Evaluation and Design Plan of a RUS Interface for DGAS

Rosario M. Piro

Istituto Nazionale di Fisica Nucleare (INFN) - Sezione di Torino
Via Pietro Giuria, 1 - 10125 Torino, Italy
Email: `piro@to.infn.it`

Version 0.2 (draft) - September 22, 2006

Abstract

OMII-Europe [1] is a EU-funded project for improving the interoperability of key Grid middleware components across heterogeneous Grid platforms. Due to the importance of tracing resource usage by Grid users, one of the OMII activities is specifically concerned with interoperability between different Grid accounting systems through the implementation of a common standard interface.

This document briefly describes the architecture and workflow of the Distributed Grid Accounting System (DGAS) [2, 3] and evaluates the possibility to extend it by a Resource Usage Service (RUS) [4, 5] interface — according to the OGF[1] specification — that allows to store and retrieve OGF Usage Records (URs) [6, 7] via Web Services. For this purpose the functionalities provided by and accounting information stored in the DGAS HLR (Home Location Register) service are confronted with the OGF specifications of RUS and UR. Furthermore a preliminary design plan for the DGAS-RUS is given.

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

1 Introduction

The purpose of this document is to evaluate the possibility of implementing a Web Services-based Resource Usage Service (RUS) [4, 5] interface for the Distributed Grid Accounting System (DGAS) [2, 3] and to present a preliminary design plan. This task is part of the OMII-Europe project and aims at achieving interoperability between different accounting systems through the adoption of common standards. The OGF RUS is a Web Service for storing and retrieving

[1]The Global Grid Forum (GGF) has recently merged with the Enterprise Grid Alliance (EGA) to form the Open Grid Forum (OGF).

accounting information in the OGF Usage Record (UR) format [6] (defining a particular XML document containing job usage information).

This document is organized as follows: Section 2 gives a very brief and generic introduction to DGAS, 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, with respect to DGAS functionalities and accounting information content. A preliminary design plan for implementing a DGAS-RUS is then presented in Section 4. Some final remarks are given in Section 5.

2 The Distributed Grid Accounting System

The Distributed Grid Accounting System (DGAS), previously called DataGrid Accounting System [2], is an accounting toolkit — originally developed within the European DataGrid (EDG) and Enabling Grids for E-sciencE (EGEE) projects —, conceived and designed to be completely Grid-oriented. It is based on a fully distributed client/server infrastructure without having 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.[2]

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*[3], 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 (per user, per resource, per VO).

2.1 User HLRs and Resource HLRs

DGAS associates accounting information to previously registered user and resource accounts (identified by the User DN and the Grid CE ID respectively) and foresees two logical types of HLR servers: the *User HLR* and the *Resource HLR*.[4] A User HLR stores information from a user's or VO's point of view and is the DGAS server that users can query for accounting information concerning themselves and 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

[2]The resource pricing servers, called *Price Authorities* are optional and of no further importance for the aim of implementing the OGF RUS.

[3]In this case we talk about the DGAS UR format, not the one defined by OGF, see Section 2.3.

[4]Each HLR server, however, can manage both user and resource accounts if required.

resources will be registered with different HLR servers. Hence it is advisable 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 easily 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) ideally 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. Note, however, that the single sites have the faculty to decide whether accounting information should be forwarded from their Resource HLR to the different User HLRs or not.

2.2 Accounting Workflow

For each job a usage record is sent[5] from the Computing Element to the Resource HLR that manages its account. Along with the accounting information the CE informs the Resource HLR server whether it has to transmit a duplicate of the usage record to the User HLR that manages the accounts of the user's VO or not. Additionally, this step may include an "economic 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 somehow more complex (as most communication is done asynchronously), but further details are omitted here.

2.3 DGAS Legacy Interface and User Queries

Each User and Resource HLR server can be queried by command line clients 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 may have access to the accounting information of the entire VO.

Since the development of DGAS started long before OGF recommendations, like the RUS and the UR, emerged, the communication between DGAS components (including clients and servers) uses a legacy protocol based on non-standard XML documents that are exchanged — for security and privacy reasons — via Globus GSI [8]. This DGAS-specific interface is wrapped by client programs such that querying a DGAS HLR server, or pushing accounting information onto it, can be accomplished by specifying the necessary parameters on the command line.

[5]Usually by the DGAS metering sensors, but other sensors might be used as well.

