EU project: RI031844-OMII-Europe

Architectural Overview of OGSA-BES Adoption in UNICORE 6

Support for Basic Executions based on the OGF OGSA-BES Standard

Note:

UNICORE 6 and all its components are open source under BSD license and available at the Sourceforge platform. The implementation and documentation of the OGSA-BES standard in UNICORE has been developed in the European OMII-Europe project under EC grant RIO31844-OMII-EUROPE, duration May 2006 - April 2008.

© 2008 by Forschungszentrum Jülich and Fujitsu Laboratories of Europe

Edition 1.0, *29th of February 2008*

This edition applies to the OGSA-BES adoption in version 1.0 within the UNICORE 6 distribution. For more information please visit www.unicore.eu or contact unicore-support@lists.sourceforge.net. The world-wide UNICORE community and the UNICORE Forum welcome comments and suggestions for further improvements.

For any further information and questions, please see http://omii-europe.org and http://support.omii-europe.org, or contact support@omii-europe.org. OMII-Europe welcomes users' comments.

OMII-Europe

This document © *Forschungszentrum Jülich GmbH, Fujitsu Laboratories of Europe, 2008*

Summary

In recent years, the UNICORE 5 Grid system evolved to a full-grown Grid middleware system used in daily production at supercomputing centers and research facilities worldwide. Furthermore, UNICORE serves as a basis in many International research projects that use UNICORE components to implement advanced features and support scientific applications from a growing range of domains. More recently, the new Web services-based UNICORE 6 was released that provides significant improvement in terms of extensibility and standard compliance.

This document focuses on the enhancements of resource management capabilities using the job management interface OGSA – Basic Execution Services (OGSA-BES) standardized within the Open Grid Forum (OGF). The OGSA-BES adoption is included in the core part of UNICORE 6 and thus available for use at www.unicore.eu .

Table of Contents

Architectural Overview of OGSA-BES Adoption in UNICORE 6 i
1. About This Document 1
2. Component Overview 2
2.1. What Is the OGSA – Basic Execution Services Implementation 2
2.2. Functionalities and Features of the OGSA-BES Adoption 2
2.3. Why Use the OGSA-BES Adoption in UNICORE 3
2.4. When to Use the OGSA-BES Adoption in UNICORE 3
2.5. Sub Component UNICORE Container 4
3. The Architecture of the OGSA-BES Adoption in UNICORE 5
3.1. High Level Design 5
4. Comprising Components 7
4.1. UNICORE Activity States and OGSA-BES States 7
4.2. XNJS and Target System Interface (TSI) 8
5. Comparable Technologies 9
6. Conclusion 10
7. Reference 11
8. Acknowledgements 11
Appendix A Glossary 12

Table Index

Table 1 Icons used in the document 1
Table 2 Table of the glossary 12

Figure Index

Figure 1 One use case scenario for using the OGSA-BES adoption in UNICORE 6. 4
Figure 2 High level design of the OGSA-BES adoption in UNICORE 6. 6
Figure 3 UNICORE Activity State Model. 7
Figure 4 UNICORE 6 Execution backend of OGSA-BES Implementation 8

1. About This Document

This document gives an architectural introduction of the adoption of the OGSA – Basic Execution Services (BES) standard within UNICORE 6. The expected audiences of this document are those who have already had the related technological background and want to know more about the design and implementation of the particular adoption of this open standard in UNICORE 6. For more information about the core technology UNICORE please refer to www.unicore.eu. In addition, some parts of this document have been also published in the proceedings of the 3rd Workshop on Scheduling and Resource Management for Parallel and Distributed Systems (SRMPDS) 2007 held at the 13th International Conference on Parallel and Distributed Systems in Taiwan with the title *"Enhanced Resource Management Capabilities using Standardized Job Management and Data Access Interfaces within UNICORE Grids"*.

Section 2 of this document describes the foundations for the OGSA-BES adoption in UNICORE 6 and discusses use cases that explain why and when using the OGSA-BES implementation is useful.

Followed by the general component overview, Section 3 will focus on the architecture of the OGSA-BES adoption in UNICORE 6, including high level and low level design features.

Section 4 lists the comprising components in more detail, especially focussed on security demands, used interfaces and sub components.

