Tracing Resource Usage Over Heterogeneous Grid Platforms: A Prototype RUS Interface for DGAS

Rosario M. Piro[1,*], Michele Pace[2,*], Antonia Ghiselli[2], Andrea Guarise[1], Eleonora Luppi[3], Giuseppe Patania[1], Luca Tomassetti[3], Albert Werbrouck[1]

*These authors have contributed equally to the presented work.

[1]INFN Torino
Via Pietro Giuria, 1
10125 Torino, Italy
piro@to.infn.it

[2]INFN-CNAF
Viale Berti Pichat, 6/2
40127 Bologna, Italy
michele.pace@cnaf.infn.it

[3]INFN Ferrara
Via Saragat, 1
44100 Ferrara, Italy
tomassetti@fe.infn.it

Abstract

Tracing resource usage by Grid users is of utmost importance — especially in the context of large-scale scientific collaborations such as within the High Energy Physics (HEP) community — to guarantee fairness of resource sharing, but many difficulties can arise when tracing the resource usage of distributed applications over heterogeneous Grid platforms. These difficulties are often related to a lack of interoperability of the accounting components across middlewares.

This paper briefly describes the architecture and workflow of the Distributed Grid Accounting System (DGAS) [1] and evaluates the possibility to extend it with a Resource Usage Service (RUS) [2, 3] interface — according to the Open Grid Forum (OGF) specification — that allows to store and retrieve OGF Usage Records (URs) [4, 5] via Web Services. In this context the OGF RUS and UR specifications are critically analyzed. Furthermore, a prototype of a RUS interface for DGAS (DGAS-RUS) is presented and the most recent results towards a full interoperability between heterogeneous Grid platforms are outlined.

Keywords: Grid Interoperability, Distributed Grid Accounting, Resource Usage Service, Usage Record.

1. Introduction

Although both research and production Grid projects have initially focused on designing, implementing and deploying the basic functionalities required for executing user applications in a transparent way on globally distributed heterogeneous resources — such as Resource Brokers (RBs) or meta-schedulers, certificate-based security infrastructures and middleware components that hide the underlying local systems on Computing Elements (CEs) and Storage Elements (SEs) — the need for a proper accounting of resource usage by Grid users has recently significantly increased. Above all multi-organizational collaborations with stringent funding policies are interested in accurately tracing the resource usage by single users as well as entire Virtual Organizations (VOs).

The requirement of proper usage accounting may concern not only widely distributed resources within a single Grid environment, but also multiple distinct Grid platforms, infrastructures or projects. In fact, many international scientific collaborations — involved in research fields such as High Energy Physics (HEP), Astrophysics, Astronomy and Bioinformatics — that form VOs of large numbers of scientists from different institutions and countries, contemporaneously use multiple "National" or regional Grid infrastructures.

The HEP experiment BaBar [6], for example, pressed by the need to quickly process data, performs its computational work on the LHC Computing Grid (LCG) [7] in Europe [8] and GridX1 [9, 10] in Canada [9] without, however, sharing software and resources between the different Grids.

ATLAS [11], another international HEP experiment, relies on tools and resources provided by LCG, NorduGrid [12] and Grid3/OSG [13], but wraps the diverse middleware stacks with a custom tool (Windmill [14]) that provides uniformity of job submission and management.

Tracing resource usage over multiple Grids will also require additional efforts to improve the interoperability between different Grid accounting infrastructures and tools.

The standardization work we describe in this paper is related to the OMII-Europe project [15] whose Accounting Activity task aims at achieving interoperability between different accounting systems through the adoption of com-

mon standards. Our work is based on two OGF recommendations, the Usage Record (UR) [4, 5], defining a well-structured XML document containing job usage information, and the Web Services-based Resource Usage Service (RUS) [2, 3] specification, that defines an interface for storing and retrieving such accounting information. The adoption of standard interfaces for accounting tools allow Grids to exchange usage information, thereby enabling a fair sharing of resources not only within single Grids but also between collaborating Grid projects/infrastructures.

In this paper we present the design of a RUS interface for DGAS [1] (DGAS-RUS) as well as the current DGAS-RUS prototype and the obtained preliminary results. In this context we critically analyze the underlying OGF RUS and UR specifications.