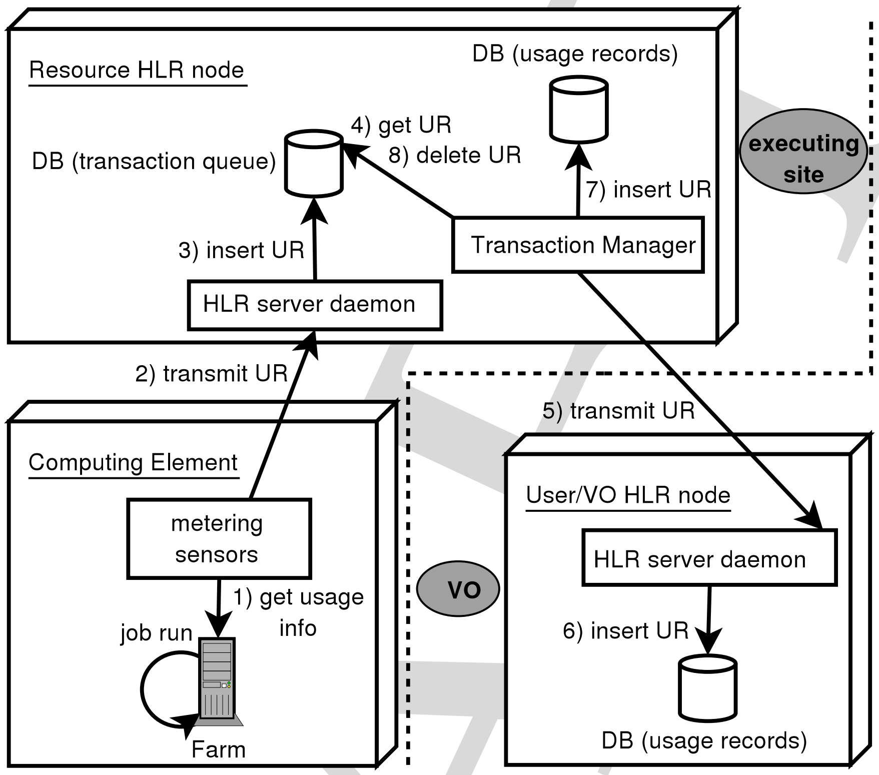

Figure 1: DGAS accounting procedure.

3 Comparison of RUS and UR to DGAS

The OGF Resource Usage Service (RUS) specification [4] is based on the OGF Usage Record (UR) format [6], in that it describes a service for storing and retrieving UR documents.[6] Therefore, we evaluate both specifications distinctively with respect to functionalities of DGAS and the accounting information stored by it. Since the RUS is based on the UR the following Section first discusses the latter, while the RUS itself will be discussed afterwards.

3.1 Comparison of the OGF UR and the DGAS Accounting Information Schema

The first version of the OGF UR specification is currently being finalized and a second version is being planned. In this Section we refer to the draft of version 1. The following is a brief (incomplete) example UR according to the OGF

[6]Actually, the RUS stores URs, but provides RUS-URs that wrap the original UR document but add information about who has stored/modified the UR (and when).

format:

```
<?xml version="1.0" encoding="UTF-8"?>
<JobUsageRecord xmlns="http://schemas.ggf.org/ur/2006/06/usage.xsd"
                xmlns:urwg="http://schemas.ggf.org/ur/2006/06/usage.xsd"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <urwg:RecordIdentity urwg:createTime="1994-11-05T13:15:30Z"
                        urwg:recordId="t2-hlr-01.to.infn.it:56568:_113"/>
   <urwg:JobIdentity>
      <urwg:GlobalJobId>https://lxb1404.cern.ch:9000/GIUnJA_DwAFjWnV2EE-pfw</urwg:GlobalJobId>
      <urwg:LocalJobId>34221.t2-ce-01.to.infn.it</urwg:LocalJobId>
   </urwg:JobIdentity>
   <urwg:UserIdentity>
      <urwg:LocalUserId>dteam007</urwg:LocalUserId>
      <ds:KeyInfo>
         <ds:X509Data>
            <X509SubjectName>/C=IT/O=INFN/OU=Personal Certificate/L=Torino/CN=Rosario Michael \
PIRO</X509SubjectName>
         </ds:X509Data>
      </ds:KeyInfo>
   </urwg:UserIdentity>
   <urwg:JobName>Hello World</urwg:JobName>
   <urwg:Charge unit="grid credits">22</urwg:Charge>
   <urwg:Status>done</urwg:Status>
   <urwg:Memory storageUnit="KB" phaseUnit="S" metric="max">125</urwg:Memory>
   <urwg:Swap storageUnit="KB" phaseUnit="S" metric="max">145</urwg:Swap>
   <urwg:WallDuration>PT2S</urwg:WallDuration>
   <urwg:CpuDuration usageType="user">PT1S</urwg:CpuDuration>
   <urwg:CpuDuration usageType="system">PT0S</urwg:CpuDuration>
   <urwg:EndTime>1994-11-05T13:15:29Z</urwg:EndTime>
   <urwg:StartTime>1994-11-05T13:15:27Z</urwg:StartTime>
   <urwg:MachineName description="site name">INFN-TORINO</urwg:MachineName>
   <urwg:Queue description="local LRMS queue">cert</urwg:Queue>
   <urwg:ProjectName description="VO name">dteam</urwg:ProjectName>
   <urwg:Host primary="true">wn42.to.infn.it</urwg:Host>
</JobUsageRecord>
```

