Recent ARC developments: Through modularity to interoperability

This article has been downloaded from IOPscience. Please scroll down to see the full text article.
2010 J. Phys.: Conf. Ser. 219 062027
(http://iopscience.iop.org/1742-6596/219/6/062027)

View the table of contents for this issue, or go to the journal homepage for more

Download details:
IP Address: 130.235.185.191
The article was downloaded on 17/05/2011 at 10:43

Please note that terms and conditions apply.
Recent ARC developments: through modularity to interoperability

O Smirnova\(^1\,2\), D Cameron\(^1\,3\), P Dóbé\(^2\,4\), M Ellert\(^1\,5\), T Frágát\(^3\), M Grønager\(^1\), D Johansson\(^1\,6\), J Jönemo\(^2\,5\), J Kleist\(^1\,7\), M Kočan\(^8\), A Konstantinov\(^3\,9\), B Kónya\(^2\), I Mártón\(^4\), S Möller\(^10\), B Mohn\(^5\), Zs Nagy\(^4\), J K Nilsen\(^1\), F Ould Saada\(^3\), W Qiang\(^3\), A Read\(^3\), P Rosendahl\(^1\), G Rőczei\(^4\), M Savko\(^8\), M Skou Andersen\(^11\), P Stefán\(^4\), F Szalai\(^3\), A Taga\(^3\), S Z Toor\(^12\) and A Wäänänen\(^11\)

\(^1\) NDGF, Kastruplundsgade 22, DK-2770 Kastrup, Denmark
\(^2\) Lund University, Experimental High Energy Physics, Institute of Physics, Box 118, SE-22100 Lund, Sweden
\(^3\) University of Oslo, Dept. of Physics, P. O. Box 1048, Blindern, N-0316 Oslo, Norway
\(^4\) NIIF/HUNGARNET, Victor Hugo 18-22, H-1132 Budapest, Hungary
\(^5\) Uppsala University, Dept. of Physics and Astronomy, Div. of Nuclear and Particle Physics, Box 535, SE-75121 Uppsala, Sweden
\(^6\) Linköping University, National Supercomputer Centre, SE-581 83 Linköping, Sweden
\(^7\) Aalborg University, Dept. of Computer Science, Frederik Bajersvej 7E, DK-9220 Aalborg Ø, Denmark
\(^8\) Pavol Jozef Šafárik University, Faculty of Science, Jesenná 5, SK-04000 Košice, Slovak Republic
\(^9\) Vilnius University, Institute of Material Science and Applied Research, Saulėtekio al. 9, Vilnius 2040, Lithuania
\(^10\) University of Lübeck, Inst. Of Neuro- and Bioinformatics, Ratzeburger Allee 160, D-23538 Lübeck, Germany
\(^11\) University of Copenhagen, NBI, Blegdamsvej 17, DK-2100 Copenhagen Ø, Denmark
\(^12\) Uppsala University, Dept. of Information Technology, Box 337, SE-75105 Uppsala, Sweden

E-mail: oxana.smirnova@hep.lu.se

Abstract. The Advanced Resource Connector (ARC) middleware introduced by NorduGrid is one of the basic Grid solutions used by scientists worldwide. While being well-proven in daily use by a wide variety of scientific applications at large-scale infrastructures like the Nordic DataGrid Facility (NDGF) and smaller scale projects, production ARC of today is still largely based on conventional Grid technologies and custom interfaces introduced a decade ago. In order to guarantee sustainability, true cross-system portability and standards-compliance based interoperability, the ARC community undertakes a massive effort of implementing modular Web Service (WS) approach into the middleware. With support from the EU KnowARC project, new components were introduced and the existing key ARC services got extended with WS technology based standard-compliant interfaces following a service-oriented architecture. Such components include the hosting environment framework, the resource-coupled execution service, the re-engineered client library, the self-healing storage solution and the peer-to-peer information system, to name a few. Gradual introduction of these new services and client tools into the production middleware releases is carried out together with NDGF and thus ensures a smooth transition to the next generation Grid middleware. Standard interfaces and modularity of the new component design are essential for ARC contributions to the planned Universal Middleware Distribution of the European Grid Initiative.
1. Introduction
The ARC Grid middleware [1] originally started as a rather monolithic solution meeting requirements of the High Energy Physics community and heavily based on the pre-Web Service Globus Toolkit [2] libraries and API. Since this was a common practice for Grid projects at that time, several Globus technologies got effectively promoted to de facto standards supporting basic interoperability, most notably, in the security area. Later on, different projects developed proprietary approaches in order to meet various performance and operational requirements. Grid middleware solutions quickly diverged, to the point of being incompatible with each other and industry standards [3].