The paper is organized as follows: Section 2 gives a very brief and generic introduction to DGAS (version 3.1.10), leaving out nearly all technical details, but allowing to better understand the following discussion. The OGF RUS and OGF UR are then evaluated in Section 3. The design and prototype of DGAS-RUS are presented in Section 4 and some final remarks are given in Section 5.

2. The Distributed Grid Accounting System

DGAS is an accounting toolkit — originally developed within the European DataGrid (EDG) [16] and Enabling Grids for E-sciencE (EGEE) [17] projects — conceived and designed to be completely Grid-oriented. It is based on a fully distributed client/server infrastructure without requiring a central repository of accounting information, relying instead upon a network of independent accounting servers used to keep the accounting records, as well as a network of independent servers for resource pricing in order to enable the deployment of a Grid resource market. Resource pricing is optional and of no further importance for this paper.

The *Home Location Register (HLR)* service is the part of DGAS that is responsible for keeping the accounting information for both Grid users and Grid resources. It receives the accounting information, the so called *usage records*[1], from the Grid resources, and stores them for later retrieval. Usage information can be obtained from the HLR service for single jobs as well as in aggregate form. Summary information is computed on the fly and can be queried for in a flexible way (per user, per resource, per VO, per VO group/user role, etc.).

2.1. User HLRs and Resource HLRs

DGAS associates accounting information with previously registered user and resource accounts (identified by the User DN — Distinguished Name, or subject of the user's certificate — and the Grid CE ID respectively) and foresees two logical types of primary HLR servers: the *User HLR* and the *Resource HLR* (each HLR server, however, can manage both user and resource accounts if required). A User HLR stores information from a user's or VO's point of view and is the DGAS server that users may query for accounting information concerning the jobs they have submitted. A Resource HLR stores information from a resource owner's or site manager's point of view and is the DGAS server that resource owners, or site managers, can query for information concerning their resources.

The reason of this division is straightforward. In order to guarantee a reasonable scalability there will be many HLR servers on the Grid, and different resources will be registered with different Resource HLRs. Hence it is desirable that all accounting information concerning a given user be forwarded to the HLR that manages the user's account ("User" HLR) in order to be able to compute accounting statistics for the single Grid users although they submit jobs to many different Grid resources. With a distributed accounting system with duplicated usage records each Grid participant (user or resource owner) theoretically needs to query only a single HLR server in order to have an exhaustive accounting view, nonetheless preserving a reasonable scalability that cannot be achieved through a single centralized accounting repository.

2.2. Second (or Higher) Level HLRs

For those use cases in which the collection of usage information in centralized accounting repositories is feasible and desired, DGAS allows to forward usage information from the primary Resource and/or User HLRs to so called Second Level HLRs (L2HLRs) that can in turn forward accounting data to higher level HLRs. This allows to deploy additional hierarchical levels of regional, national and/or grid-wide HLR servers without a need to alter the distributed primary accounting infrastructure. A vertical organization in VO-specific L2HLRs is also possible.

2.3. Accounting Workflow

Upon job completion, a job usage record (usually built by the DGAS metering sensors) is sent from the Computing Element to the Resource HLR that manages its account. The Resource HLR then forwards the record to the User HLR that manages the account of the user that submitted the job.[2] Optionally, this step may include an "economic

[1] In this case we talk about the DGAS UR format, not the one defined by OGF, see Section 2.4.

[2] Note, however, that the single sites have the faculty to decide whether to forward accounting information from their Resource HLR to the different User HLRs or not.

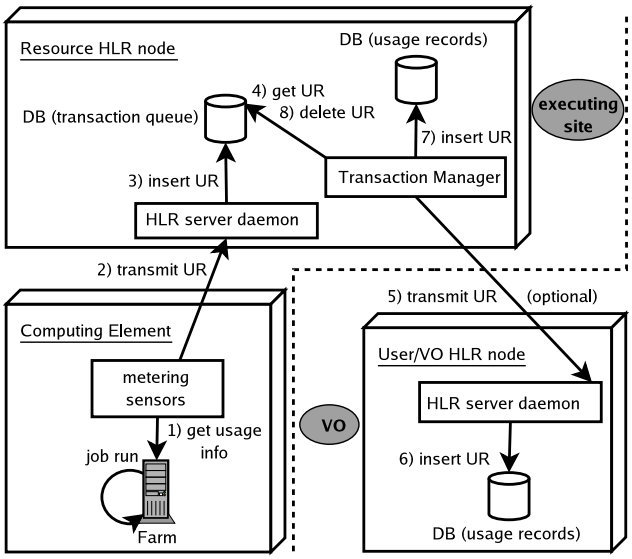

Figure 1. Simplified DGAS accounting procedure (L2HLRs are omitted).