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 [6]. `ProjectName` might be used as the name of the User VO, but it may also have different meaning. Likewise, `GlobalUsername` might be the User DN (subject of the user's certificate) or not. Such a lack of semantical 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.

Even worse, the OGF UR has no data field to specify the unique Grid ID (e.g. "`t2-ce-01.to.infn.it:2119/jobmanager-lcglsf-cert`" for LCG) of a Grid resource/Computing Element. An explicit data field `ResourceIdentity` would be more appropriate in a Grid context (and DGAS requires it, see below).

Concerning the user's identity a data field that can be used to specify the user's role when submitting a job should be added (e.g. for the purpose of charging it might be different if a user is a VO admin or not). Most LHC experiments that use LCG, for example, require the FQAN of the user's VOMS certificate [9] 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 (one second of CPU time on a Commodore 64 is very different from one second of CPU time on a Pentium-III or even newer

OGF data field	type	num	DGAS data field	type	num
RecordIdentity	—	1			
RecordIdentity#recordId	xsd:token	1	—	—	—
RecordIdentity#createTime	xsd:dateTime	0–1	urCreation(1)	string	0–1
RecordIdentity|ds:KeyInfo	ds:KeyInfo	0–1	—	—	—
—	—	—	Transaction ID	int	1(2)
—	—	—	Transaction time	timestamp	1(2)
JobIdentity	—	0–1			
JobIdentity|GlobalJobId	xsd:string	0–1(3)	dgJobId	string	1(4)
JobIdentity|LocalJobId	xsd:string	0–1(3)	lrmsId	string	1
JobIdentity|ProcessId	xsd:string	0+	—	—	—
UserIdentity	—	0+			
UserIdentity|LocalUserId	xsd:string	0–1	localUserId	string	0–1
UserIdentity|GlobalUsername	xsd:string	0–1	—	—	—
UserIdentity|ds:KeyInfo	ds:KeyInfo	0–1	User DN(1)	string	1(4)
—	—	—	group (local,user)	string	0–1
—	—	—	Resource/CE ID	string	1(4)
JobName	xsd:string	0–1	jobName	string	1
JobName#description	xsd:string	0–1	—	—	—
Charge	xsd:float	0–1	amount(1)	int	1(2,5)
Charge#description	xsd:string	0–1	—	—	—
Charge#unit	xsd:token	0–1	(always credits)	—	—
Charge#formula	xsd:string	0–1	—	—	—
Status	xsd:token	1	—	—	—
Status#description	xsd:string	0–1	—	—	—
—	—	—	exitStatus	int	0–1
Disk*	xsd:positiveInteger	0+*	—	—	—
Disk#description	xsd:string	0–1	—	—	—
Disk#storageUnit	xsd:token	0–1	—	—	—
Disk#phaseUnit	xsd:duration	0–1	—	—	—
Disk#metric	xsd:token	0–1	—	—	—
Disk#type	xsd:token	0–1	—	—	—
Memory*	xsd:positiveInteger	0+*	pmem	int	1
Memory#description	xsd:string	0–1	—	—	—
Memory#storageUnit	xsd:token	0–1	(always KB)	—	—
Memory#phaseUnit	xsd:duration	0–1	—	—	—
Memory#metric	xsd:token	0–1	(always max)	—	—
Memory#type	xsd:token	0–1	—	—	—
Swap*	xsd:positiveInteger	0+*	vmem	int	1
Swap#description	xsd:string	0–1	—	—	—
Swap#storageUnit	xsd:token	0–1	(always KB)	—	—
Swap#phaseUnit	xsd:duration	0–1	—	—	—
Swap#metric	xsd:token	0–1	(always max)	—	—
Swap#type	xsd:token	0–1	—	—	—

