Improvements of Common Open Grid Standards to Increase High Throughput and High Performance Computing Effectiveness on Large-scale Grid and e-Science Infrastructures

M. Riedel, M.S. Memon, A.S. Memon,
A. Streit, F. Wolf, Th. Lippert
Jülich Supercomputing Centre
Forschungszentrum Jülich
Jülich, Germany
m.riedel@fz-juelich.de

B. Konya, O. Smirnova
Lund University
Lund, Sweden

A. Konstaninov
Vilnius University
Vilnius, Lithuania

L. Zangrando
INFN
Padova, Italy

M. Marzolla
University of Bologna
Bologna, Italy

J. Watzl, D.Kranzlmüller
Ludwig Maximillians University Munich
Munich, Germany

Abstract— **Grid and e-science infrastructure interoperability is an increasing demand for Grid applications but interoperability based on common open standards adopted by Grid middle-wares are only starting to emerge on Grid infrastructures and are not broadly provided today. In earlier work we have shown how open standards can be improved by lessons learned from cross-Grid applications that require access to both, High Throughput Computing (HTC) resources as well as High Performance Computing (HPC) resources. This paper provides more insights in several concepts with a particular focus on effectively describing Grid job descriptions in order to satisfy the demands of e-scientists and their cross-Grid applications. Based on lessons learned over years gained with interoperability setups between production Grids such as EGEE, DEISA, and NorduGrid, we illustrate how common open Grid standards (i.e. JSDL and GLUE2) can take cross-Grid application experience into account.**

Keywords: HPC; HTC; Interoperability, Open Standards

I. INTRODUCTION

Production Grid and e-science infrastructures such as NorduGrid [1], EGEE [2], or DEISA [3] provide a wide variety of different Grid resources to end-users (i.e. e-scientists) on a daily basis today. Nowadays we observe an increasing amount of e-science applications that require resources in more than one Grid often leveraging both HTC and HPC resources in one scientific workflow. But using different production Grids represents still a challenge due to the absence of a wide adoption of open standards in deployed Grid middleware today. Nevertheless, we have worked in the OGF Grid Interoperation Now (GIN) community group to enable cross-Grid applications between different production Grids. Some well-known examples are interoperability setups for the WISDOM [4] community, the EUFORIA project [5], or the Virtual Physiological Human (VPH) [6] community. Thus we identified in earlier work [7] a well-defined set of open standards that play an important role for production Grids today or are considered to be adopted soon in production Grids. These standards are OGSA - Basic Execution Service (BES) [8], Job Submission Description Language (JSDL) [9], Storage Resource Manager (SRM) [10], GridFTP [11], GLUE2 [12], and several standards from the security domain (e.g. X.509, Security Assertion Markup Language, etc.). Standards like Usage Records (UR), Service Level Agreements (SLAs) via WS-Agreement [13], WSDAIS, or the Data Moving Interface (DMI) are getting more important for production Grids as well.

These common open standards are a good step towards the right direction, but we also identified in earlier work [14] that many of them can be still improved by production Grid experience (e.g. OGSA-BES and JSDL), or that missing links between specifications of different areas (e.g. JSDL and GLUE2) must be defined. The GIN group created a spin-off group named as Production Grid Infrastructure (PGI) working group with the particular goal to work on these improvements and to provide thus a production stable standards-based ecosystem named as the infrastructure interoperability reference model (IIRM) [14]. Members of the PGI working group represent a significant fraction of Grid middleware such as ARC [22], gLite [15],

978-1-4244-6534-7/10/$26.00 ©2010 IEEE

UNICORE [16], or Genesis that all play a major role in production Grids today (e.g. NorduGrid, EGEE, DEISA / PRACE, etc.).

While improvements of the OGSA-BES interface have been described in earlier work [17], we emphasize in this contribution on lessons learned from production Grid applications in the context of JSDL-based job descriptions and GLUE2-based information model dependencies.