While UNICORE was one of the first technologies that adopted OGSA-BES, there are many other related technologies that have adopted this standard as well. Therefore, we provide shortly an overview of a comparable technology currently deployed in Grids using gLite and OMII-UK.

Finally, Section 6 provides concluding remarks about the OGSA-BES adoption in UNICORE 6.

The following table lists a full explanation of specific icons or signs used during the text.

Icon	Description
omii europe open middleware infrastructure institute	This is the trademark of OMII-Europe project. When this icon used, the content is related to the products or partners involved in the OMII-Europe project.
	Note well
UNICORE	This is the logo of UNICORE Grid technology. When this icon used, the content is related to the products or partners involved in the world-wide UNICORE community.

Table 1 Icons used in the document

2. Component Overview

This section provides a general introduction to the OGSA-BES adoption in UNICORE 6. It describes its dependencies to the UNICORE backend and outlines in which scenarios the OGSA-BES implementation is necessary.

2.1. What Is the OGSA – Basic Execution Services Implementation

The OGSA-BES implementation in UNICORE is a crucial component in the context of job submission and management and in order to understand it we shortly describe other implied technologies. First and foremost, the Job Submission Description Language (JSDL) defines standard XML schema elements for job descriptions that should be executed on computational Grid resources, for instance a simple execution of a an application on a parallel resource.

However, JSDL is only a language and schema for the standardized description of computational job submissions, but the submit operation itself is represented by the emerging standard interface named as OGSA-BES. In more detail the OGSA-BES implementation describes a Web services-based interface that is standardized by the OGF. In particular, the interface comprises functionality for the creation, monitoring, and control of computational jobs. Within the OGSA-BES specification such jobs are names as activities described via JSDL documents.

Finally, the related HPC Basic Profile (HPC-BP) specifies the usage of the OGSA-BES interface in conjunction with certain extensions for HPC environments to JSDL. By using different security models, the OGSA-BES adoption within UNICORE is also compliant with HPC-BP.

2.2. Functionalities and Features of the OGSA-BES Adoption

The main functionality of the here described OGSA-BES adoption of UNICORE is to use this service for computational job submissions to UNICORE. This means that jobs that are send by end-users are actually executed on the system underneath the interface. In the case of UNICORE, this might be often supercomputers or large clusters.

A closer look reveals that the OGSA-BES interface is similar to the Target System Service and Job Management Service of the Unicore Atomic Services (UAS) of UNICORE 6. While the UAS rely on standardized WS-RF compliant message exchanges, the syntax of several UAS operations are still proprietary such as the actual *submit()* operation that takes a JSDL document as parameter.

Thus, the additional benefit of the OGSA-BES adoption in UNICORE 6 is the provisioning of a standardized syntax of Web service operations that deal with job control and management. Nevertheless, the UAS and the OGSA-BES implementation presented here are using internally the same UNICORE backend named as enhanced Network Job Supervisor (XNJS). Hence, the Web service layer is well encapsulated from the lower-level backend execution engine. This backend in turn forwards the job to an interface at the target system and submits it to a resource management system or batch subsystem (e.g. Torque, LoadLeveler, SunGridEngine, etc.).

2.3. Why Use the OGSA-BES Adoption in UNICORE

In recent years, the UNICORE 5 Grid system evolved as one of the major Grid middlewares used for resource management at supercomputer centers and research facilities worldwide. UNICORE is deployed on top of the site-specific Resource Management Systems (RMSs) (e.g. Torque or LoadLeveler) and thus it abstracts from their specific behavior and access mechanisms providing a seamless and secure access to heterogenous resources.

More recently, the first release of the Web services-based UNICORE 6 become available that is based on standard technologies such as WS-RF or WS-Addressing. The adoption of standards into the Grid technologies provides interoperability among the different systems and thus makes the change from one to another more easy and transparent to the end-users. In the context of open standards, UNICORE 5 used proprietary protocols such as the UNICORE Protocol Layer (UPL) and proprietary job descriptions named as Abstract Job Objects (AJOs).

The new Web services-based UNICORE 6 on the other hand, provides a more standard compliant interface layer to the underlying resource management capabilities. Therefore, it consists of several basic services for job management and file transfer collectively named as the UNICORE Atomic Services (UAS). In addition the UNICORE Gateway was re-developed to authenticate Web service message exchanges between Grid clients and the new UNICORE Grid middleware. As within UNICORE 5, the UNICORE Gateway provides a single point of entry to multiple UNICORE Grid middleware installations. In particular, the UNICORE Gateway checks whether a request from an end-user uses a certificate that is signed by a trusted CA, valid and not revoked.