Table 1: Data fields of OGF URs and DGAS usage records. For the OGF UR, child elements are represented by a pipe (e.g. "RecordIdentity|ds:KeyInfo"), while attributes are represented by '#' (e.g. "RecordIdentity#recordId"). "`num`" indicates how often the data fields may or have to appear. OGF UR data fields marked by an asterisk (*) can occur multiple times as long as they are differentiated by their `metric` and/or `type` attributes. Legend: (1) This is a proposal for mapping data fields, although the fields are semantically and/or syntactically not equivalent and may require conversion; (2) Determined automatically (either by the DGAS metering sensors or the HLR server); (3) At least one of both needs to be present; (4) These unique IDs are required for DGAS, but may also be constructed if the global Grid IDs are not available; (5) If no economic accounting is desired, the charge/cost is set to zero; (6) CpuDuration can occur twice in a OGF UR, once for user CPU time and once for system CPU time (specified as attribute `type`); (7) The `SubmitHost` is usually present in the unique Grid ID generated by the User Interface; (8) Extension elements that may be used for an arbitrary purpose, the attribute `description` is used for semantical definition; (9) In case records are forwarded to the User HLR. Table continues ...

OGF data field	type	num	DGAS data field	type	num
Network*	xsd:positiveInteger	0+*	—	—	—
Network#description	xsd:string	0–1	—	—	—
Network#storageUnit	xsd:token	0–1	—	—	—
Network#phaseUnit	xsd:duration	0–1	—	—	—
Network#metric	xsd:token	0–1	—	—	—
TimeDuration*	xsd:duration	0+*	—	—	—
TimeDuration#type	xsd:token	0–1	—	—	—
TimeInstant*	xsd:dateTime	0+*	submitTime(1)	timestamp	0–1
TimeInstant#type	xsd:token	0–1	—	—	—
ServiceLevel*	xsd:token	0+*	—	—	—
ServiceLevel#type	xsd:token	0–1	—	—	—
WallDuration	xsd:duration	0–1	wallTime(1)	int	1
WallDuration#description	xsd:string	0–1	—	—	—
CpuDuration(6)	xsd:duration	0–2(6)	cpuTime(1)	int	1
CpuDuration#description	xsd:string	0–1	—	—	—
CpuDuration#usageType	"user"/"system"	1(6)	(always total)	—	—
NodeCount	xsd:positiveInteger	0–1	—	—	—
NodeCount#description	xsd:string	0–1	—	—	—
NodeCount#metric	xsd:token	0–1	—	—	—
Processors	xsd:positiveInteger	0–1	processors	int	0–1
Processors#description	xsd:string	0–1	—	—	—
Processors#metric	xsd:token	0–1	(always max)	—	—
Processors#consumptionRate	xsd:float	0–1	—	—	—
EndTime	xsd:dateTime	0–1	end(1)	timestamp	1
EndTime#description	xsd:string	0–1	—	—	—
StartTime	xsd:dateTime	0–1	start(1)	timestamp	1
StartTime#description	xsd:string	0–1	—	—	—
MachineName	domainNameType	0–1	ceHostName(1)	string	0–1
MachineName#description	xsd:string	0–1	—	—	—
—	—	—	siteName	string	0–1
SubmitHost	domainNameType	0–1	—(7)	—	—
SubmitHost#description	xsd:string	0–1	—	—	—
Queue	xsd:string	0–1	queuename	string	0–1
Queue#description	xsd:string	0–1	—	—	—
—	—	—	ctime (as for PBS)	timestamp	0–1
—	—	—	qtime (as for PBS)	timestamp	0–1
—	—	—	etime (as for PBS)	timestamp	0–1
ProjectName	xsd:string	0+	—	—	—
ProjectName#description	xsd:string	0–1	—	—	—
—	—	—	User VO	string	1(2)
—	—	—	User FQAN	string	0–1
Host	domainNameType	0+	execHost(1)	string	0–1
Host#description	xsd:string	0–1	—	—	—
Host#primary	xsd:boolean	0–1	—	—	—
ConsumableResource(8)	xsd:float	0+	—	—	—
ConsumableResource#description	xsd:string	0–1	—	—	—
ConsumableResource#units	xsd:string	0–1	—	—	—
PhaseResource(8)	xsd:float	0+	—	—	—
PhaseResource#description	xsd:string	0–1	—	—	—
PhaseResource#units	xsd:string	0–1	—	—	—
PhaseResource#phaseUnit	xsd:duration	0–1	—	—	—
VolumeResource(8)	xsd:float	0+	—	—	—
VolumeResource#description	xsd:string	0–1	—	—	—
VolumeResource#units	xsd:string	0–1	—	—	—
VolumeResource#storageUnit	xsd:token	0–1	—	—	—
Resource(8)	xsd:string	0+	—	—	—
Resource#description	xsd:string	0–1	—	—	—
—	—	—	Remote HLR server	string	0–1(9)
—	—	—	SpecInt2000	int	0–1
—	—	—	SpecFloat2000	int	0–1
—	—	—	CE timezone	string	0–1
—	—	—	accountingProcedure	string	0–1
—	—	—	atmClientVersion	string	0–1
—	—	—	economicAccounting	boolean	0–1

Continuation of Table 1 (see there for the legend).

processor). Elements such as `ProcessingPower`, `SpecInt2000` or similar would be needed.

