Enhanced Resource Management Capabilities using Standardized Job Management and Data Access Interfaces within UNICORE Grids

M. Shahbaz Memon, A. Shiraz Memon, Morris Riedel, Bernd Schuller,
Daniel Mallmann, Bastian Tweddell, Achim Streit
Central Institute of Applied Mathematics
Forschungszentrum Jülich
D-52425 Juelich, Germany
{m.memon}@fz-juelich.de

Sven van de Berghe, David Snelling, Vivian Li
Fujitsu Laboratories of Europe
Hayes, Middlesex UB4 8FE, UK

Moreno Marzolla, Paolo Andreetto
INFN Padova
35100 Padova, Italy

Abstract

Many existing Grid technologies and resource management systems lack a standardized job submission interface in Grid environments or e-Infrastructures. Even if the same language for job description is used, often the interface for job submission is also different in each of these technologies. The evolvement of the standardized Job Submission and Description Language (JSDL) as well as the OGSA - Basic Execution Services (OGSA-BES) pave the way to improve the interoperability of all these technologies enabling cross-Grid job submission and better resource management capabilities. In addition, the ByteIO standards provide useful mechanisms for data access that can be used in conjunction with these improved resource management capabilities. This paper describes the integration of these standards into the recently released UNICORE 6 Grid middleware that is based on open standards such as the Web Services Resource Framework (WS-RF) and WS-Addressing (WS-A).

1 Introduction

Over the last years, production Grids such as DEISA, EGEE, OSG or TeraGrid have begun to offer services with an increasing number of applications that require access to resources managed by multiple Grid technologies. In this context, many production Grids regularly interact with other Grids in a pair wise fashion, e.g. cross-Grid job submission. Nevertheless these single interoperation efforts are different from native interoperability. Long-term *interoperability* is the ability of Grid technologies to interact via common open standards. *Interoperation*, on the other hand, is defined as what needs to be done to get Grids to work together as a fast short-term achievement using as much existing technologies as available. Many examples of these interoperation efforts are conducted within the *Grid Interoperation Now (GIN)* [9] community group of OGF. Today, the Grid middlewares are integrated with standards to achieve interoperability such as the UNICORE Grid middleware.

In recent years, the UNICORE 5 Grid system [30] 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 (e.g. Chemomentum [3] or A-Ware [1]) that use UNICORE components to implement advanced features and support scientific applications from a growing range of domains [20]. More recently, the new Web services-based UNICORE 6 was released that provides significant improvement in terms of extensibility and standard compliance. Therefore, this paper focuses on the enhancements of resource management capabilities using the job management interface OGSA-BES [10] and data access mechanisms based on ByteIO [24] both standardized within the Open Grid Forum (OGF).

This paper is structured as follows. Section 2 provides a small overview of the recently released Web services-based UNICORE 6 middleware. Section 3 describes the enhancements to UNICORE 6 in the areas of resource management and data access through standardized interfaces. Section 4 provides evaluations and Section 5 a survey of related work. This paper ends with some concluding remarks.

978-1-4244-1890-9/07/$25.00 ©2007 IEEE

2. UNICORE and Resource Management

In recent years, the UNICORE 5 Grid system [30] 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. It is deployed within DEISA [4] and D-Grid [5], just to list but a few. More recently, the first release of the *Web services-based UNICORE 6* become available that is based on standard technologies such as WS-RF [8] 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)* [30] and proprietary job descriptions named as *Abstract Job Objects (AJOs)* [30]. The new Web services-based UNICORE 6 on the other hand, provides a WS-RF 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)* [28] developed during the European UniGrids project [14]. Furthermore, the execution backend of UNICORE 6 was significantly improved [29] in order to execute emerging standard compliant job descriptions based on JSDL and with more performance as in UNICORE 5. In addition the *UNICORE Gateway* [23] 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 and whether it is valid and not revoked.