In recent years, a strong drive towards wide interoperability emerged in the Grid community [4]. The Open Grid Forum [5] became an efficient standards development organization, delivering specifications that can be used as a basis for standard-conforming and thus interoperable Grid solutions. Modular Web Service based approach to Grid components was introduced in Globus and many other middlewares, further widening interoperability opportunities. At the same time, transition to new technologies presents Grid developers with new challenges. ARC developers were among the first to start implementing new, often not yet fully matured standards. The main challenge on this path is two-fold: how to transform the middleware without undermining what has been achieved in terms of performance and usability and how to be able to continuously adapt to the changing specifications. In what follows, we will present a brief summary of the existing ARC features and illustrate its performance using the example of the Nordic DataGrid Facility [6], give an overview of the relevant standards, and proceed to the description of the new ARC components being developed by the EU KnowARC project [7].

2. ARC features and performance
This section gives a high-level overview of the current production ARC middleware; detailed description is available elsewhere [1]. ARC is a comparatively light-weight middleware, optimized for serial data-intensive computational tasks. It implements a service-based architecture with well defined though sometimes custom interfaces. The key design concept is the absence of a single point of failure. Combined with a stateful implementation of services, this provides a reliable and thus efficient system, as proven by many years of contribution to LHC computing [8, 9]. Figure 1 shows higher than average percentage of utilization of computing resources pledged for ATLAS production by the ARC-enabled NDGF, fully attributed to high middleware efficiency.

![Figure 1. Utilization of computing resources pledged for ATLAS production by the involved Tier-1 centers in 2008. Arrow indicates NDGF. PIC and TRIUMF sites acquired more physical resources in course of the year than originally pledged.](image)
In data-intensive HEP computing, the leading source of failures and inefficiencies is data management [10]. ARC is optimized for this kind of high-throughput distributed computing by moving input and output data handling into computing service plugins executed on the front-end. This increases CPU utilization, allows for data caching, enables optimization of bandwidth usage through configuration tuning, and minimizes the risk of accidental distributed denial of service attacks.

ARC services deploy as a comparatively thin layer, with software installation limited to the front-end, allowing to keep compute nodes off-line. Computing service includes support for retries of transfer failures, transparent downtime handling, support for memory and CPU time limits, and all configuration is done through only one file. ARC is interfaced to batch systems via a plugin framework and currently supports seven batch systems plus a simple fork process launch.

ARC implements a dynamic hierarchial multi-rooted distributed information system that holds information not only about computing and storage resources, but also about individual jobs. All such information is produced and kept locally at each service instance. This allows to encapsulate resource discovery, matchmaking, task scheduling and status monitoring in a client, without the need of an intermediary brokering or bookkeeping service.

The standard command line interface (CLI) of ARC integrates all the client functionality necessary for basic job and data management, including resource discovery and brokering. The CLI tools are implemented on top of a public API; this API facilitates development of application-specific clients, portals or even workload or workflow management systems, and a number of such exist – for example, the Lunarc Application Portal [11].

Since ARC user community relies on a wide variety of operating systems, the code has to be very portable and contain minimal number of external dependencies. Globus libraries (mostly those providing the GSI [12] security layer and GridFTP [13] functionality) constitute the bulk of such external dependencies. Thanks to the joint effort of KnowARC and NDGF projects, all the necessary external software is now available through official Fedora and Debian repositories, and ARC itself will soon follow the suit.

3. Grid standards
As follows from the previous section, ARC efficiently implements many core Grid functionalities. Naturally, there are bugs to fix and more features to implement, but in general there is no pressing need for dramatic changes. That is, as long as ARC occupies its own niche and does not interact with other solutions.

However, one must always keep in mind that the key attractive point of the Grid concept is that of resource sharing. This is particularly important for the distributed collaborative HEP computing. Non-interoperable middleware solutions create an unnecessary overhead, forcing users to create higher-level application-specific tools.

In general, when different applications need to use different infrastructures, standards must be in place. Like in any other infrastructure, such standards must cover all possible aspects: from basic interfaces and schemas, to access control, to service level agreements and policies.

When HEP-driven projects all over the world started developing Grid solutions, there were no standards in place, only a couple of reference implementations. By now HEP computing makes use of three major middlewares, plus a large set of experiment-specific solutions bordering the line between application software and middleware. Absence of standards creates enormous overhead and at times leads to bottlenecks, especially when the documentation of proprietary interfaces and schemas is scarce.