The remainder of the paper is structured as follows. After the introduction, Section 2 provides details about rather general Grid application description improvements, while Section 3 gives insights into extensions required for effective executions on HPC-driven Grids. Section 4 provides an overview of the concepts related to application sequences and this paper ends in Section 5 with concluding remarks.

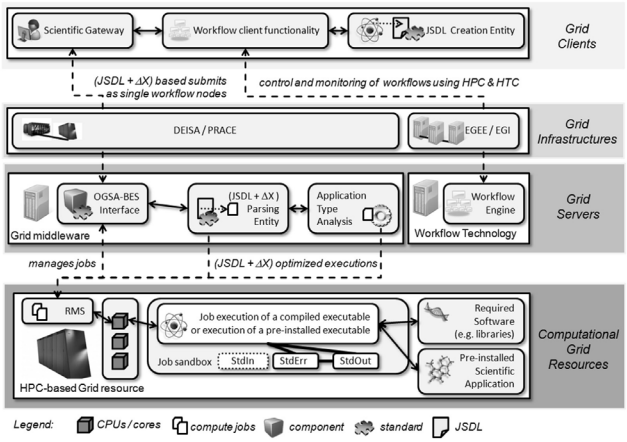


Figure 1. Concepts of Grid application description improvements in a broader context including its Grid middleware and Grid resource dependencies. The delta JSDL indicates our applied changes.

II. GRID APPLICATION IMPROVEMENTS

In this section we review how Grid application job descriptions can be improved, because we want to reveal more meaningful description elements in order to enable Grid middleware to execute Grid jobs and their described applications more effectively.

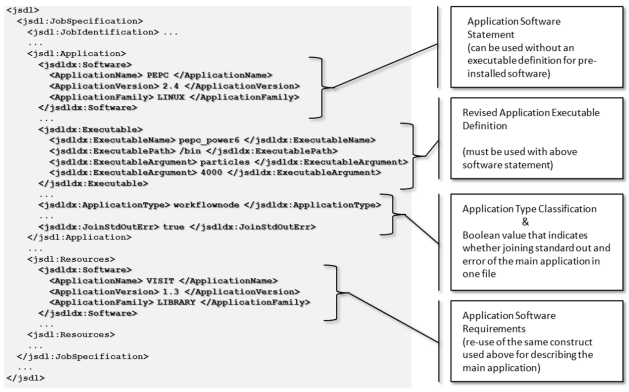
In using Grid middleware with cross-Grid applications we learned that Grid middleware can take advantage of several more detailed descriptions about the Grid job that is currently defined with standards like JSDL [9] or its extensions (i.e. JSDL SPMD Extension [18] and HPC Profile Application Extensions [19]). Therefore, this section describes a number of improvements to JSDL that are shown in a broader context in Figure 1 in order to understand where these improvements take effect.

In addition to the descriptions found in JSDL, the concept of (a) application types classification provides useful information about the Grid job that affects its handling in numerous different ways within Grid middlewares (e.g. parsing effectiveness). In short, this classification consists of the following enumeration *serial, collection, metaparallel, parallel,* and *workflownode*. The key benefit of all these pieces of information within the JSDL is to provide Grid middleware an exact classification that can be used to parse and process JSDL documents much more effectively. Of course, one Grid application can be described by more than one element of this classification.

The classification element *serial* stands for an individual stand-alone job while the *collection* element refers to a job that is submitted as part of a collection of individual jobs that do not communicate among each other. This is often used in HTC use cases or parameter sweep studies. In contrast the classification *metaparallel* refers to a job that is submitted as part of a collection of individual jobs that do communicate with each other via the mechanisms of metacomputing (i.e. Meta-MPI). We observed that this is rarely used, but some use cases (e.g. like in the VPH community [6]) are based on this mechanism crossing numerous production Grids. Much more often used is the *parallel* job classification that also refers to a job submitted that could be part of a larger collection, but the communication is basically performed in the job itself using parallel computing mechanisms (i.e. MPI or OpenMP). These jobs are often used in the context of HPC-driven e-science infrastructures such as DEISA or TeraGrid. Finally, jobs that are of the *workflownode* type are typically submitted as part of a larger cross-site workflow and handled by a workflow engine technology.