Illustration 1 shows the basic Web service-based architecture of UNICORE 6. It consists of a WS-RF hosting environment for Grid services based on an embedded Jetty HTTP(S) server [7] and the XFire SOAP stack [15]. Furthermore, it consists of an execution backend and a perl-based *Target System Interface (TSI)* that directly interacts with the various RMSs (e.g. Torque, LoadLeveler) on a computational resource. Before standardization efforts with respect to OGSA-BES begun, the proprietary *Target System Factory (TSF)* was used to create an instance of the *Target System Service (TSS)* and thus implements the WS-RF factory pattern [8]. 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. The JSDL-based job description is parsed and interpreted by the *enhanced Network Job Supervisor (NJS)* [29] that also performs the authorization of users by using the *enhanced UNICORE User Database (UUDB)*. After successful authorization, the 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 *High Performance Computing (HPC)* resource (e.g. supercomputer or cluster).

This indicates how UNICORE is providing an additional layer on top of RMS systems or batch subsystems that typically handle the resource management on supercomputers or clusters. In more detail, a RMS-specific TSI implements the interface to the correspondent underlying RMS and runs as a stateless daemon on the target system tier as shown in Figure 1. Since the environment of HPC resources usually requires a lightweight installation, the set of available TSIs for the most common RMSs are implemented in perl, which in turn is commonly installed on supercomputer or clusters today. In addition to the wide variety of perl TSIs, UNICORE 6 also provides an *embedded Java-based TSI* that can be used for embarrassingly parallel scenarios. Hence, this TSI runs where the enhanced NJS runs and is thus able to run on Windows or UNIX-style PC pools without the need for a target system tier.

Finally, UNICORE 6 can be used with a wide variety of clients that follow the emerging standards guidelines of OGF, OASIS and W3C. For instance, the open source *Grid Programming Environment (GPE)* [27] provides three different clients that can be used to access resources managed by the UNICORE Grid middleware. This client suite is developed by Intel as an evolution of the production UNICORE 5 client motivated by the WS-based standardization processes of UNICORE. In particular, the GPE provides the *Expert Client* that is realized as a Java application to support end-users running the clients on typical workstations. This client offers all functionalities known from the UNICORE 5 client, including a powerful workflow-editor and the support for multiple application-specific plugins (i.e. GridBeans). Thus this client provides full featured access to UNICORE 6 Grids for expert users. The *Application Client* is a lightweight Java application intended to run on mobile devices. The goal of this client is to provide access to specific applications on the Grid. Finally, the Portal Client is a server application accessible by a typical Web browser.

UNICORE 6 is open source under BSD license and available under sourceforge. For more information about UNICORE 6, please refer to [13].

3. Enhancing the Resource Management and Data Access Capabilities in UNICORE

The *OGSA - Basic Execution Services (BES)* specification [10] describes an Web service-based interface that is standardized by the OGF. In particular, the interface comprises functionality for the creation, monitoring and control of computational jobs. In the context of this specification, such computational jobs are named as *activities* that can be described via JSDL. This section provides an overview of the integration of OGSA-BES interfaces into the new WS-based UNICORE 6 Grid middleware as shown in Figure 1, including security considerations and an emphasis on the internal state model as well as *ByteIO interfaces* for data access. Hence, this section demonstrates the enhancements of resource management capabilities and data access through these standardized interfaces.

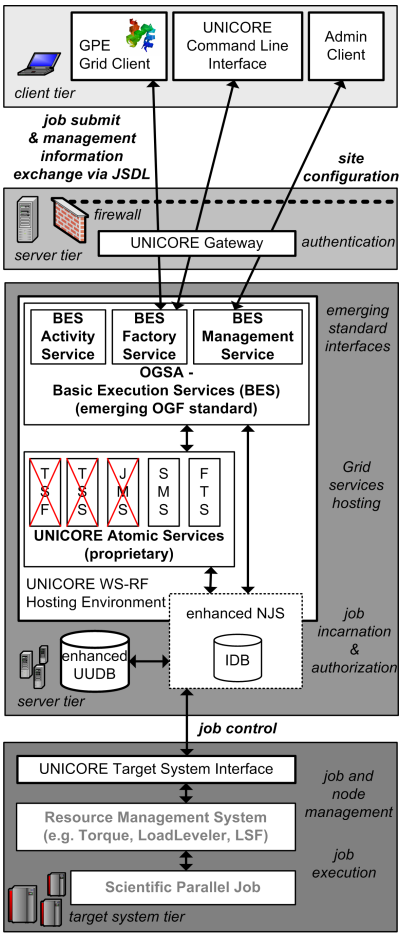

Figure 1. UNICORE 6 with deployed OGSA-BES services. The canceled proprietary UAS interfaces indicate their replacement by standardized OGSA-BES interfaces.