It is practically impossible to abandon all the developed tools or to re-write them in a standard-conforming manner. ARC strategy is to add standard interfaces to existing services, thus preserving the architecture and the underlying functionality. When other middleware
developers will follow the same path, the community will finally get the pervasive interoperable Grid.

As was mentioned earlier, Grid-specific standards are developed in the framework of OGF by a large number of working groups representing all middleware developers.

Some OGF standards are already implemented and used in production by ARC and other middlewares. Most notably, these are the ones pertaining to data transfer and management: the GSI-enabled file transfer protocol GridFTP and the Storage Resource Manager (SRM) interface [14]. This in practice creates the basis for the current LHC computing, allowing for seamless cross-infrastructure data movement.

Most relevant standards in the job description and execution domain are the Job Submission Description Language (JSDL) [15] and the Basic Execution Service (BES) interface [16]. Although being final specifications, they are not suitable for production environments and thus early implementations of these specifications are not widely deployed in HEP computing. One of the obstacles is the inherently generic nature of these standards, which means that real-life solutions must add numerous extensions. There is on-going work in the OGF’s PGI working group driven by ARC, gLite [17] and UNICORE [18] consortia to agree on a common set of extensions and modifications defined via a production profile. JSDL is already supported by ARC, and BES interface is available in the new components – both with the necessary ARC-specific extensions.

For the information and resource discovery domain, a new Glue2 [19] specification has been recently released. New components of ARC come compliant with this standard, and there is commitment from the gLite developers to move to Glue2 in near future as well.

Some standards are still sorely missing; this is particularly true for those related to access control, accounting and user management. Middlewares deployed in HEP computing, including ARC, make use of de facto standards like GSI and VOMS [20], which effectively provides another solid basis for current interoperation. The work of OGF and its PGI group in particular will hopefully fill this gap soon.

4. New components of ARC

The ongoing development of the ARC middleware is driven by the following considerations:

- The ARC server-side components should exhibit standard-based, well-documented open interfaces. The custom interfaces are being replaced by community-embraced ones (see Sections 4.2, 4.4 and 4.5).
- In order to facilitate the rapid development of new components (both on the service and the client side) a modular and developer-friendly framework should be implemented (Sections 4.1 and 4.3). The new framework must offer easy extensibility.
- The ARC code base should carefully select and isolate 3rd party dependencies into optional and modular plugins. In particular, the dependency on the legacy Globus libraries should be as minimal as possible. A clean code base is particularly important due to our portability goal of ARC being available on MacOS and MS Windows platforms in addition to the already supported numerous Linux flavors.
- ARC should come with rich and self-sufficient services. The service development framework should facilitate easy creation of intelligent and powerful services such as the A-REX (Section 4.2) and the storage solution (Section 4.5).
- Last but not least the increased modularity and the introduction of new interfaces should not result in decreased performance or usability.

In the following sections, overview of key new ARC components will be given.
4.1. Hosting Environment Daemon

The cornerstone of the new approach is a light-weight Web Service container called the Hosting Environment Daemon (HED), which provides hosting environment for various services, as well as a number of modules to support flexible, interoperable, and efficient communication mechanism for building SOAP-based Web Services. The design of the HED is built around the idea of flexibility and modularity, such that the service developers can simply concentrate on the application level Web Service implementation by only using the core minimum amount of components. It also simplifies work on the middleware level: for example, it makes possible to implement another communication protocol or authentication mechanism. Meanwhile, a service administrator can easily configure and deploy the middleware and application-specific services satisfying a variety of requirements without having to know much about the implementation.

The architecture of HED is illustrated in Figure 2. The key components of it are the so-called Message Chain Components (MCC) which are in charge of implementing different protocol levels. For example, as shown in the message flow, the HTTP MCC will process a stream from the TLS MCC to parse the HTTP message and pass its body to the SOAP MCC, and also process the SOAP response from the SOAP MCC to generate the HTTP message for the TLS MCC.

The dotted line in Figure 2 shows an alternative path for the information propagation between MCCs. A service administrator can configure the MCCs according to the interoperability requirements with a counterpart. For instance, the configuration shown with the dotted line is compatible with the WSE (Web Services Enhancement for .NET) SOAP message mechanism. Another configuration could be SOAP over HTTPG (HTTP over GSI) which is needed to interoperate with services like the Storage Resource Manager (SRM). This approach ensures flexibility of HED in terms of protocols support.