Especially in Grid interoperability setups, the definition of the main Grid application executable within the Grid job description represents a challenge since a wide variety of inhomogeneous systems are typically used with setups that are defined by our reference model. All previous concepts of the JSDL specification like its POSIX normative extension (i.e. POSIXApplication executable definition), which is also used in the JSDL SPMD extension, or its HPC Profile Application extension (i.e. HPCPAApplication executable definition) have only used an Executable element and an Argument element. But the lack of having the path information of an executable clearly separated from the executable sometimes caused trouble while parsing JSDL documents in cross-Grid infrastructure setups.

To overcome these limitation, we thus argue that it make sense to use a (b) revised application executable definition with the three elements *ExecutableName*, *ExecutablePath*, and *ExecutableArgument* while the latter element can appear n times in the job description. We explored that this concept enables the most flexible support in terms of supporting different varieties of job submission approaches that are (i) compile and execute applications in the job sandbox, (ii) pre-installed applications with a fixed and known path, and (iii) pre-installed applications that take advantage of complex constructs using environment variables (e.g. $PATH variable) and such like.



(i) Listing: Example of JSDL + ΔX instance with more meaningful Grid job descriptions

Figure 2. Design layout for the functionalities related to the Grid application description improvements of the JSDL language. This example shows the PEPC plasma physics application that uses the VISIT visualization library.

Another concept of JSDL that could be improved was the definition of the *ApplicationName* and *ApplicationVersion* information. We improve this concept by using the (c) application software statement concept that adds to these pieces of information also the *ApplicationFamily* (e.g. LINUX, WIN, Library, etc.). In addition, we re-use this concept to define a much clearer formulation of the (d) application software requirements concept. It is important to understand that one and only one instance of the application software statement concept describes the main Grid application itself in the Application part of the JSDL instance; while n other instances of it are used to describe application software requirements in the Resource part of the JSDL instance.

Figure 2 illustrated the re-use of the concept in a broader view using the example of an application that requires a dedicated visualization library called VISIT [20] during runtime for its execution on a HPC-based system. We also encountered many times the need to have a concept named as (e) application output joins, which refers to a boolean value indicating whether the standard-out and standard-error outputs of the main applications should be joined in one file or not. In fact, some applications use the standard-error for significant output (e.g. AMBER MD suite) and thus e-scientists are sometimes interested to have only one file for easier analyzing of the application run. Finally, we provide a summary of the proposed improvements to the JSDL standard in Table 1.

TABLE I.

Functionality Extensions and Improvement Concepts	Area	Extended Standard
(a) Application types classification	Compute	JSDL
(b) Revised application executable	Compute	JSDL
(c) Application software statement	Compute	JSDL
(d) Application software requirements	Compute	JSDL
(e) Application output joins	Compute	JSDL

III. HIGH PERFORMANCE COMPUTING EXTENSIONS

This section explores refinements of the JSDL language and GLUE2 in terms of large-scale HPC application support, because we want to find out which recent production HPC resource features are missing in JSDL and GLUE2 so that scientists can use it to submit and execute applications more efficiently and to obtain more accurate information.

Our experience from production Grid interoperability use cases clearly reveal that the support for HPC-based job application descriptions can be improved, especially when using resources available within HPC-driven infrastructures such as DEISA / PRACE or TeraGrid. Furthermore, we identified that pieces of information about resource features (i.e. available network topologies) must be exposed by Grid information systems in more detail. This was the case in use cases related to the VPH community, or more recently, with the members of the EUFORIA project as part of the fusion community.

Some core building blocks of our IIRM have been originally defined several years ago and thus they lack support of concepts of recently used Grid resources in general and large-scale HPC systems in particular. JSDL [9], for instance, was originally defined in 2005 and revised in 2008, but still lack the required functionalities to enable an efficient HPC oriented Grid job execution. In fact, JSDL extensions during 2007 such as the SPMD specification [18] or the HPC Profile Application extensions [19] aim at delivering some of these required functionalities, but also do not cover the essential functionality listed in this section.