transaction", by means of exchanging virtual credits between user and resource account. For obvious security and privacy reasons, all connections are authenticated and encrypted with x509 host certificates. As can be seen in Fig. 1, the workflow is somewhat more complex (as most communication is done asynchronously), but further details are omitted here.

2.4. DGAS Legacy Query Interface

Each HLR server can be queried to retrieve accounting information. For querying an HLR server the user has to authenticate with a valid user certificate or proxy. Access to private information is granted only to authorized users. That is, users can generally access only information regarding their own jobs, while VO admins, for example, may have access to the accounting information of the entire VO. The role-based access control (RBAC) can be either defined by means of VOMS certificates [18] or for single user DNs.

Since the development of DGAS started long before OGF recommendations, like the RUS and the UR, emerged, the communication between DGAS components uses a legacy protocol based on non-standard XML documents that are exchanged — for security and privacy reasons — via Globus GSI [19]. This DGAS-specific protocol is wrapped by client programs such that querying a DGAS HLR server, or pushing accounting information onto it, can be accomplished by means of a simple command line interface.

3. Evaluation of RUS and UR for DGAS

The OGF Resource Usage Service (RUS) specification is based on the OGF Usage Record (UR) format, in that it describes the interface to a service for storing and retrieving UR documents. Therefore, we evaluate both specifications separately with respect to functionalities of DGAS (version 3.1.10) and the accounting information stored by it. Since the RUS is based on the UR we first discuss the latter, while the RUS itself will be discussed afterwards. We will focus the discussion in this paper on conceptual aspects, for a technically more detailed discussion please see [20].

3.1. OGF Usage Record

The first version of the OGF UR specification has recently been finalized and a second version is being planned. In this Section we refer to version 1 [4].

The OGF UR is a syntactically well-defined XML document (root element `JobUsageRecord`) that contains various usage "properties" (element nodes), such as `CpuDuration`, `WallDuration`, `Memory`, `Swap`, `Disk`, `Network` and `NodeCount` that allow to specify metrics for the consumption of computational resources. This usage information can be associated with a specific job (`JobIdentity` and `JobName`), user (`UserIdentity`) and computational resource (`MachineName`, `Queue`, and `Host`). Since requirements and usage metrics can widely vary between different environments and systems, the only mandatory UR properties are the `RecordIdentity` and the job's `Status`. Moreover, an extension framework allows to specify resource usage metrics that are not explicitly covered by the standard UR properties.

Unfortunately, the OGF UR, although syntactically explicit, is semantically unclear in many data fields. `MachineName`, for example, can be the host name, cluster name, or site name [4]. `ProjectName` might be used, for example, as the name of the user's VO, but it may also refer to a Grid project or else. Likewise, `GlobalUsername` might be the user's DN or not. Such a lack of semantic definition may undermine standardization efforts because it is likely that different accounting systems will require these ambiguous data fields to be populated with different kinds of information.

Similar considerations can be made for the UR's extension framework, that allows to specify arbitrary additional information (not explicitly covered by the document specification) by means of optional `Resource` properties. The user's VO, for example, might also be specified as:

```
<Resource description="VOName">dteam</Resource>
```

where the `Resource` node's attribute `description` defines the meaning of the additional data field.

As for semantically unclear properties, an extensive use of the extension framework may lead to incompatible (although valid) UR documents, if for example different `Resource` descriptions are used to refer to the same entity (e.g. `"VOName"` and `"VirtualOrganization"`).

The risk of having incompatible UR documents can usually be neglected when considering the exchange of accounting information *within* Grid environments/projects that use a single accounting tool, but may lead to problems when exchanging accounting information *among* different Grids or accounting systems, above all if the extension framework has to be used to specify properties that are essential for the correct functioning of the accounting procedures.