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, for example, the OGF UR as it is now, would allow to specify disk usage (element `Disk`), but no identifier for files (`FileIdentity` or something similar is missing).

Table 1 compares the data content of a OGF UR with that of a DGAS usage record.

As can be seen, a mapping between OGF UR data fields and DGAS data fields is to a good degree possible. Moreover, DGAS stores many additional job information in a blob (MySQL) such that further information can be added to the legacy data scheme (although some data fields that occur only once in DGAS but can be present multiple times in the OGF UR[7]). Fields that are present on a DGAS HLR, but not foreseen by the OGF UR format, can be added as extensions to the latter (using `ConsumableResource`, `PhaseResource`, `VolumeResource` or `Resource`), having however the disadvantage to undermine standardization efforts. We therefore believe that while initially implementing these fields as extensions to the OGF UR, we should propose the addition to the official OGF UR of at least the most important fields, such as `ResourceIdentity`, `UserVO`, `UserFQAN` (and we have already proposed some of them to the UR-WG).

DGAS has more mandatory data fields than the OGF UR, this however is not a problem, since the RUS specification allows to define further mandatory fields. There is however one important exception: the RUS allows to specify as mandatory only data fields that are regularly part of the UR specification, i.e. data included through the extension framework (`Resource` elements, etc.) cannot be declared mandatory.

The DGAS HLR, however, requires the non-UR data field CE ID (or Grid resource ID) to be specified, since this ID defines to 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[8]

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

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

Also, the UR field `Status` (not equivalent to the DGAS `exitStatus`!) is a mandatory field that is usually not provided in DGAS. However, when URs

[7]The `CpuDuration`, for example, may be (not necessarily) split into user and system CPU usage in the OGF UR, while DGAS treats only the total value. Nonetheless a conversion from DGAS to OGF records and vice versa should be straight forward.

[8]Note that DGAS uses a slightly different format of the User DN. While the specification of `ds:KeyInfo` requires the DN fields to be separated by commas [10], DGAS uses the openssl version with slashes as field separators.

received through the DGAS legacy interface are converted to OGF UR documents, the `Status` might be derived as "`completed`" (for `exitStatus=0`) or "`unknown`" (for any other exit code).[9]

3.2 Comparison of the OGF RUS Functionalities and the DGAS HLR Service

The following is a comparison between functionalities foreseen by the RUS specification and functionalities implemented in the DGAS HLR service. The discussion is based on the August 2006 draft of the RUS specification [4].

It has to be noted that the RUS wraps stored URs in a *RUSUsageRecord* (RUS-UR) that contains additional information on who has stored/modified a UR (and when). Additionally the RUS-UR contains a `RUSRecordId` element that uniquely identifies the UR *within that specific RUS*. This element would be equivalent to the DGAS HLR transaction ID (tid), listed in Table 1, but it might be difficult to match (see the discussion of `RUS::insertUsageRecords`).

Furthermore, the RUS allows to specify further mandatory UR elements (beyond those specified as mandatory by the OGF UR, see Section 3.1) and provides a method for retrieving the list of mandatory elements. Only specific UR elements, however, can be declared to be mandatory. Extensions to the UR cannot be made mandatory.

Most RUS SOAP methods return an `OperationResult` element that reports the overall success/failure of the operation, and/or a `RUSRecordIdList` element with a detailed report for an operation on (possibly) multiple URs.

The following lists the RUS SOAP methods and describes similarities and differences with respect to the DGAS HLR service:

- `RUS::insertUsageRecords`:
  - *input*: list of UR elements
  - *output*: OperationResult, RUSRecordIdList.

  The legacy interface of the DGAS HLR allows to insert only one record per call of the respective client (*atmClient*). This, however, should be quite straight forward to implement for the RUS interface, since it won't rely on the legacy interface. A more severe problem is the fact that the RUS specification requires the `RUSRecordIds` of the inserted URs to be reported back, since DGAS usually inserts incoming records into a temporary transaction queue before asynchronously processing them (and attributing the transaction ID, or `tid`, that would be equivalent to the `RUSRecordId`). Eventually a mapping between `tids` and `RUSRecordIds` will have to be established.

- `RUS::extractRUSUsageRecords`:

[9] Please note, that "`unknown`" is not a an official value of `Status` according to the UR specification, but the specification foresees non-standard values to be supported.