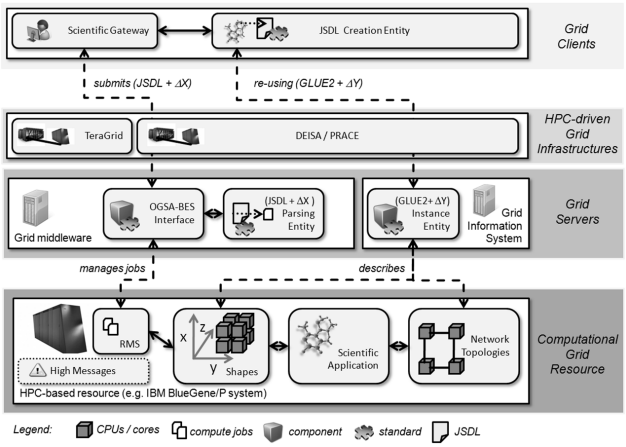


Figure 3. Growing complexity of HPC machines makes it necessary to expose more accurate information via GLUE2 and to enable a more precise job submission with JSDL. Delta X and delta Y indicate our applied changes.

One of the most significant extensions to JSDL is the support for different types of (a) network topologies. In fact, the choice of network connections can have a big

influence on the performance of applications that make use of parallel programming models (i.e. MPI). To provide an example, the state-of-the-art BlueGene/P HPC system as shown in Figure 3 offers different types of network connections: three dimensional torus, mesh, global tree (collective network), 10 Gigabit Ethernet (functional network).

Another extension to JSDL that is necessary to efficiently run parallel programming applications on recent HPC-based Grid resources is the (b) shape reservation functionality as supported by BlueGene/P systems today [21]. The optimal shape for an application depends on the communication pattern of the MPI-based parallel code and thus it is application-specific and in turn should be part of the application job description. Typically, a shape is indicated with x X y X z where x, y, and z are positive integers that indicate the number of partitions in the X-direction, Y-direction, and Z- direction of the requested job shape.

The above described extensions to JSDL are complementary also added as extensions to the GLUE2 standard. Although the GLUE2 NetworkInfo t data type [12] already described network information, these are limited to a few certain values and do not cover some technologies that offer torus networks or collective networks (e.g. global trees). Therefore we improve with some (c) network information enhancements the GLUE2 standard. In addition, we expose information about Grid resources more accurately and thus propose to add (d) available shape characteristics as extensions to the GLUE2 standard.

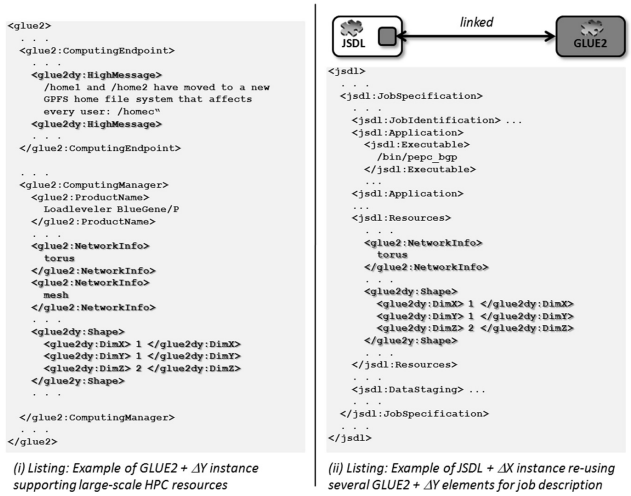


Figure 4. Design layout for the functionalities related to high performance computing extension concepts of JSDL and GLUE2 including their inter-dependencies.