The OGF UR, for example, has no explicit property to specify the unique Grid ID of a Grid resource/Computing Element. DGAS, however requires such a data field, since it associates accounting records to resource accounts that are identified by the corresponding CE IDs.

Apart from the (already mentioned) lacking property for specifying the user's VO, a data field that can be used to specify the user's role when submitting a job should be added (e.g. for role-based charging and billing). Most High Energy Physics experiments that use the LHC Computing Grid (LCG) [7], for example, require the FQAN (Fully Qualified Attribute Name) of the user's VOMS certificate [18] to be recorded.

Also, the OGF UR does not contain any information on resource (processor) performance, that can be crucial when normalizing resource consumption values across heterogeneous resources or platforms (the value of a second of CPU time largely depends on the processing power). Elements such as `ProcessingPower`, `SpecInt2000` or similar would be needed, but might in most cases also be specified through the extension framework.

Another drawback of the OGF UR is the fact that it is very batch job specific and needs to be extended (customized) for other resource types and more generic services. With respect to storage resources (that we plan to account with DGAS), for example, the OGF UR as currently defined, would allow to specify disk usage (element `Disk`), but no identifier for files (no `FileIdentity` or similar UR property).

Nonetheless, a mapping between OGF UR data fields and DGAS data fields is to a good degree possible [20]. Fields that are present on a DGAS HLR, but not foreseen by the OGF UR format, can be added as extensions to the latter (see [20] for a detailed list), having however the disadvantage of potentially undermining standardization efforts. Therefore, while we initially implemented these fields as OGF UR-compliant extensions, we proposed the addition of at least the most important fields to the official OGF UR. Of the extensions we defined (as `Resource` descriptions) for DGAS-RUS, those of more general interest for other implementations are: `"GlobalResourceId"` (for the global CE ID), `"UserVOName"`, `"UserFQAN"`, `"SiteName"`, `"specInt2000"` and `"specFloat2000"`.

3.2. OGF Resource Usage Service

The OGF RUS Working Group (RUS-WG) defines a Web Services-based interface to accounting systems, allowing a standardized upload and retrieval of resource usage information in the form of OGF UR documents. Since the first version of the RUS specification has not yet been finalized, the following discussion is based on the last stable draft that has gone through public comment (August 2006) [2]. Some improvements have already been discussed and decided in the RUS-WG and will be mentioned here where appropriate.

RUS methods are invoked by means of SOAP messages and cover the most basic functionalities a usage information service should provide (`insertUsageRecords`, `extractRUSUsageRecords`, `deleteRecords` and `modifyUsageRecordPart` to mention the most important ones). Most queries to the service are specified — by means of the XML query languages XPath [21]/XQuery [22] (for extracting stored URs and selecting them for deletion) and XUpdate [23] (for modifying records) — as input parameters to the SOAP methods. The use of XML query languages enables a standard query mechanism for UR documents that does not depend on the underlying implementations and data storage models (see Section 4.1).

The current RUS specification wraps stored URs in a *RUSUsageRecord* (RUS-UR) that contains additional audit information about who has stored/modified a UR (and when). Upon a user request RUS-URs rather than URs are returned, but the RUS-WG has recently decided that the audit information should be kept distinct from the usage information and should be returned only upon an explicit request. Therefore, the wrapping RUS-UR will probably be removed from the RUS specification. Instead the RUS interface is supposed to be augmented by an appropriate SOAP method to extract a UR's audit trail. This would allow also for a separate authorization policy for audit information (users might, for example, access the accounting data of the jobs they have submitted, but not necessarily the audit information for the URs).

Since the UR specification declares only the properties `RecordIdentity` and `Status` as mandatory (see Section 3.1), but specific RUS implementations may require other properties to be present, the RUS specification allows to specify further mandatory UR elements and provides a method for retrieving the list of mandatory elements (`listMandatoryUsageRecordElements`). Only

standard UR elements, however, can be declared to be mandatory, i.e. UR properties included through the extension framework (`Resource` elements, etc.) cannot be made mandatory. The RUS-WG, however, plans to develop a more flexible and generic mechanism for the specification of mandatory UR elements. This would be necessary for DGAS that requires the non-UR data field CE ID (or Grid resource ID) to be specified, since this ID defines with which resource account a usage record has to be associated. Likewise, DGAS user accounts are identified by the User DN (subject of the user's x509 certificate), but the corresponding element node in the OGF UR specification

```
UserIdentity|ds:KeyInfo|
ds:X509Data|X509SubjectName
```

(where the pipe indicates a parent-child relationship) cannot be declared mandatory according to the current RUS draft (although the ancestor node `UserIdentity` can).