3.1 Standardized Web Services Interfaces

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 benefit 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 WS-Resource instances and security tokens.

As shown in Figure 1, 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.

Finally, UNICORE is also compliant with the *HPC - Profile* [11]. In particular, the profile specifies the usage of the OGSA-BES interface in conjunction with certain extensions to JSDL described in [22]. 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.

3.2 UNICORE Activity State Model

In any batch or execution activities in resource management 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* [10] that must be mapped to the *UNICORE state model* in order to achieve a good integration. The here developed UNICORE state model as shown in Figure 2 is designed to include data-staging and therefore regarded as *data-staging profile* in OGSA-BES terms [10].

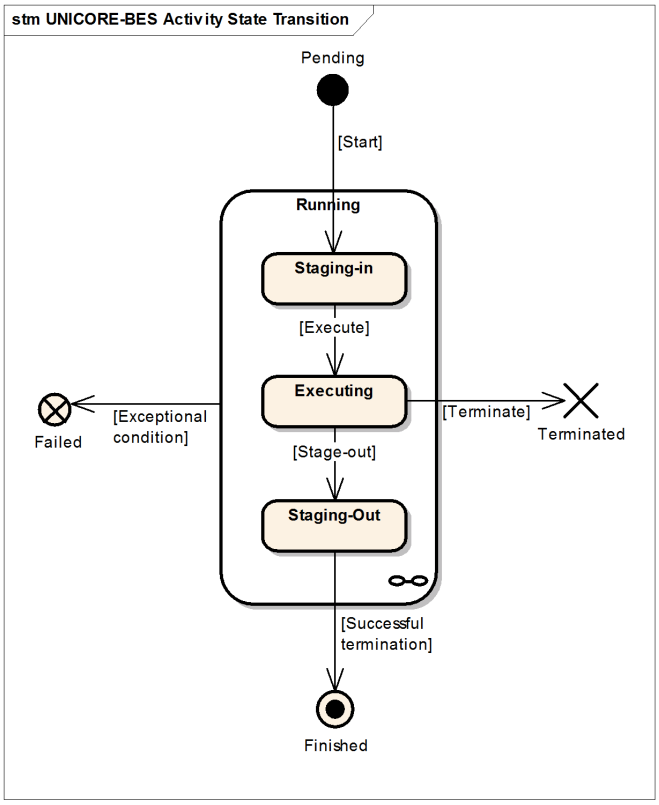

Figure 2. UNICORE Activity State Model.

As shown in Figure 2, OGSA-BES activities in UNICORE are passed through five different status 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* and are rather self-explaining. In addition, there are the states *Failed* and *Terminated*.

3.3 Data Access and File Transfer

During the adoption of the OGSA-BES interface (which includes JSDL) it becomes clear that it does not cover the complete atomic functionality in terms of job execution like the UAS. In most of the job management use cases, data needs to be staged-in before execution and after the job execution the results needs to be staged-out to a dedicated 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 respective implementers. But both file transfer and storage management are needed if JSDL descriptions have to be supported that make use of the data staging capabilities during job submission and execution. Therefore, the OGSA-BES implementation within UNICORE is working together with the File Transfer Services (FTSs) as described in Section 2. In more detail, the FTS provide the *Streamable ByteIO (SByteIO)* [24] and *Random ByteIO (RByteIO)* [24] data access mechanisms that we use in conjunction with the OGSA-BES interface 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.