However, even if the UAS work with JSDL, end-users should use the OGSA-BES implementation to be more flexible in choosing the right system according to their needs. This means if end-users use an OGSA-BES implementation of technology, it is by far easier to switch their clients and applications to another OGSA-BES implementation of technology B. So if one technology does not satisfy the end-users requirements they are able to use the other technology without changing their client. At least if both technologies agree on a common security profile.

2.4. When to Use the OGSA-BES Adoption in UNICORE

One particular use case scenario for the OGSA-BES implementation in UNICORE is to use the interoperability between UNICORE and gLite (with CREAM-BES). This is the case for projects that would like to use massively parallel resources with UNICORE and embarrassingly parallel resources with gLite (with CREAM-BES). This means an end-user would like to submit a computational job to the OGSA-BES implementation of UNICORE using for instance his GridSphere portal. This is typically no problem since interoperability is not so far required.

When the end-user would like to submit a computational job to gLite (with CREAM-BES) using the same client interoperability becomes a crucial part. Hence, end-users use OGSA-BES adoptions when they would like to submit computational jobs to different Grids without changing their client environments. Of course, there have been so far some portals that do this already, but normally using tweaks or only using proprietary ways of submitting jobs to both systems. The usage of proprietary solutions however impose significant maintenance issues in the clients that are all not relevant to OGSA-BES implementations since they are fixed standards.

In addition, as shown in Figure 1, both CREAM-BES of gLite and the OGSABES interface of UNICORE 6 supports the common security profile of OMII-Europe, which includes the support of Virtual Organization Membership Service (VOMS)-based authorizations that are based on the Security Assertion Markup Language (SAML) standard. Hence, apart from interoperability on the OGSA-BES level as described earlier, the security setup lays the foundation for secure interoperability of both systems. This interoperability of both systems is used by usage scenarios of the WISDOM project.

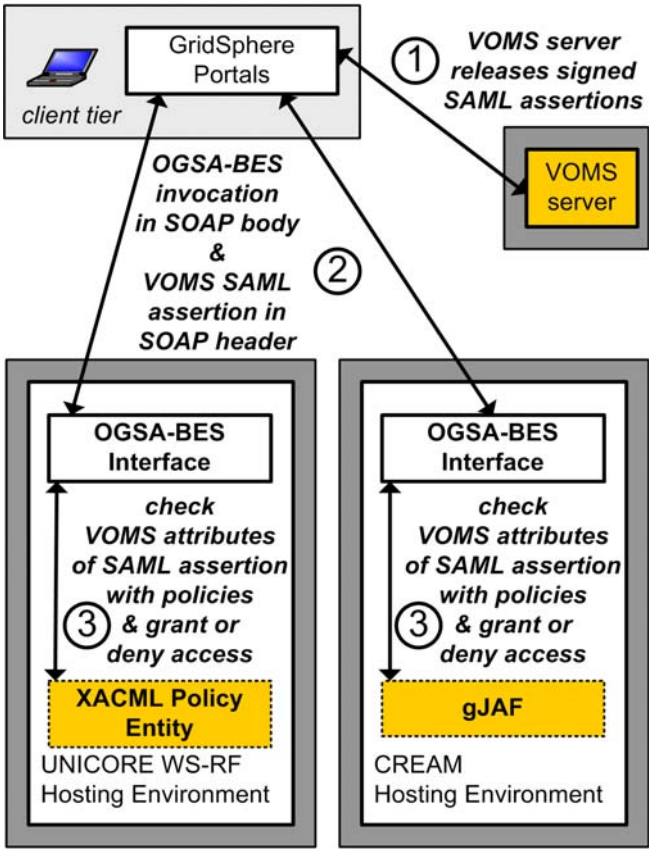

Figure 1 One use case scenario for using the OGSA-BES adoption in UNICORE 6.