Unfortunately, the current RUS specification is limited to the extraction or retrieval of complete RUS-URs, hence strongly focusing on a storage service and less on an information service. It is for example not possible to retrieve aggregated summary information, while many users (Grid users as well as system administrators) will rarely need the detailed per-job information and will often want only aggregated numbers for specific time periods (e.g. total resource consumption of a VO during the last month, or of a specific user on a specific resource within the last year, etc.). The legacy interface of DGAS allows to query the HLR server for such aggregated information in a flexible way.

For the RUS interface, additional SOAP methods for this purpose would be appreciable (we propose to define the method `aggregateUsageRecords`). Otherwise all aggregation has to be done by the client that queries the RUS, which is suboptimal above all when a large number of records is involved, since it significantly multiplies the amount of information that has to be transmitted from the RUS to the client. However, the OGF UR format is not meant to handle summary information, and a specific format for aggregated or summary usage information will have to be defined. More standardization work within both the UR-WG and the RUS-WG will be needed for this purpose and the two working groups have agreed upon a collaboration for achieving this goal.

4. DGAS-RUS Design and Prototype

Based on the evaluation of the OGF recommendations for UR and RUS, we have designed and implemented a DGAS-RUS prototype. Here we present its architecture, its relation to the legacy DGAS core as well as the first results of interoperability tests.

4.1. Storage Models for XML Documents

The main difficulty in implementing a DGAS-RUS is that the RUS interface handles XML documents while DGAS stores records in a relational database. XML documents can be stored in fundamentally different ways: in flat files (that are not considered here), native XML database systems or mapped to relational database tables (see for example [24, 25, 26, 27]).

Although it cannot be the purpose of this paper to evaluate all advantages and disadvantages of native XML databases and relational databases, it is necessary to discuss at least the most important ones, since they have a significant impact on the implementation of the DGAS-RUS.

Native XML Databases: The most important advantage of native XML databases is the native supporting of XML query languages such as XPath [21], XQuery [22] and XUpdate [23]. The major disadvantage is the generally lower performance compared to relational databases (see for example [26]). Research on more performant indexing techniques for XML documents in native XML databases is still going on.

Using a native XML database for the DGAS-RUS would require to continuously synchronize the content of two databases in two different database systems, since for reasons of backward compatibility (and performance) the legacy interface and legacy relational database of DGAS cannot be simply abandoned. Nonetheless records stored through the RUS interface should of course be available through the legacy interface and vice versa.

Important issues that have to be considered when choosing a database system are the critical key features necessary for a deployment in production environments — such as concurrency, transactions and safety — but also the license, that is the legal terms, under which it can be used, and the continuing support, maintenance and development. Based on these considerations, the most promising native XML database system seems to be eXist [28], an open source project (GNU LGPL).

Relational Databases: A mapping of XML elements to relational tables (in the following called "XML2RDBS") can be either *schema-based* (using knowledge about the document format — if a document schema is available as in the case of the OGF UR — for example by "inlining" child nodes, that occur at most once, as columns in the tables of parent nodes [25]) or *schema-less* (or *schema-oblivious*; a generic mapping of the XML structure to a relational database, with parent and child nodes and associations between them) [24].

The most important disadvantage of XML2RDBS is the lack of support for XPath, XQuery and XUpdate