In more detail, there are two variations to use these file transfer mechanism with OGSA-BES. The first option is via *local storage* that allows an end-user to upload the data of the job to the target machine where it will be used for job submission. Hence, this data will be gathered by the execution engine XNJS for further processing and job submission at the TSI. This, in particular, caters the entire scenario of data staging capabilities of JSDL. The second option is via *remote storage* wherein a separate dedicated machine is especially used for managing file transfer and data storage. In this scenario, the end-user first uploads the data to this third entity and and while sending the job the end-users specifies the location of this third host within the job described by JSDL. Tests reveal that this approach results in more staging turn around times in comparison with the local storage.

To sum up, the data transfer and access mechanisms are realized using the FTS and Storage Management Service (SMS) (see Figure 1). FTS is the actual service exposing the ByteIO interface specification and it is only responsible for staging data in and out during job submission and execution. In this context, the SMS represents a factory that creates different FTS instances for different data transfers. Furthermore, the SMS is used to manage the storage of data on all the tiers such as client tier, server tier, or even on a remote data server. All in all, we provide 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.

3.4 Support for WS-RF Rendering

The OGSA-BES specification states in its introduction that implementations may support other resource models and related access mechanisms, in particular by composing appropriate portTypes from the WS-RF families of specifications. The OGSA-BES activity properties for WS-RF resource models are provided and supported by the UNICORE 6 middleware. Hence, beside the typical operations defined in the OGSA-BES portTypes, UNICORE 6 also supports WS-Resource properties [8] requests of job instances to be compliant with the WS-RF resource model. This is supported because the resource model of UNICORE 6 itself is based on WS-RF and thus every instance, including job instances submitted through OGSA-BES by using a JSDL description, have their own properties. WS-RF implies that you can even query for properties, for instance enquiry the initial submitted JSDL.

3.5 Security Issues and Solutions

The OGSA-BES specification focuses on job submission and management and thus security considerations have been kept out of this specification. But since the OGSA-BES services are just deployed in parallel to the UAS within the hosting environment in UNICORE 6, all the strong security functionalities of UNICORE are inherited to these services as well. In particular, the *authentication* of jobs that are submitted to the OGSA-BES interface is done via the UNICORE Gateway.

The *authorization* is done by using the UUDB as described within Section 2. The information about the credentials of the end-users are transmitted inside the SOAP header while the actually OGSA-BES message exchanges are inside the SOAP body. In addition, a beta support already exist to work with a new VOMS [19] server that uses SAML-based [18] assertions to indicate project membership or role possession. In particular, the SAML assertions is received by a WS operation call to the new SAML-based VOMS server that releases signed SAML assertions. This SAML assertions are in turn transmitted within the SOAP header using WS-Security extensions along with the OGSA-BES invocations. At the UNICORE 6 server, we use an XACML [25] policy entity for authorization that checks these SAML assertions and grant or deny access.

Finally, to participate in the Supercomputing 2006 interoperability demonstrations of various OGSA-BES implementations, we provided support for the WS-Security Username Token Profile [26] over HTTP/SSL encrypted connections. Of course, this is a very simple approach only used during demo purposes of the OGSA-BES interface itself and clearly are not suitable for large-scale deployments like within DEISA or D-Grid.

4. Implementation Evaluations

The implementation proposed in this paper is currently evaluated in the testbed of OMII-Europe [12] for its production usage in DEISA and D-Grid in future. Furthermore, the WS interfaces provided by our implementation leverages the described new UNICORE backend and an analysis reveals that this new WS interface has not a high impact on performance. The overall evaluation in terms of performance analysis (on top of RMSs) was done in OMII-Europe by the neutral organization KTH in Sweden (a non UNICORE Forum member). They found out that UNICORE 6 provides very good performance and presented as well as published the detailed performance results at the UNICORE Summit 2007 in Rennes [16]. Finally, by our participation in the ByteIO interoperability fest [2] and the Supercomputing demonstrations in 2006, we have shown that our work is highly interoperable with other implementations.