2.5. Sub Component UNICORE Container

A fundamental sub component in the context of the OGSA-BES implementation for UNICORE is the UNICORE container based on Jetty and the xFire SOAP engine. All UNICORE services are deployed within this container that represents a powerful front-end with high performance for the UNICORE backend named as enhanced Network Job Supervisor (XNJS). This sub component is responsible for the job management within the core of UNICORE and thus this component plays a major part during the job submission process with OGSA-BES as described in the next chapters.

3. The Architecture of the OGSA-BES Adoption in UNICORE

This section introduces the architecture of the OGSA-BES adoption in UNICORE. While this section describes a high level design, Section 4 addresses some specific low level design issues related to the comprising components that make it easier to understand this component.

3.1. High Level Design

Figure 2 shows the overall design of the OGSA-BES adoption within UNICORE 6 and its dependencies to the sub components and UAS.

A closer look reveals that the OGSA-BES interface is similar to the Target System Service and Job Management Service of the UAS of UNICORE 6. While the UAS rely on standardized WS-RF compliant message exchanges, the syntax of several UAS operations are still proprietary such as the actual submit() operation that takes a JSDL document as parameter. Thus, the additional functionality of OGSA-BES interfaces for UNICORE 6 is the provisioning of a standardized syntax of Web service operations that deal with job control and management.

Nevertheless, the UAS and the OGSA-BES implementation presented here are using internally the same XNJS. Hence, the Web service layer is well encapsulated from the lower-level execution engine. All OGSA-BES invocations are transmitted to the server-side within the SOAP body, except pieces of information within the SOAP header that are used to address specific job WSResource instances and security tokens.

As shown in Figure 2, the integrated OGSA-BES interface of UNICORE 6 consists of three standardized services. First and foremost, the BES-Factory Service is responsible for the creation and control of a set of activities described with JSDL. In our implementation, this Web service interface uses the newly developed and enhanced NJS as execution backend that provide significantly improvements over production UNICORE 5 in terms of performance and scalability.

To provide an example, the CreateActivity(JSDL) operation of the BES-Factory Service leads to the creation of a job resource within the enhanced NJS that represents the computational job described by the JSDL. On the Web service level, this resource can be controlled and monitored by using the BES-Activity Service operations, for instance by using the Terminate() or GetStatus() operations. In addition, the OGSA-BES implementation for UNICORE also includes the BES-Management Service. This service in particular improves the functionality of UNICORE in terms of remote administration. Thus, an administration client, for instance, can be used to access this service and thus 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. A more detailed description about the actual submission on the target system is provided in the next section.

Finally, UNICORE is also compliant with the HPC – Basic Profile. In particular, the profile specifies the usage of the OGSA-BES interface in conjunction with certain extensions to JSDL. In more details, we implemented the 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.