In the context of the GLUE2-based description of Grid resources, we also identified the need to add so called important messages of the day (i.e. high messages) as extension to the GLUE2 standard. Here, GLUE2 provides attributes about DownTime information (i.e. endpoint entity) or a more general possibility pointing to certain information in the Web via the StatusInfo attribute (i.e. service entity), or another complete general OtherInfo attribute. Nevertheless, because of its general applicability and major importance, we add the (e) high message support to the GLUE2 standard. Some examples of these messages include temporary important information about file system usage (e.g. directory movements within PGFS) or about certain changes in complex compiler and application executions. Furthermore, high messages also cover pieces of information about local storage situation change (i.e. local storage cluster access) or other administrative pieces of information such as the transition period from one HPC-driven Grid resource to another. Hence, not always it is related to system downtime.

In order to take the baseline reference model design into account, we clearly identify that major parts of this functionality are part of a missing link between the specifications JSDL and GLUE2. We therefore define the following relationships between the above described functionalities:

concept (c) in GLUE2 ComputingManager Entity
implies
concept (a) in JSDL Resource Description

concept (d) in GLUE2 ComputingManager Entity
implies
concept (b) in JSDL Resource Description

In words, (c) implies (a) and (b) is a logical consequence of (d). The relationship is an implication since the antecedents (c) and (d) might be extended to a far deeper level of information than useful in JSDL elements for pure job description. In terms of our design model layout, we can thus define XML renderings of (c) and (d) within GLUE2 and re-use them within JSDL as shown in Figure 4. Finally, we provide a summary of the proposed improvements to the JSDL and GLUE2 standard in Figure 4 and in Table 2. Other more complex concepts such as the 'precise task/core mapping' descriptions also significantly improve the execution efficiency of the corresponding application. This concept and a few others more complex ones have been kept out due to readability of the paper and the page restriction.

TABLE II.

Functionality Extensions and Improvement Concepts	Area	Extended Standard
(a) Network topology (torus, global tree, etc.)	Compute	JSDL
(b) Shape reservation (x X y X z)	Compute	JSDL
(c) Network information enhancements	Info	GLUE2
(d) Available shape characteristics	Info	GLUE2
(e) High message support	Info	GLUE2

IV. SEQUENCE SUPPORT FOR COMPUTATIONAL JOBS

In this section we study the difference between Grid workflows and resource-oriented application sequences, because we want to find out how we can jointly support different types of application execution modes (i.e. serial, parallel) for one Grid application so that e-scientists can conveniently use reference model implementations that fit their needs in terms of remote compilation and pre- and post-processing functionalities.

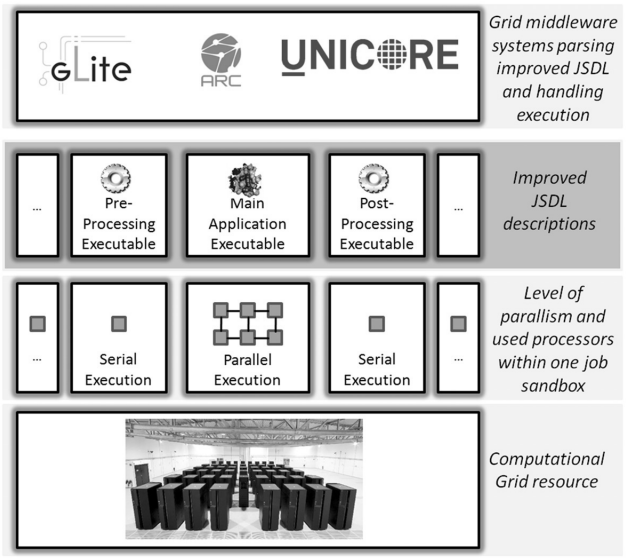


Figure 5. Support for application sequence executions within one Grid sandbox supporting multiple types of application execution modes (i.e. serial, parallel).

One specific missing feature encountered during production Grid interoperability is the support of pre- and post-processing functionalities within JSDL using different application execution modes. As shown in Figure 5, the HPC elements within the WIDOM workflow works with the molecular dynamics package AMBER that consists of a set of applications and some of them are used to transform input data in a suitable format for production runs. Of course, these transformation and short running pre-processing steps are typically executed in a serial mode, while the actual corresponding molecular dynamic simulation is executed in a parallel mode using many CPUs / cores.