HED contains a flexible security framework for implementing and enforcing security-related functionality, such as authentication and authorization. Each security-related functionality can be implemented as a pluggable and configurable component (plugin) called SecHandler. Each MCC or service is usually configured with two queues of SecHandler – one for the incoming messages and another for outgoing ones. In Figure 2, the “AuthZ” and “AuthN” sub-modules inside MCCs and services are examples of SecHandlers.

HED is often perceived as yet another Web Services development framework; in reality, the purpose of HED is not to re-implement Axis, Tomcat or gSOAP, but simply to provide framework for gluing various functionalities, including other service container solutions via plugins if necessary. In the current implementation there are no Apache or Axis plugins, because the developers of HED were focused on making the solution lightweight and implemented only bare essentials like SOAP and HTTP. This is currently sufficient for our purposes and does not require any other traditional Web Service hosting environment.

4.2. Execution service with a standard-compliant interface

The ARC Resource-coupled EXecution service (A-REX) is the next generation computing element of ARC offering WS-interfaces and advanced security solutions [21]. The powerful computing element implements job execution capability over a large variety of computational resources. A-REX is built around the Grid Manager component of the production ARC computing element. A-REX interprets standard job description (JSDL, with NorduGrid extensions), offers OGF-compliant job execution and management interface (BES, also with NorduGrid extensions), features transparent, automatic and integrated data staging and caching capability, support for large number of batch systems, session directory management, comes with logging capability and support for Runtime Environments. A-REX offers Web Service-based local information interface serving Glue2 information. A-REX is also capable of working together with community approved security frameworks.
4.3. Interoperable client

The client tools for job and data management are based on libraries implemented in C++. These libraries are plugin based, and adding support for a new Grid job execution service flavor or a new data access protocol can be done by developing a new plugin. The main libraries and the client tools can take advantage of the new functionality provided by the new plugin without having to be re-compiled or re-linked.

The client libraries have been wrapped using SWIG to create language bindings for Python and Java, thereby making it easy to integrate the Grid client functionality provided by the libraries in application frameworks based on these languages. Python bindings are used in the implementation of the self-healing storage solution (see Section 4.5) as well as in the Lunarc Application Portal [11]. Java bindings are not yet widely used. So far no major obstacle caused by the wrapper approach has been met.

The client libraries implement uniform handling of user and host credentials, computing resource discovery and information retrieval as well as matchmaking and brokering, job submission and input/output data handling.

Plugins for job handling for several Grid job execution services are already available, including the ARC Grid Manager, A-REX and the gLite CREAM service. Support for UNICORE execution services is currently being developed.

The client library supports several job description languages, including the extended resource description language (xRSL) [22] used by the ARC Grid Manager, the JDL used by e.g. CREAM, and the standard JSDL. The library is capable of converting between these descriptions as needed.

Figure 2. Example of the Hosting Environment Daemon deployed with job execution services
to interoperable with different services. Not all job description elements can be matched one-to-one between the languages, which is one of the key factors motivating work on a common standard JSDL profile.

Plugins for data management exist for most data management protocols currently in use by different Grid projects, including HTTP (including HTTPS and HTTPg), GridFTP and SRM as well as support for several replica location catalogues such as the Globus RLS and the gLite LFC. There is also support for the new ARC self-healing storage solution.

Also the brokering used in the client is based on plugins and users can write their own brokering modules using either C++ or Python.

4.4. P2P information system backbone

The operation of Grid infrastructures is based on co-operation of services. The information system holds any Grid system together and provides a way for the other participants to find each other: the information system is the backbone of any Grid. Its special role requires high redundancy. To adhere to this requirement the information should be stored over the network in multiple places. The next generation information indexing backbone of ARC is being implemented as a P2P network of ISIS components.

ISIS as the building block of the P2P information cloud collects information from other registered services (and also from other ISIS services), stores these in its local database and then redistributes them towards its neighbors in the network in a reliable manner.

An ISIS instance stores the so-called Registration Entries submitted by the services. The maintained database of these entries can be queried by XQuery 1.0 and XPath 2.0 [23] expressions through a custom Web Service interface. The same interface is used to insert records into the ISIS database. The ISIS database itself is a soft-state database, which allows its entries to be up-to-date even in case of lost network connection.