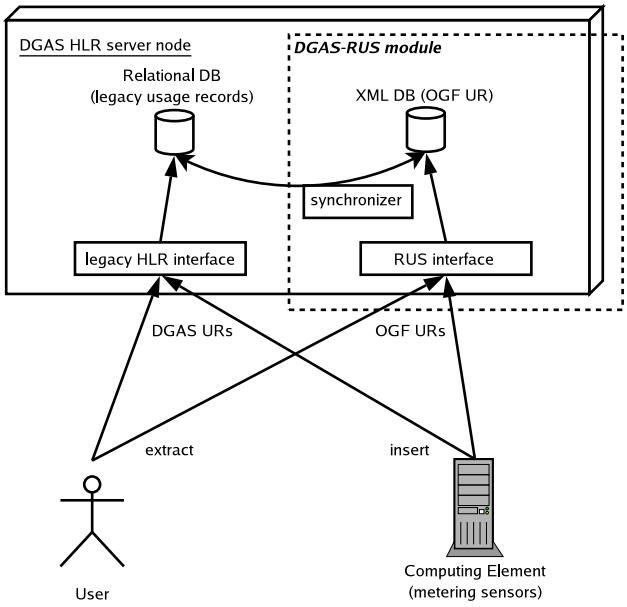

Figure 2. DGAS-RUS as an additional module to the core DGAS HLR.

and one of the major problems when translating complex XPath/XQuery or XUpdate expressions to SQL is the optimization of the queries, since a straight forward translation often leads to poor performance [25]. Even worse, no complete algorithm that considers all XML query cases seems to have been developed so far [29], but not being able to translate all possible XPath/XQuery and XUpdate expressions, that can be executed on UR documents, would undermine the compliance to the OGF RUS specification.

Last but not least, XML documents have to be parsed when being "shredded" (decomposed) in order to fit into a relational schema. Also, they have to be reconstructed in the correct order (which requires an appropriate order encoding model in addition to the storage model for the XML data content) before they can be returned to a RUS client. Both are costly operations and might add a significant amount of overhead.

Taking into account these considerations (for a more complete discussion see [20]), using a native XML database for the DGAS-RUS seems to be the more appropriate decision, keeping however in mind the lower performance and that the synchronization of the two databases (relational and XML) is a non trivial task.

4.2. DGAS-RUS as Optional HLR Module

The DGAS-RUS is designed to be an optional module for DGAS, as show in Fig. 2, such that the core system can work without the RUS interface being installed. In closed environments, the DGAS core system would be sufficient, while open systems that require interoperability can additionally install the standard RUS interface. Optionally, the DGAS-RUS may also be run as a stand-alone service. The required synchronization of the relational database, that stores the records in the legacy format, and the XML database for the DGAS-RUS, will be achieved through a dedicated process ("synchronizer") that continuously converts newly inserted usage information from one format to the other.

This approach would allow both users (for information retrieval) and metering sensors (for information storage) to use either the more optimized and performant legacy interface or the more standardized RUS interface, whichever is more appropriate. For example, a generic RUS-client might be used by an accounting portal to retrieve OGF URs from a site that uses DGAS although jobs have been accounted using the legacy interface. Likewise, a Grid user might also use the legacy interface to retrieve accounting information originating from a site that does not use DGAS but has forwarded OGF URs, using the standard RUS interface, to an HLR server managed by the user's VO.

4.3. DGAS-RUS Prototype Implementation

As previously outlined, the general design of the DGAS-RUS module is thought to be minimally intrusive and to follow the architectural design of DGAS. As the DGAS HLR, the DGAS-RUS prototype has been implemented in C++. SOAP (de)serializers are generated, by the gSOAP toolkit [30], from the WSDL description of the RUS service. gSOAP offers a SOAP-to-C/C++ language binding and generates so-called stub (client) and skeleton (server) routines for all SOAP methods defined for the RUS interface, taking care of all the communication between client and server.

The following briefly introduces the DGAS-RUS client and server prototype architecture and implementation. Technical details are given in [31].

DGAS-RUS clients: The actions of RUS clients may be basically divided into two classes: *user queries* (i.e. extraction) of URs and *insertion, modification and deletion* (i.e. management) of URs. The DGAS-RUS client prototype handles both types of actions and can thus be used for invoking all server-side methods defined by the RUS specification. Since information is passed to the client by means of command line arguments (e.g. XPath expressions, UR file names, ...), it can be easily incorporated into the scripts that, for example, orchestrate the collection of resource consumption data on the CE. These DGAS-RUS *metering sen-*

sors have been derived from the legacy metering sensors that collect usage information from LRMS and CE log files to build DGAS accounting records and forward them via the legacy interface to the Resource HLR (see Section 2). For reasons of reliability, both versions of the metering sensors handle UR creation and forwarding asynchronously. This is achieved by means of two daemons, one of which generates and locally stores accounting records, the other one taking these records and forwarding them to the DGAS servers through the respective interfaces (legacy or RUS).