Another example encountered in production Grids is the demand for remote compilation of source-code thus avoiding the need to login manually with SSH. In the context of EUFORIA, we encountered that most applications have to be installed beforehand on execution sites using SSH since a suitable support for remote compilation is missing in JSDL. This approach is feasible when the source-code of the application is rather stable, but the lack of remote compilation becomes even more a problem when source-code of Grid applications are subject to change as encountered in many different applications that make use of HPC-driven Grid infrastructures. For instance, we experience in the VPH community this problem with the source code Hemelb that must be re-compiled on each of the different Grid resources due to different hardware architectures.

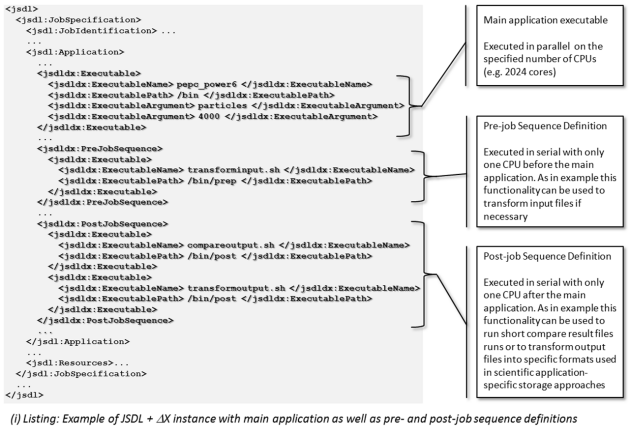


Figure 6. Design Layout for the functionalities related to the pre-job and post-job sequences.

In the context of the above described obstacles, we have to study what the difference between Grid workflows and sequences really are. In this context, it seems to make sense that compilation and execution are performed in one sandbox, otherwise the application might be installed beforehand. There is no exact boundary and you can realize it with both workflows and sequences, but when using sequences you can often circumvent data-transfers into the job sandbox. In addition, often the codes are specific for some types of architectures, which is in terms of many-core even more and more evolving thus requiring re-compilation anyway.

As a consequence, we extend JSDL with the capabilities to execute pre-job sequences (a) in order to enable the definition of n pre-processing applications that are serially executed before the main Grid job application. This also satisfies the demand for remote compilation since one or many of these pre-processing applications defined in the pre-job sequence can be represented by a compiler. In turn, this compilation sequence step is serially executed before the main compiled Grid job application is started.

In analogy to the pre-job sequences, our functionality extensions also cover post-job sequences (b) in order to support n post-processing applications. This sequence is only started when the main (often in parallel executed) Grid job application is finished.

Finally, we provide a summary of the proposed improvements to the JSDL standard in Figure 6 and Table 3.

TABLE III.

Functionality Extensions and Improvement Concepts	Area	Extended Standard
(a) Pre-job sequences (pre-processing)	Compute	JSDL
(b) Post-job sequences (post-processing)	Compute	JSDL

V. Execution Adjacencies Concept

The fundamental idea of execution adjacencies concept is to have a Grid middleware-independent common execution environment (CEE) that can be used by Grid applications during run-time. The realization of this concept includes two major aspects that are 'common environment variables' and a 'common execution module concept'. Although this sounds trivial, we observed that not a few applications actually fail on having different execution environments on different Grid resources, especially in cross-Grid use cases. Hence, we define a common execution environment that a Grid job can find on every Grid resource. As there is no particular standard in this field, we propose this additional standard in the context of the larger computational-driven standard ecosystem that consists of OGSA-BES and JSDL (including its numerous extensions).

In more detail, the first aspects of the execution adjacencies are simply realizable using (a) common environment variables across different middleware distributions. In several applications, we observed that the actually running source-code makes use of environment variables such as number of cores, available memory and such like. So far, every Grid middleware such as gLite, ARC, or UNICORE provided such pieces of information via environment variables in rather proprietary execution environments (i.e. no common syntax or semantics). Therefore, we propose a standardized list of environment variables by not only defining their precise syntax, but also the corresponding semantics that lead to a significant advantage in interoperability setups.