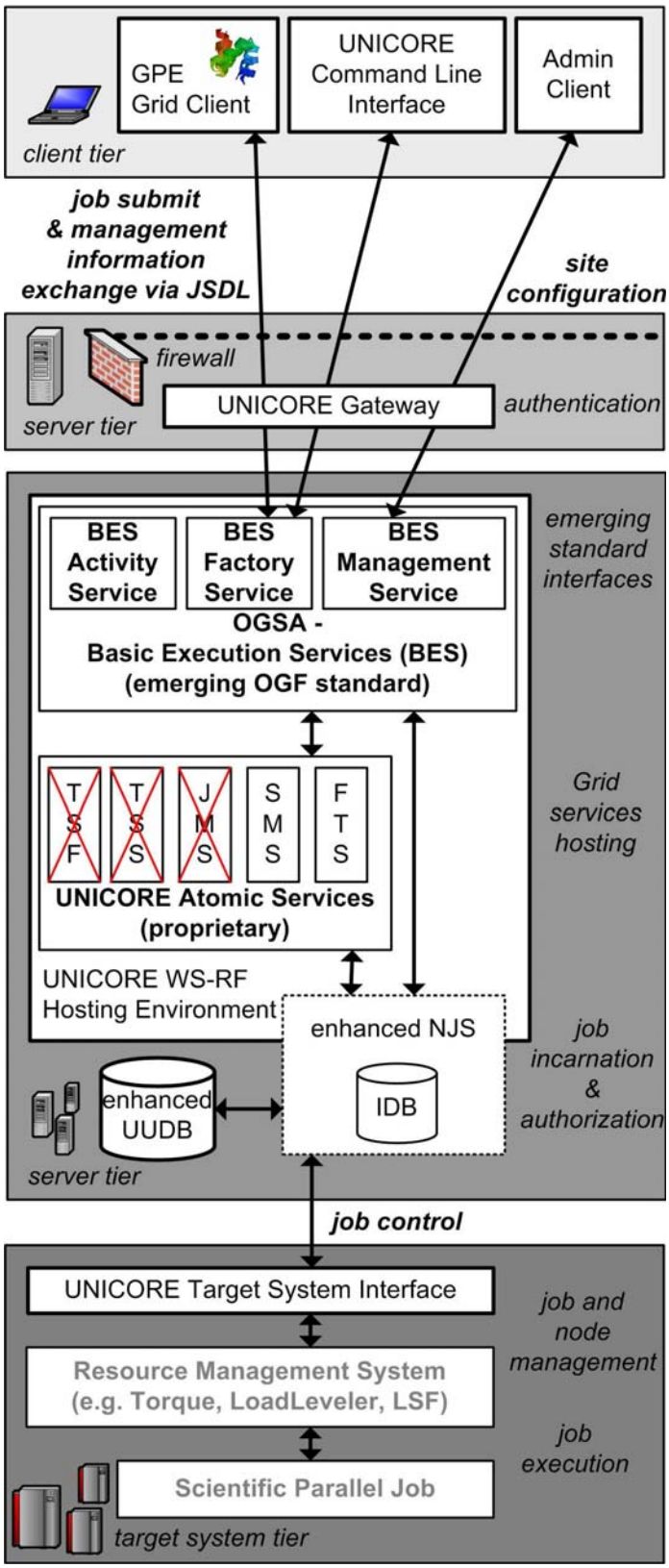

Figure 2 High level design of the OGSA-BES adoption in UNICORE 6.

4. Comprising Components

As described in Section 3, the OGSA-BES adoption for UNICORE 6 consists of a number of comprising components. This section highlights some of the comprising components and related technologies that are directly involved in the OSAG-BES-based job submission process.

4.1. UNICORE Activity States and OGSA-BES States

In any batch submission or execution, activities are the major entities to get privileges of being submitted, executed, or canceled. Deriving a robust and simplified model for activities is challenging when activities comprise of complex tasks. Each activity dynamicely encapsulates a concrete status and each state in turn corresponds to the level of progress the computational job has approached. In the context of this work, the OGSA-BES specification defines a basic state model that has been mapped to the UNICORE state model in order to achieve a good integration. The here developed UNICORE state model as shown in Figure 3 is designed to include data-staging and therefore regarded as data-staging profile in OGSA-BES terms.

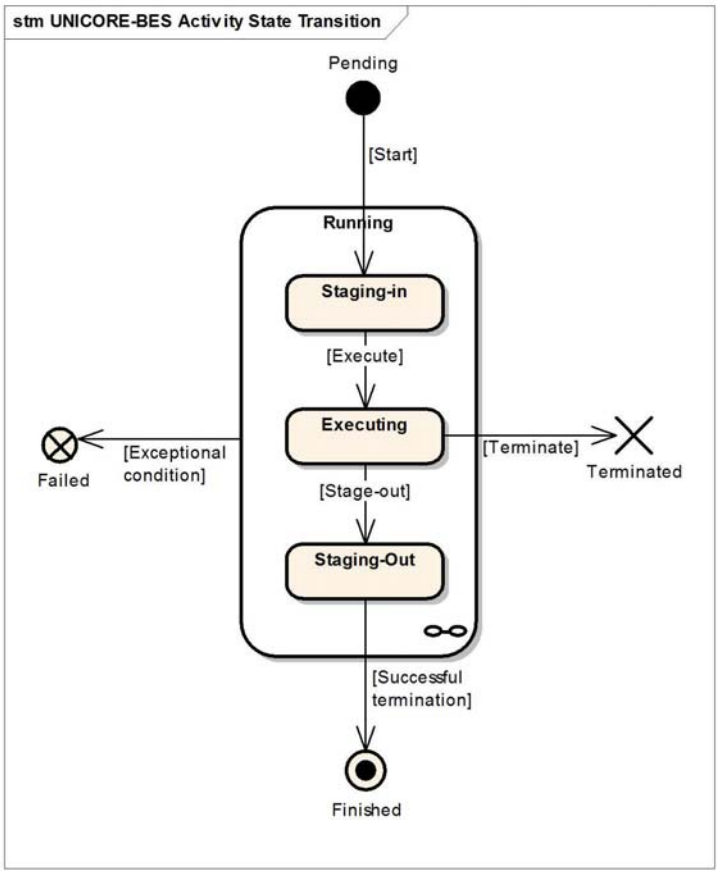

Figure 3 UNICORE Activity State Model.