- *input*: xPathQuery (XPath expression)
- *output*: OperationResult, list of RUS-URs

The DGAS HLR service provides a remote query client for the legacy interface, that allows to extract single or multiple records in a quite flexible way. It however works on completely different principles. A server engine that can handle XPath expressions will be needed (unless the handling is mandated to a native XML database, but the current implementation of DGAS relies on a relational database; see the discussion in Section 4.1).

- `RUS::extractRUSRecordIds`:
  - *input*: xPathQuery (XPath expression)
  - *output*: OperationResult, RUSRecordIdList

  See the comment to `RUS::extractRUSUsageRecords`.

- `RUS::extractSpecifiedRUSUsageRecords`:
  - *input*: RUSRecordIdList
  - *output*: OperationResult, list of RUS-URs

  The extraction of records according to their ID is consistent with the current DGAS query framework and can be easily integrated (although the current implementation does not provide UR XML documents).

- `RUS::incrementUsageRecordPart`:
  - *input*: RUSRecordId, xPathQuery (for identification of the element to be incremented), increment (xsd:long)
  - *output*: OperationResult (refers only to the location of an accessible record), modified (xsd:boolean)

  The current implementation of DGAS does not foresee to modify or replace a record after a job has been accounted. A new server engine has to be developed for this purpose.

- `RUS::modifyUsageRecordPart`:
  - *input*: RUSRecordId, XUpdate (XUpdate expression that identifies an element and describes its modification)
  - *output*: OperationResult (refers only to the location of an accessible record), XUpdateResult (xsd:boolean)

  See comment to `RUS::incrementUsageRecordPart`.

- `RUS::replaceUsageRecords`:
  - *input*: list of ReplacementRecord elements (each one composed of a RUSRecordId and a UR)

  - *output*: OperationResult, RUSRecordIdList

  See comment to `RUS::incrementUsageRecordPart`.

- `RUS::deleteRecords`:
  - *input*: xPathQuery (XPath expression)
  - *output*: OperationResult, RUSRecordIdList

  See comment to `RUS::incrementUsageRecordPart`.

- `RUS::deleteSpecificRecords`:
  - *input*: RUSRecordIdList
  - *output*: OperationResult, RUSRecordIdList

  See comment to `RUS::incrementUsageRecordPart`.

- `RUS::listMandatoryUsageRecordElements`:
  - *input*: (none)
  - *output*: OperationResult, MandatoryElements

  Only regular UR elements (see Table 1) can be declared as mandatory according to the RUS specification. Extensions (`ConsumableResource`, `PhaseResource`, `VolumeResource` and `Resource`) cannot be declared mandatory. DGAS, however, requires non-UR elements as mandatory, namely the User DN and the resource Grid ID (CE ID), see Section 3.1. These data fields identify the HLR user and resource account to which a usage record has to be associated.

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 statistical/aggregated 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, ...). The legacy interface of DGAS allows to query the HLR server for such aggregated information in a flexible way.

Additional SOAP methods for this purpose would be appreciable (we propose to define the methods `aggregateRUSUsageRecords`, `aggregateRUSRecordIds` and `aggregateSpecifiedRUSUsageRecords`). 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 clear on how to handle aggregated accounting information, leading to problems when implementing such methods for the RUS.

More standardization work within both the UR-WG and the RUS-WG will be needed for this purpose.

Another difference of the RUS specification that has to be taken into account is the limited number of error codes/fault messages that can be returned. The DGAS HLR, in contrast to the plain RUS, for example associates usage records to previously created user and/or resource accounts. In case a UR is submitted to a User HLR (via the DGAS-RUS interface) that doesn't manage the user's account, the HLR might respond with a RUS `RUSInputFault` which however is less explicit than the DGAS error `E_NO_USER ('bankEngine: the account doesn't exists in the database')`.

Other DGAS-specific features might be addressed by relying on the extension framework of the UR. The instruction to the Resource HLR to forward an inserted usage record to the User HLR, for example, might be implemented as a `Resource` element in the UR itself. If the default is "false" then URs without this extension will still be fully supported. For the (optional) economic transaction, or account balancing, that can be done with DGAS, the OGF UR's `Charge` element will be sufficient.

Still an open issue is, whether URs received by the Resource HLR through the DGAS-RUS interface will have to be forwarded to the User HLR also trough the DGAS-RUS interface or whether the (probably) more performant legacy interface can be used.

Also, DGAS will be required to integrate information on who (and when) stored/modified a usage record into its legacy data scheme (at least for storage since modification is not foreseen so far). This information will have to be contained in a returned RUS-UR (additionally to the UR itself). For old usage records on DGAS HLR servers, it should however be possible to build such information by taking the remote Resource HLR's contact string (for records that arrived from a Resource HLR to a User HLR) or the CE ID (for records that arrived from a CE to a Resource HLR).