TABLE IV.

Environment Variable Syntax	Environment Variable Semantics
GLUE2: MainMemorySize	The total amount of physical RAM
GLUE2: PhysicalCPUs	The number of physical CPUs
ExtensionGLUE2: PhysicalCores	The number of physical cores
…	*Others from the GLUE2 Execution Environment attribute specification*

As an example, a few of the environment variables can be found in Table 4, which content is self-explaining and not repeated within the text. Many of these variables provide information that must be consistent with information provided by a Grid information system for each Grid resource (i.e. by using GLUE2 schema elements). Also, it is important to agree that all this information must be provided. When some environment variables will have no information then it is not worth using them at all since applications will expect to get the information from that variable at a point in time. A closer look within GLUE2, for instance, reveals the definition of attributes in the so-called *'ExecutionEnvironment'*. We argue that it make sense to basically 'render' those attributes as environment variables within the Grid middleware. Also, in this context we require to add a few attributes to the execution environment that are useful for applications during run-time. To provide an example, although GLUE2 defines the amount of physical CPUs we also argue that it make sense to provide the amount of physical cores to address the different core setups (single-core, dual-core, quad-core, upcoming n-core, etc.) on computing resources nowadays.

Closely related to this first aspect is also the second aspect of that we call the (b) 'common execution module' concept that originates from our work with applications that require a pre-defined setup of not only environment variables, but also path settings and such like. Hence, it is not about pieces of information about a pre-installed application setup (cf. Table I) and instead works on a far deeper level that of the application itself. To provide an example, in the DEISA infrastructure, we defined a so-called 'AMBER module' that includes the setup of all necessary pieces of information to run the AMBER scientific package including roughly 50-80 executables and programs. To avoid that scientists always have to setup the required details like (PATH and execution locations, AMBER environment variables, versioning, etc.), they simply use 'module load AMBER' before any production run with AMBER. We argue that this concept is very beneficial and thus seek to integrate it smoothly in the context with GLUE2 and JSDL and propose a new small standard for this. Nevertheless, partly parts of the GLUE2 specification in the context of so-called *'ApplicationEnvironments'* are re-used as well.

Finally, we provide a summary of the proposed functionality and concepts to improve the execution adjacencies of scientific application runs in Table 5.

TABLE V.

Functionality and Proposed Concepts	Area	New Standard
(a) Common Environment Variables	Compute	CEE
(b) Common Execution Modules	Compute	CEE

VI. Related Work

Related work in the field of standardization is clearly found among the members of the JSDL and GLUE2 OGF working groups. Several other ideas and concepts arise also

from the work of these members and we have discussed and will discuss in future of how we can align our work in order to have new set of specifications that not fundamentally change the existing specifications and thus just improving them without break their emerging stability.

Related Work in the field of reference models typically leads to the Open Grid Services Architecture (OGSA), which has in comparison to our approach a much bigger scope. Hence, our approach only represent a subset of this scope but more focused and thus more detailed. We deliver with our reference model a much more detailed approach of how open standards can be improved and used in scientific applications that require interoperability of e-science production Grids today. Neither this contribution nor the reference model in the bigger context aim at replacing OGSA and thus rather represent a medium-term milestone towards a full OGSA compliance of Grid middlewares in future. In comparison with the former commercially-driven Enterprise Grid Alliance Reference model, our model is clearly oriented to support rather scientific-based use cases.

VII. CONCLUSIONS

We have shown how the common open Grid standards JSDL and GLUE2 can be significantly improved to integrate lessons learned gained by an academic analysis of the production Grid interoperability experience over years. Because of the page restriction for this focused workshop, we only present a few focused concepts of our infrastructure interoperability reference model and kept many important aspects of it (e.g. security aspects) out and refer to other publications mentioned in the introduction. In having the chair position in the OGF Grid Interoperation Now (GIN) community group and Production Grid Infrastructure (PGI) working group we are driving the standardization of the concepts described in this very focused contribution as best as possible being still open for other required concepts that arise from other production Grid applications.