DGAS-RUS server: All URs received from authenticated and authorized CEs (metering sensors) are stored in a native XML database (eXist), allowing authorized user queries (XPath expressions) to be passed to the underlying database. Since eXist offers no C/C++ APIs the communication to the database is done via the XML Remote Procedure Call (XML-RPC) framework, that offers a good support for exception management and error reporting. The DGAS-RUS prototype server keeps an audit trail (in XML format) for each single UR (see Section 3.2). Since the modification or deletion of URs is not foreseen by the DGAS HLR, the DGAS-RUS interface can be configured to return, in compliance with the RUS specification, a `RUSUserNotAuthorizedFault` upon such requests.

Synchronizer: The synchronizer, necessary for keeping the legacy relational database and the XML database aligned, is currently being implemented, but the DGAS-RUS prototype can already be used as a stand-alone service.

4.4. Interoperability Tests

During this first phase of development we conducted various interoperability tests to ensure that our implementation can correctly be used with other implementations of the RUS interface.

The DGAS-RUS client, developed before the server, has been tested against a RUS server prototype developed by KTH (Swedish Royal Institute of Technology) for the SweGrid Accounting System (SGAS) [32]. Cross interoperability has recently been verified by using the SGAS RUS client implementation to connect to the DGAS-RUS prototype server. In both cases, all possible actions regarding management (insertion, modification and deletion) and extraction (user queries) of URs could be executed with success. This ensured that the two components can effectively interoperate across different grid middlewares.

Although more extensive functionality and compliance tests should be done also with other reference implementations, interoperability — the scope of the OMII-Europe project — has been taken in consideration since the beginning of the development of the RUS interface for DGAS.

5. Conclusions

In this paper we presented the prototype of a standardized RUS interface for DGAS, that can help in tracing resource usage across heterogeneous Grid platforms. In this context we critically analyzed the OGF UR and RUS recommendations on which the presented work is based.

For reasons of backward compatibility, the DGAS-RUS has been designed as an optional module for the existing HLR service, but it can also be used as a stand-alone service. Our preliminary results are an important step towards a full interoperability between heterogeneous Grid platforms (Grid interoperability) and we will continue our efforts within (and beyond) the OMII-Europe project for achieving the goal of providing the necessary information tools that allow for fair resource sharing between the different Grid projects and infrastructures of a continuously growing e-Science community.

Acknowledgments

DGAS is supported by the European Union under contracts INFSO-RI-031688 (EGEE-II project) and INFSO-RI-031844 (OMII-Europe project). The DGAS-RUS module is developed within the OMII-Europe project. We want to express our thanks to Gilbert Netzer and Fredrik Hedman of KTH, Sweden, for their help in testing the interoperability between the DGAS and SGAS RUS implementations.

References

[1] R.M. Piro, A. Guarise, and A. Werbrouck. "An Economy-based Accounting Infrastructure for the DataGrid". *Proc. 4th Int. Workshop on Grid Computing (GRID2003)*, Phoenix, AZ, November 17, 2003; and Distributed Grid Accounting System website. `http://www.to.infn.it/grid/accounting/`

[2] J. Ainsworth, S. Newhouse, and J. MacLaren. Resource Usage Service (RUS) based on WS-I Basic Profile 1.0. Draft specification: draft-ggf-wsi-rus-17, August 2006. Available at: `http://forge.ogf.org/sf/go/doc7965?nav=1`

[3] OGF Resource Usage Service Working Group website. `http://forge.ogf.org/sf/projects/rus-wg/`

[4] R.Mach et al. [L. McGinnis (ed.)]. Usage Record – Format Recommendation. Version 1. GDF.98, September 2006. Available at: `http://www.ogf.org/documents/GFD.98.pdf`

[5] OGF Usage Record Working Group website. http://www.psc.edu/~lfm/PSC/Grid/UR-WG/

[6] N. Geddes. "The BaBar computing model". *Computer Physics Communications* **110**(1):38-42, 1998.