4 Preliminary Design Plan for the DGAS-RUS

In this Section the previously described similarities and differences between the DGAS HLR service and the RUS specification are evaluated and a preliminary design plan and architecture for a DGAS-RUS is presented.

4.1 Storage of 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 (MySQL).

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 databases tables (see for example [11, 12, 13, 14]).

Although it cannot be the purpose of this document 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.

4.1.1 Native XML Databases

The most important advantage of native XML databases is the native supporting of XML query languages such as XPath [15], XQuery [16] and XUpdate [17]. The major disadvantage is the generally lower performance compared to relational databases (see for example [13]), 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. The synchronization might be achieved in basically two different ways: a) through a contemporary insertion of new record into both databases, independent on the interface through which they are submitted to the HLR (this however would add an additional overhead to the insertion); or preferably b) through a frequent synchronization by a dedicated process running on the HLR (this would also allow to automatically convert old usage records when the DGAS-RUS is installed on an HLR). Using a synchronizer would also allow to leave the previous DGAS code unchanged and thus to render the DGAS-RUS an optional module without requiring its installation in order for DGAS to work correctly, see Fig. 2.

Since the DGAS HLR server is written in C++, a native XML database should provide a C++ API in order to qualify for the implementation of the DGAS-RUS (for accessing the legacy relational database without needing to re-implement already existing DGAS code in another programming language).

More important, however, is that a native XML database provides the critical key features necessary for a deployment in production environments, such as concurrency, transactions and safety.[10] For most relational database systems these are standard features, but many native XML database systems have been developed from scratch by small research groups.

Two other issues that have to be considered when choosing a database system, are the license, that is the legal terms, under which it can be used, and the continuing support, maintenance and development.

The most promising native XML database systems seem to be Natix [18], that however has a restrictive license for non-commercial use only that does not allow to distribute derivate products (and thus cannot be used for a DGAS-RUS), and eXist [19], an open source project (GNU LGPL). Unfortunately,

[10]The DGAS HLR, for example, uses multiple threads to allow contemporary connections from multiple remote clients, including accounting requests. Thus a database needs to be able to handle multiple contemporary insertions and queries without problems.

eXist provides only a Java API. It does however provide network interfaces (XMLRPC, SOAP or REST over plain HTTP) that might be used from C++ code as well. This, however, would most probably further lower the performance of the DGAS-RUS.

4.1.2 XML documents in Relational Databases

A mapping of XML elements to relational tables (in the remainder of the document called "XML2RDBS") can be either *schema-based* (using knowledge about the document format — if a document specification 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 [12]) 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) [11]. The schema-based approach exploits knowledge about the document structure and can help in minimizing the number of join operations necessary for queries (due to the excessive fragmentation/"shredding" of the XML document in the relational model [11]), while the schema-less approach is generally less performant, but allows to store arbitrary XML documents (see [14] for a comparison of different schema-less approaches).

The most important disadvantage of XML2RDBS is the lack of support of XPath, XQuery and XUpdate. Most research has been done on how to translate such queries to SQL for the schema-less mapping, but the schema-based mapping usually requires ad-hoc design and implementations of XML-to-SQL query translation engines.[11] The main problem when translating complex XPath or XUpdate expressions to SQL is the optimization of the queries, since a straight forward translation often leads to poor performance [12]. So far no complete algorithm that considers all XML query cases seems to have been developed (see [20] for a comparison of different XML-to-SQL query translation approaches).

Another disadvantage is the fact that a schema-based approach, although minimizing the necessary join operation, is less flexible and causes difficulties when document schemes change significantly (as might be the case with version 2 of the OGF UR).

Last but not least, XML documents have to be parsed when being shredded in order to fit into a relational schema. Also, they have to be reconstructed[12] before they can be returned to a RUS client. Both are costly operations and might add a significant amount of overhead, above all when considering that the current RUS specification foresees to always return complete documents.

Even when using a relation model for storing OGF URs for the DGAS-RUS the data content in the legacy database tables and the new tables for the OGF

[11] In our case only for XPath and XUpdate. XQuery, a complex XML query language that uses XPath expressions, is currently not required for the OGF RUS. But it cannot be excluded that it will be supported in future versions.

[12] In the correct order. This requires an appropriate order encoding model since the order of XML documents has to be stored as additional data values within the relational schema that keeps the data contents of the XML documents.

UR has to be synchronized (see Section 4.1.1 on a brief discussion), but this synchronization would be more straight forward since it does not involve two completely different database systems and the code already developed for the legacy data model could be partly utilized for the new OGF UR storage model.

Nonetheless, taking into account these considerations, using a native XML database for the DGAS-RUS seems to be the more appropriate decision, keeping however in mind that the synchronization of the two databases (relational and XML) is a non trivial task.