While ARC services can register information to ISIS services and these records can be queried by clients, ISIS services cannot discover one another following this way. ISIS solves this issue by using P2P technology to build a cloud of information systems and to balance the load among them. To join the P2P cloud, an ISIS service only needs to know the address of some already connected ISIS and then it can connect the existing network by simply synchronizing its local database. The databases are kept synchronized by sending modification messages to other ISIS services in multiple paths.

ISIS is implemented as a service within the HED, thus all the WS related communications are performed by the hosting environment framework. Further advantages of using the HED are the flexible and uniform configuration possibility, ready-to-use security framework and the built-in self-registration mechanism.

Though ISIS as such is largely information schema agnostic, it has to be mentioned that ARC information system follows the Glue2 specification.

The current ISIS implementation lacks proper P2P-like authorization framework: the system is either configured as an open network or controlled via a central authorization service.

4.5. Self-healing storage solution

An ever increasing number of Grid applications demand not only increased CPU power, but also vast amounts of storage space. Nowadays, single Grid jobs can easily produce gigabytes or even terabytes of data, thus ramping up the requirements of storage systems to the petabyte-scale and beyond.

A Grid storage system is a set of services providing access to data stored at distributed facilities. A traditional architecture used by infrastructures like NDGF consists of an indexing service, indexing files from storage resources, a replication service for managing replica locations, and occasionally a (often centralized) metadata catalog imposing a global namespace on top of
the resources. Data access can be handled either through a file transfer service or directly through the storage resource by querying the metadata catalog. For all these services, third-party solutions were used so far, with ARC providing only client-side utilities. While this architecture has strongly improved the accessibility of physically distributed data, the following challenges still remain:

- A centralized metadata catalog can quickly become a bottleneck.
- When storage resources are unaware of the state of their files, consistency throughout the system can be hampered.
- It is often not straightforward to plug in an arbitrary storage resource, which may limit the amount of available hardware resources and introduce an overhead of data migration.
- Maintenance and operation of complex distributed storage systems can be rather resource-consuming.

In order to address the above mentioned issues the KnowARC project set forth to develop a distributed by design, self-healing and flexible Grid storage solution which has a replicated metadata catalog, state-aware storage resources and an operating system agnostic implementation. The developed system consists of a set of SOAP based services residing within HED, which together provide a robust, scalable, and consistent data storage system. Data is managed in a hierarchical global namespace with files and subcollections grouped into collections. A dedicated root collection serves as a reference point for accessing the global namespace, making it possible to reference the hierarchy using logical names. The global namespace is accessed in the same manner as in local filesystems.

![Diagram](image.png)

**Figure 3.** High-level overview of the ARC self-healing storage solution

Being based on a service-oriented architecture, the developed system consists of a set of services as shown in Figure 3. The services are as follows: The Bartender (B) provides the high-level interface to the user and offers the possibility to connect to third-party storage solutions; the Librarian (L) handles the entire storage namespace, using the A-Hash (A-H) as a metadatabase; and the Shepherd (S) is the frontend for the physical storage node.

As is the case for all openly accessible Web Services, the security model is of crucial importance for the developed system. The security architecture of the storage system can be

---

1 A concept very similar to files and directories in most common file systems.
split into three parts: the inter-service authorization which ensures that only trusted services can get or modify data; the transfer-level authorization which ensures secure file transfers; and the high-level authorization which handles user’s permissions to modify files and collections, e.g. read and write access to files and collections.

Three different client tools, serving different needs, have been developed for the storage system. A prototype command line tool offers the end user all the storage system functionality from a terminal window. However, end users frequently interacting with the storage system may want to an even easier tool than the command line tool, and for this purpose a FUSE module has been developed. The FUSE module allows the user to mount the namespace of the system into her local filesystem, and use the storage system as if it was a local directory. It uses the same libraries and transfer methods as the command line tool and preliminary tests have shown no significant difference in performance when comparing the two. Finally, a storage system plugin (so-called Data Management Component) for the ARC data libraries has been developed. The plugin enables transparent, integrated access to the storage system from the ARC middleware thus making it possible to run Grid jobs which both down and uploads data from the developed storage system.

The ARC storage solution does not require standards like SRM or GridFTP, and by default relies on standard HTTP and TLS; however, the fact that the services reside in HED means that e.g. an SRM interface can be easily added whenever necessary.