5. Related Work

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 and provide support for the HPC-Profile (HPC-P) [11], which includes an OGSA-BES interface in conjunction with dedicated JSDL elements [22]. With the work described in this contribution, we achieved that UNICORE 6 is HPC-P compliant. A well known implementation of the OGSA-BES interface is provided by OMII-UK and named as the *Grid Job Submission and Monitoring Service (GridSAM)* [6]. 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) that is the major difference between our work and GridSAM. The OGSA-BES-enabled *Computing Resource Execution and Management (CREAM)* system [17] is a 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. While the gLite middleware is used in embarrassingly parallel scenarios, the UNICORE middleware is rather designed for the usage in massively 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)* [21] 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 [13] 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 describe the proprietary resource management interfaces of the UAS and their replacement by the standard interface OGSA-BES. In conjunction with the support for ByteIO data access mechanisms, we illustrated the enhanced resource management capabilities for standardized job management and data access in the recently released new UNICORE 6 middleware. Also, this paper describes the HPC-P support of UNICORE that was demonstrated in interoperability scenarios at the Supercomputing 2006 conference. 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 here 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 workflow 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.

Acknowledgment

The work presented in this paper has been supported by the OMII - Europe project under EC grant RIO31844-OMII-EUROPE, duration May 2006 - April 2008.

References

[1] A-WARE Project. http://www.a-ware.org/.
[2] ByteIO. http://forge.ogf.org/sf/wiki/do/viewPage/projects.byteio-wg/wiki/HomePage.
[3] Chemomentum Project. http://www.chemomentum.org/.
[4] DEISA. http://www.deisa.org/.
[5] German National Grid: D-Grid. http://www.d-grid.de.
[6] GridSAM. http://gridsam.sourceforge.net/2.0.1/index.html.
[7] Jetty Web Server. http://www.mortbay.org.
[8] OASIS - WSRF Technical Committee. http://www.oasis-open.org/committees/wsrf.
[9] OGF - Grid Interoperation Now (GIN-CG). http://forge.ogf.org/sf/projects/gin.
[10] OGSA - Basic Execution Services (OGSA-BES). http://forge.ogf.org/sf/projects/ogsa-bes-wg.
[11] OGSA - High Performance Computing Profile (OGSA-HPC-P). http://forge.ogf.org/sf/projects/ogsa-hpcp-wg.
[12] OMII - Europe. http://omii-europe.org/.
[13] UNICORE. http://www.unicore.eu/.
[14] UniGrids. http://www.unigrids.org/.
[15] XFIRE. http://xfire.codehaus.org.
[16] P. Alexius, B. Elahi, F. Hedman, P. Mucci, G. Netzer, and Z. Shah. A Black-Box Approach to Performance Analysis of Grid Middleware. In *Proc. of the 3rd UNICORE Summit*, 2007. http://www.unicore.eu/summit/2007/schedule.php.
[17] P. Andreetto et al. CREAM: A simple, Grid-accessible, Job Management System for local Computational Resources. In *CHEP 2006, Mumbay, India*, 2006.
[18] S. Cantor, J. Kemp, R. Philpott, and E. Maler. *Assertions and Protocols for the OASIS Security Assertion Markup Language*. OASIS Standard, 2005. http://docs.oasis-open.org/security/saml/v2.0/.
[19] Y. Demchenko et al. VO-based Dynamic Security Associations in Collaborative Grid Environments. In *Proc. of the Int. Symp. on Collaborative Technologies and Systems*, 2006.
[20] D. Erwin. *UNICORE Plus Final Report - Uniform Interface to Computing Recources*. 2000. ISBN 3-00-011592-7.
[21] I. Foster. Globus Toolkit version 4: Software for Service-Oriented Science. In *Proceedings of IFIP International Conference on Network and Parallel Computing, LNCS 3779*, pages 213–223. Springer-Verlag, 2005.
[22] M. Humphrey. *JSDL HPC Profile Application Extension*. OGF Draft.
[23] 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.
[24] M. Morgan et al. *ByteIO Specification*. OGF (GFD87), 2006.
[25] T. Moses et al. *eXtensible Access Control Markup Language*. OASIS Standard, 2005.
[26] A. Nadalin et al. *OASIS Web Security Username Token Profile 1.1*. 2006. OASIS Standard Specification.
[27] R. Ratering et al. 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.
[28] 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.
[29] B. Schuller et al. A Versatile Execution Management System for Next Generation UNICORE Grids. In *Proc. of the 2nd UNICORE Summit*, 2006.
[30] 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.