4.2 Design of the DGAS-RUS as Optional HLR Module

The basic idea for the DGAS-RUS is to implement it as 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.

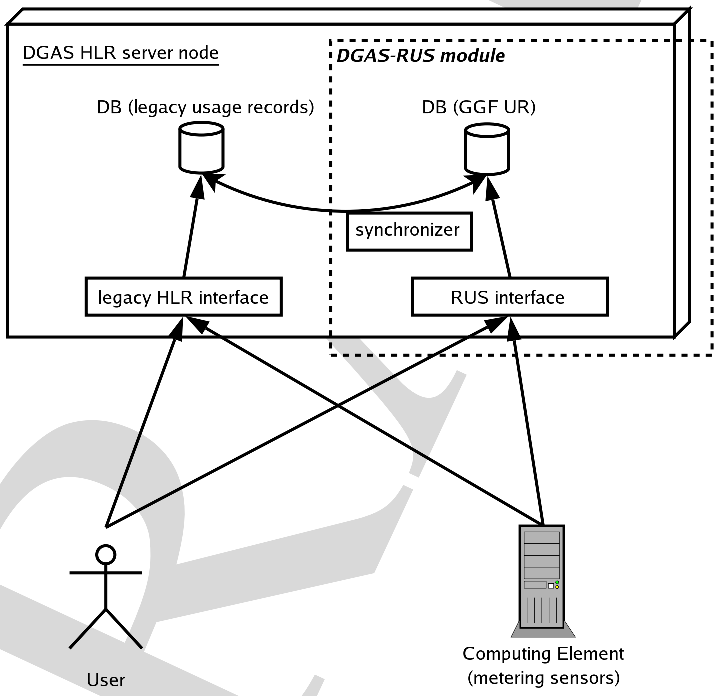

Figure 2: DGAS-RUS as an additional module to the core DGAS HLR.

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 to the DGAS HLR or the more standardized RUS interface, whatever is more appropriate.

This generic design is independent from the implementation issues discussed in the previous Sections and applies to both an underlying native XML database system or a relational model for storing OGF UR documents.

4.3 Possible Implementation of the DGAS-RUS

We have opted for using a native XML database for the DGAS-RUS (tending towards eXist), since we believe that the effort of designing, implementing and maintaining an XML-to-SQL query translator, that is required if using a custom schema-based relational model for UR storage, has many disadvantages and may require more manpower than available. The design and implementation of a daemon that keeps the legacy HLR database and the new DGAS-RUS database synchronized is preferable, since it would be necessary as well (though in a more straight forward way) for aligning the legacy database tables with new relational tables for the OGF UR.

For the implementation of the RUS interface itself, we plan to rely on gSOAP [21], a toolkit for the implementation of Web Services in C++.

Implementation details, such as the data storage model, are however still object of research and experimentation, as long as the general design of the DGAS-RUS remains the one described in Section 4.2.

5 Conclusions

The main purpose of this document was to present a preliminary design plan for a DGAS-RUS interface. For reasons of backward compatibility, the DGAS-RUS will be an optional module for the HLR service.

We furthermore discussed similarities and difference between the DGAS HLR service on one side and the OGF RUS and UR specifications on the other side.

We finally described some possible implementations, although implementation details may still be subject to changes throughout the work on the DGAS-RUS.

References

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

[2] Rosario M. Piro, Andrea Guarise, and Albert Werbrouck. "An Economy-based Accounting Infrastructure for the DataGrid". *Proceedings of the 4th International Workshop on Grid Computing (GRID2003)*, held in conjunction with SuperComputing 2003, Phoenix, Arizona, November 17, 2003.

[3] Distributed Grid Accounting System website. `http://www.to.infn.it/grid/accounting/`

[4] 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.

[5] GGF Resource Usage Service Working Group website. `http://www.doc.ic.ac.uk/~sjn5/GGF/rus-wg.html`

[6] R.Mach et al. L. McGinnis (ed.). Usage Record – Format Recommendation. Version 1. Draft specification, August 2006.

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

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

[9] V. Ciaschini. "A VOMS Attribute Certificate Profile for Authorization". April, 2004. Available at: `grid-auth.infn.it/docs/AC-RFC.pdf`

[10] XML-Signature Syntax and Processing. `http://www.w3.org/TR/xmldsig-core/`

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

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

[13] 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 5th, 2005.

[14] 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.

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

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

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

[18] Natix website. `http://pi3.informatik.uni-mannheim.de/natix.html.en`

[19] eXist website. `http://exist.sourceforge.net/`

[20] 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 8th, 2003.

[21] gSOAP website. `http://www.cs.fsu.edu/~engelen/soap.html`