5. Conclusion: Towards a common middleware distribution
Recent ARC developments transformed the middleware from a rather monolithic solution based on old and sometimes obsolete technologies to a modern modular software that meets many industry and community standards. Decreasing dependency on third-party technologies and increasing reliance on commonly used ones allows ARC to achieve better interoperability with other related solutions. Driven by the community requirements and supported by the maturing open Grid standards, a growing number of middleware providers develop similar modular standard-compliant solutions consisting of interchangeable and complementary components. This is a welcoming sign for resource owners who would like to join large Grid infrastructures and yet retain the freedom of technology choice. The European Grid Initiative (EGI) [24] project strives to achieve exactly this: a pan-European Grid infrastructure for research communities. Such an environment has to rely on a set of agreed middleware tools and utilities that satisfy common criteria and meet common standards. This set currently goes by the name Universal Middleware Distribution (UMD) and will be based on middleware components provided, among others, by ARC, gLite and UNICORE consortia. Standard-based interoperability between the UMD components is the principal criterion in the selection process. There is a strong confidence that this middleware suite is achievable, not least because of the commitment of the core consortia to the OGF standardization process. With the re-designed and newly introduced components, ARC is well on track to offer its traditionally reliable and efficient Grid tools and services conforming to strictest interoperability and standardization requirements.

Acknowledgments
This work was supported in part by the Information Society and Technologies Activity of the European Commission through the work of the KnowARC project (Contract No.: 032691).

References
[1] Ellert M et al. 2007 Future Gener. Comput. Syst. 23 219–240 ISSN 0167-739X
[2] Foster I and Kesselman C 1997 International Journal of Supercomputer Applications 11 115–128 available at: http://www.globus.org
[3] Field L and Schulz M 2008 Proc. of CHEP 2007, J. Phys.: Conf. Ser. 119 012001 ed R Sobie, R Tafirout and J Thomson (IOP) URL http://dx.doi.org/10.1088/1742-6596/119/1/012001
[4] Riedel M et al. 2009 Journal of Concurrency and Computation: Practice and Experience 21 961–990
[5] Open Grid Forum Web site URL http://www.ogf.org/
[6] Nordic DataGrid Facility Web site URL http://www.ndgf.org
[7] EU KnowARC project Web site URL http://www.knowarc.eu
[8] Eerola P et al. 2003 Proc. of CHEP 2003, eConf C0303241:MOcT011
[9] Sturrock R et al. 2005 Proc. of CHEP 2004, CERN-2005-002 vol 2 ed A Aimar, J Harvey and N Knoors p 1095
[10] J Andreeva, S Campana, F Fanzago and J Herrala 2008 Journal of Grid Computing 6 3–13
[11] Lunarc Application Portal Sourceforge site URL http://laportal.sourceforge.net/
[12] Foster I et al. 1998 CCS ’98: Proceedings of the 5th ACM conference on Computer and communications security (ACM Press) pp 83–92 ISBN 1-58113-007-4
[13] Allcock W et al. 2002 Parallel Comput. 28 749–771 ISSN 0167-8191
[14] A Sim, A Shoshani and others 2008 The Storage Resource Manager Interface (SRM) Specification v2.2 GFD-R-P.129 URL http://www.ggf.org/documents/GFD.129.pdf
[15] Anjomshoaa A et al. 2008 Job Submission Description Language (JSDL) Specification, Version 1.0 (first errata update) GFD-R.136 URL http://www.gridforum.org/documents/GFD.136.pdf
[16] Foster I et al. 2007 OGSA™ Basic Execution Service Version 1.0 GFD-R-P.108 URL http://www.ogf.org/documents/GFD.108.pdf
[17] gLite. Lightweight Middleware for Grid Computing Web site URL http://glite.web.cern.ch/glite/
[18] UNICORE, Uniform Interface to Computing Resources Web site URL http://www.unicore.eu
[19] Andreozzi S et al. 2009 GLUE Specification v2.0 GFD-R-P.147 URL http://www.ogf.org/documents/GFD.147.pdf
[20] Alfieri R et al. 2005 Future Gener. Comput. Syst. 21 549–558 ISSN 0167-739X
[21] Konstantinov A The ARC Computational Job Management Module - A-REX NORDUGRID-TECH-14 URL http://www.nordugrid.org/documents/a-rex.pdf
[22] Smirnova O Extended Resource Specification Language The NorduGrid Collaboration NORDUGRID-MANUAL-4 URL http://www.nordugrid.org/documents/xrsl.pdf
[23] 2007 XQuery 1.0: An XML Query Language W3C Recommendation URL http://www.w3.org/TR/xquery/
[24] European Grid Initiative Web site URL http://www.eu-egi.eu/