[7] The LHC Computing Grid (LHC) project. http://lcg.web.cern.ch/LCG/

[8] C.A.J. Brew et al. "BABAR Experience of Large Scale Production on the Grid". *Proc. 2nd IEEE Int. Conf. on e-Science and Grid Computing (e-Science'06)*, Amsterdam, Netherlands, December 4-6, 2006.

[9] A. Agarwal et al. "GridX1: A Canadian computational grid". *Future Generation Computer Systems* **23**(5):680-687, 2007.

[10] GridX1 website. http://www.gridx1.ca/

[11] ATLAS Computing Group [G. Duckeck et al. (ed.)]. ATLAS Computing – Technical Design Report. ATLAS TDR-017, CERN-LHCC-2005-022, June 2005.

[12] P. Eerola et al. "Building a Production Grid in Scandinavia". *IEEE Internet Computing* **7**(4):27-35, 2003; and NorduGrid website. http://www.nordugrid.org/

[13] Grid3 website. http://www.ivdgl.org/grid2003/; and Open Science Grid (OSG) website. http://www.opensciencegrid.org/

[14] Windmill project website. http://www-hep.uta.edu/windmill/

[15] OMII-Europe project website. http://www.omii-europe.org/

[16] European DataGrid (EDG) project website. http://eu-datagrid.web.cern.ch/eu-datagrid/

[17] Enabling Grids for E-sciencE (EGEE) project website. http://www.eu-egee.org/

[18] R. Alfieri et al. "From gridmap-file to VOMS: managing authorization in a Grid environment". *Future Generation Computer Systems* **21**:549-558, 2005.

[19] I. Foster et al. "A Security Architecture for Computational Grids". *Proc. 5th ACM Conf. on Computer and Communications Security*, pp. 83-92, 1998.

[20] R.M. Piro. Evaluation and Design Plan of a RUS Interface for DGAS - Version 0.2 (draft). Technical report, September 22, 2006. Available at: http://www.to.infn.it/grid/accounting/techrep/RUSandUR4DGAS-0_2.pdf

[21] XML Path Language (XPath) 2.0. http://www.w3.org/TR/xpath20/

[22] XML Query Language (XQuery) 1.0. http://www.w3.org/TR/xquery/

[23] XML Update Language. http://xmldb-org.sourceforge.net/xupdate/

[24] J.H. Gerritsen. "Native XML databases". *5th Twente Student Conference on IT*, Enschede, Netherlands, June 26, 2006.

[25] I. Tatarinov et al. "Storing and Querying Ordered XML Using a Relational Database System". *ACM SIGMOD'2002*, Madison, Wisconsin, June 4-6, 2002.

[26] F. Weigel, K.U. Schulz, and H. Meuss. "Exploiting Native XML Indexing Techniques for XML Retrieval in Relational Database Systems". *ACM WIDM'05*, Bremen, Germany, November 5, 2005.

[27] M. Emandi et al. "Approaches and Schemes for Storing DTD-Independent XML Data in Relational Databases". *Transactions on Engineering, Computing and Technology* **13**:168-173, 2006.

[28] W. Meier. "eXist: An Open Source Native XML Database". *Lecture Notes in Computer Science* **2593**:169-183; and eXist website. http://exist.sourceforge.net/

[29] R. Krishnamurthy, R. Kaushik, and J.F. Naughton. "XML-to-SQL Query Translation Literature: The State of the Art and Open Problems". *1st International XML Database Symposium (XSym 2003)*, Berlin, Germany, September 8, 2003.

[30] R.A. van Engelen and K.A. Gallivan. "The gSOAP Toolkit for Web Services and Peer-To-Peer Computing Networks". *2nd IEEE/ACM Int. Symp. on Cluster Computing and the Grid (CCGRID'02)*, Berlin, Germany, May 21-24, 2002; and gSOAP website. http://www.cs.fsu.edu/~engelen/soap.html

[31] M. Pace. Preliminary Implementation of a RUS Interface for DGAS - Version 1.0. Technical Report, July 2007. Available at: http://www.to.infn.it/grid/accounting/techrep/DGAS-RUS-Prototype-1_0.pdf

[32] T. Sandholm et al. "An OGSA-Based Accounting System for Allocation Enforcement across HPC Centers". *2nd Int. Conf. on Service Oriented Computing*. New York, USA, November 15-19, 2004; and SweGrid Accounting System website. http://www.sgas.se/