More information about the details of the OGSA-BES implementation and its architecture can be found by viewing the source-code provided in Subversion (SVN):

http://unicore.svn.sourceforge.net/viewvc/unicore/ogsabes/

4.2. XNJS and Target System Interface (TSI)

Figure 4 shows the dependencies between the OGSA-BES implementation and the powerful XNJS backend. Any computational job that is submitted to the OGSA-BES implementation is internally forwarded to the XNJS component. This component parses the JSDL description and maps the job description to target specific commands. In addition it enforces authorization based on Extensible Access Control Markup Language (XACML) polices and the UNICORE User Database (UUDB).

After successful authorization checks the computational job is forwarded to the Target System Interface (TSI) on the target machine. A TSI interfaces with a local resource management system or local batch sub-system (e.g. Torque, LoadLeveler, etc.). Hence, a UNICORE TSI is finally used to submit a computational job to these systems.

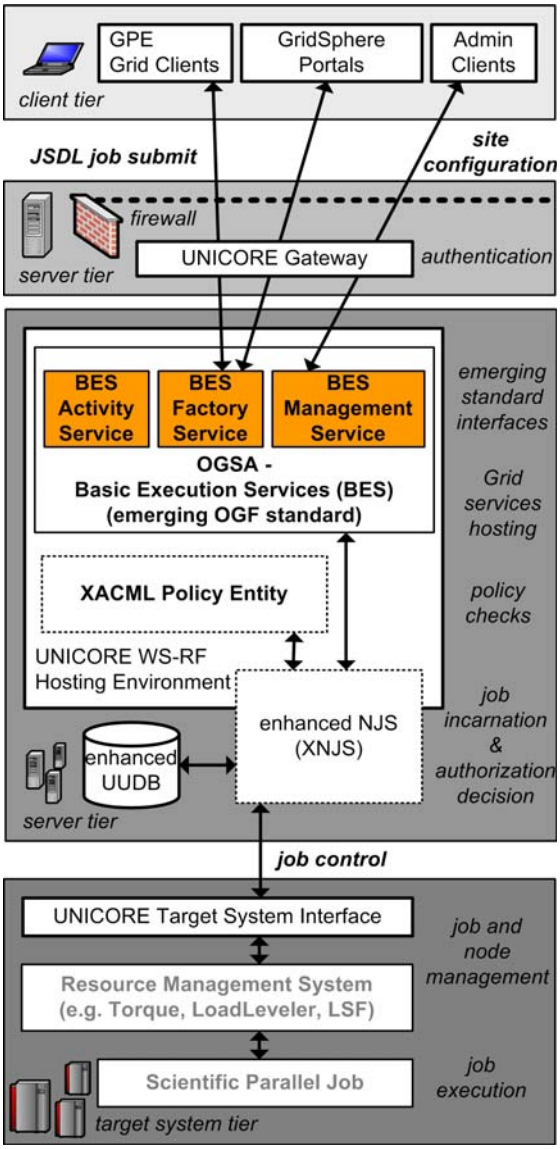

Figure 4 UNICORE 6 Execution backend of OGSA-BES Implementation

5. Comparable Technologies

In the last months, many vendors and middleware providers have started to augment their system with an OGSA-BES interface to enhance their standardized resource management capabilities. More precisely, the most of them even provide support for the HPC – Basic Profile, which includes the usage of the OGSA-BES interface in conjunction with dedicated JSDL elements.

A well known comparable technology in terms of the OGSA-BES implementation is provided by OMII-UK and named as the Grid Job Submission and Monitoring Service (GridSAM). It is an open source project providing Web services interfaces for submission, monitoring and execution of JSDL jobs and offers an interface compliant with OGSA-BES. The core of GridSAM is its Job Management Library (JML) which encapsulates message exchange mechanisms from JSDL to the RMS-specific commands of the correspondent underlying batch sub-system.