Finally, we will demonstrate our UNICORE-based reference implementation of the above described concepts and other concepts at the interoperability day at the next OGF28 in Munich.

ACKNOWLEDGMENT

We thank the OGF GIN and PGI groups for fruitful discussions in the context of this work. This work is partly funded via the DEISA-II project funded by the EC in FP7 under grant agreement RI-222919.

REFERENCES

[1] P. Eerola et al., "Building a Production Grid in Scandinavia", in IEEE Internet Computing, 2003, vol.7, issue 4, pp.27-35

[2] EGEE. [Online]. Available: http://public.eu-egee.org/

[3] DEISA. [Online]. Available: http://www.deisa.org

[4] M. Riedel *et al.*, "Improving e-Science with Interoperability of the e-Infrastructures EGEE and DEISA," in *Proceedings of the MIPRO*, 2007.

[5] EUFORIA. [Online]. Available: www.euforia-project.eu/

[6] Website, "Virtual Physiological Human (VPH)," http://www.europhysiome.org.

[7] M. Riedel, E. Laure, *et al.*, "Interoperation of World-Wide Production e-Science Infrastructures," in *Journal on Concurrency and Comp.: Practice and Experience*, 2008.

[8] I. Foster *et al.*, *OGSA Basic Execution Service Version 1.0*. Open Grid Forum Grid Final Document Nr. 108, 2007.

[9] A. Anjomshoaa *et al.*, *Job Submission Description Language Specification V.1.0*. OGF (GFD56), 2005.

[10] A. Sim *et al.*, *The Storage Resource Manager Interface Specification Version 2.2*. OGF Grid Final Document Nr. 129, 2008.

[11] I. Mandrichenko *et al.*, *GridFTP v2 Protocol Description*. OGF Grid Final Document Nr. 47, 2005.

[12] S. Andreozzi *et al.*, *GLUE Specification 2.0*. OGF Grid Final Document Nr. 147, 2009.

[13] A. Andrieux *et al.*, *Web Services Agreement Specification (WS-Agreement)*. OGF Grid Final Document Nr. 107, 2007.

[14] M. Riedel *et al.*, "Research Advances by using Interoperable e-Science Infrastructures - The Infrastructure Interoperability Reference Model applied in e-Science," in *Journal of Cluster Computing, SI Recent Research Advances in e-Science*, 2009.

[15] E. Laure *et al.*, "Programming the Grid with gLite," in *Computational Methods in Science and Technology*, 2006, pp. 33–46.

[16] A. Streit *et al.*, "UNICORE - From Project Results to Production Grids." in *Grid Computing: The New Frontiers of High Performance Processing, Advances in Parallel Computing 14*, L. Grandinetti, Ed. Elsevier, pp. 357–376.

[17] M.Riedel *et al.*, "Concepts and Design of an Interoperability Reference Model for Scientific- and Grid Computing Infrastructures," in *Proceedings of the Applied Computing Conference, in Mathematical Methods and Applied Computing, Athens, Volume II, WSEAS Press 2009, ISBN 978-960-474-124-3, Pages 691 - 698*, 2009.

[18] A. Savva, *SPMD JSDL Extension*. OGF Grid Final Document Nr. 115, 2007.

[19] M. Humphrey *et al.*, *HPC Basic Profile JSDL Extension*. OGF Grid Final Document Nr. 111, 2007.

[20] T. Eickermann *et al.*, "Steering UNICORE Applications with VISIT," vol. 363, pp. 1855–1865, 2005.

[21] I. Website, "IBM BlueGene/P Technology," http://www.ibm.org/.

[22] M. Ellert et al., Advanced Resource Connector middleware for lightweight computational Grids", Future Generation Computer Systems 23 (2007) 219-240.