM]
http://www.chemomentum.org

[CREAM]
P. Andreetto *et al.*, CREAM: A simple, Grid-accessible, Job Management System for local Computational Resources, in Proc. CHEP'06, Mumbay, India, 13-17 February 2006

[ESI]

Execution Services Interface

http://www.unigrids.org/deliverables/ESI.pdf

[GATEWAY]

R. Menday. The Web Services Architecture and the UNICORE Gateway. In Proceedings of the International Conference on Internet and Web Applications and Services (ICIW) 2006, Guadeloupe, French Caribbean, 2006.

[GPE]
R. Ratering, M. Riedel, A. Lukichev, D. Mallmann, A. Vanni, C. Cacciari, S. Lanzarini, P. Bala, K. Benedyczak, M. Borcz, R. Kluszcynski, and G. Ohme. GridBeans: Supporting e-Science and Grid Applications. In 2nd IEEE International Conference on e-Science and Grid Computing (E-Science 2006), Amsterdam, The Netherlands, 2006.

[GSM-WG]

OGF Grid Storage Management Working Group
https://forge.gridforum.org/projects/gsm-wg/

[GT4]

GT 4 WS GRAM Approach,
http://www.globus.org/toolkit/docs/4.0/execution/key/WS_GRAM_Approach.html

[JETTY]

Jetty WebServer. http://www.mortbay.org.

[JSDL]

A. Savva (editor), Job Submission Description Language (JSDL) Specification, version 1.0, GGF November 2005, available at www.gridforum.org/documents/GFD.56.pdf

[OGSA-BES]

I. Foster *et al.*, OGSA Basic Execution Service, Version 1.0, draft specification version 33, feb 22 2007.

[OGSA-DMI]

OGSA – Data Movement Interface Working Group

https://forge.gridforum.org/sf/projects/ogsa-dmi-wg

[OGSA-RUS]

OGSA – Resource Usage Services (RUS) Working Group

https://forge.gridforum.org/sf/projects/rus-wg

[OMII-JRA3SEC]

OMII-Europe JRA3/CommonSecurity Infrastructure activity page,
http://tjasse.pdc.kth.se/omii-europe/

[STANDARDIZATIONPROCESSES]

M. Riedel and D. Mallmann. Standardization Processes of the UNICORE Grid System. In Proceedings of 1st Austrian Grid Symposium 2005, Schloss Hagenberg, Austria, pages 191–203. Austrian Computer Society, 2005.

[UNICOREPLUS]

D. Erwin. UNICORE Plus Final Report - Uniform Interface to Computing Recources. 2000. ISBN 3-00-011592-7.

[UNICOREPROJECTS]

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

[UNIGRIDS]
http://www.unigrids.org

[WMPROXY]

G. Avellino *et al.*, Flexible Job Submission Using Web Services: the gLite WMProxy Experience, Proc. CHEP'06, Mumbay, India, 13-17 February 2006

[WMP-WSDL]

WSDL interface documentation for the WMProxy service,
http://trinity.datamat.it/projects/EGEE/wiki/tmpdoc/index.html

[OMII-EU-MJRA1.7]

OMII-Europe Milestone Document M:JRA1.7 “Definition of JSDL extensions”, 2006, http://grid.pd.infn.it/omii/milestones:jra17#milestone_document

[WS-ADDR]

D. Box, F. Curbera (editors), Web Services Addressing (WS-Addressing), W3C Member Submission 10 August 2004, http://www.w3.org/Submission/ws-addressing/

[WS-DAIS]

Database Access and Integration Services (DAIS), https://forge.gridforum.org/sf/go/proj1070.

[WSRF-BASE]

I. Foster *et al.*, OGSA WSRF Basic Profile 1.0, recommendation, sep 22, 2005, http://forge.ogf.org/sf/go/doc13542?nav=1

[WSRF-TC]

OASIS - WSRF Technical Committee. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf.

[WS-SEC]

A. Nadalin et al, Web Services Security: SOAP Message Security 1.1 (WS-Security 2004), OASIS Standard Specification, 1 February 2006, http://docs.oasis-open.org/wss/v1.1/

[XFIRE]

XFIRE. http://xfire.codehaus.org

[XNJS]

B. Schuller, R. Menday, and A. Streit. A Versatile Execution Management System for Next Generation UNICORE Grids. In Proceedings of the 2nd UNICORE Summit 2006 in conjunction with EuroPar 2006, Dresden, Germany, 2006.