The JML is responsible for orchestrating the execution of so called Distributed Resource Manager (DRM) - Connectors that is a reusable component comprising job management functions similar to the UNICORE TSI. All in all, it is similar to the XNJS and TSI, but UNICORE 6 as a whole represents a complete Grid middleware with a lot of other integrated functionalities (e.g. security, standardized file-transfers, workflows, etc.) that is the major difference between the OGSA-BES adoption within UNICORE 6 and GridSAM.

The Computing Resource Execution and Management (CREAM) system is a simple, minimal system designed to provide efficient processing of a large number of requests for computation on managed resources within the gLite middleware used by the EGEE e-infrastructure. The CREAM architecture is designed to be a robust, scalable and fault tolerant service and is currently reengineered within OMII-Europe with an OGSA-BES interface.

The difference to the OGSA-BES implementation described in this document is represented by the different nature of the middleware itself. While the gLite middleware is used in embarrassingly parallel scenarios on university clusters and PC pools, the UNICORE middleware is rather designed for the usage on supercomputers and High Performance Computing (HPC) resources in highly parallel scenarios.

OMII-Europe also provides an OGSA-BES interface for Globus Toolkits that can be seen as a Web service frontend to the Grid Resource Allocation Manager (GRAM) of Globus. Finally, many other commercial vendors (e.g. Microsoft, Altair Engineering, Platform Computing, HP) have adopted the OGSA-BES specification while UNICORE is used by Fujitsu, but is provided as an ready-to-use open source implementation on sourceforge under BSD license.

6. Conclusion

The additional benefit of the OGSA-BES interface for UNICORE 6 is the provisioning of a standardized syntax of Web service operations that deal with job control and management. In particular, we have described the proprietary resource management interfaces of the UAS and their replacement by the standard OGSA-BES adoption.

The interoperability among commercial and scientific technologies is very important to harness the capabilities of distributed HPC clusters and Grids to enable scientific discovery and business advantages for Virtual Organizations (VOs) today and in future. In this context, this demonstration also indicated the progress being made within the OGF and by the various standardization working groups in order to produce standards and accelerate their adoption.

Nevertheless, deploying the OGSA-BES implementations is an important next step to broadly incorporate implementations of OGSA-BES and the HPC-Profile into production Grid environments. The implementation described within this architectural overview relies on the WS-based UNICORE 6 middleware. UNICORE 6 will be soon evaluated by the DEISA and D-Grid e-Science infrastructures for production usage. When these production Grids shift their access methods from UNICORE 5 to UNICORE 6, the OGSA-BES implementation can be also deployed.

In general, an efficient usage of computational resources by using standardized access mechanisms and thus interoperable technologies must be improved with the goal to incorporate such tools into the usual work_ow of end-users. Once an implementation of the OGSA-BES and HPC-Profile implementation is deployed within production Grids such as DEISA or D-Grid, an important interoperable tool for an efficient use of e-Science infrastructures is accomplished.

7. Reference

[1] Shahbaz Memon and Shiraz Memon and Morris Riedel and Bernd Schuller and Daniel Mallmann and Bastian Tweddell and Achim Streit and Sven van den Berghe and David Snelling and Vivian Li and Moreno Marzolla and Paolo Andreetto – 2007 , "Enhanced Resource Management Capabilities using Standardized Job Management and Data Access Interfaces within UNICORE Grids", Proceedings of 3rd Workshop on Scheduling and Resource Management for Parallel and Distributed Systems SRMPDS 2007, ICPADS'07 - The 13th International Conference on Parallel and Distributed Systems Hsinchu, Taiwan

8. Acknowledgements

The work described in this document is partially funded by the OMII-Europe project under EC grant RIO31844-OMII-EUROPE, duration May 2006 - April 2008.

Appendix A Glossary

The technologies glossary of this document is listed here.

JSDL document	A Job Submission and Description (JSDL) document provides information about the computational job that should be submitted. JSDL is an emerging standard from the Open Grid Forum (OGF).
OGSA-BES Implementation	An OGSA-Basic Execution Service (BES) implementation provides the functionality to submit basic computational jobs described in JSDL.

Table 2 Table of the glossary