LLview: User-level Monitoring in Computational Grids and e-Science Infrastructures

Wolfgang Frings[1], Morris Riedel[1], Achim Streit[1], Daniel Mallmann[1],
Sven van den Berghe[2], David Snelling[2], Vivian Li[2]

[1] Central Institute for Applied Mathematics
John von Neumann Institute for Computing,
Forschungszentrum Jülich GmbH, 52425, Jülich, Germany
[2] Fujitsu Laboratories of Europe
Hayes, Middlesex UB4 8FE, UK
email: {w.frings}@fz-juelich.de
phone: (+49 2461) 61 2828, *fax:* (+49 2461) 61 6656

Abstract
Large-scale scientific research often relies on the collaborative use of Grid and e-Science infrastructures that offer a wide variety of Grid resources for scientists. While many production Grid projects and e-Science infrastructures have begun to offer services for the usage of computational resources to end-users during the past several years, the absence of a widely accepted standard for tracing resource usage of Grid users has lead to different technologies among the infrastructures. Recently, the Open Grid Forum developed a set of emerging standard specifications, namely the Usage Record Format (URF) and the Resource Usage Service (RUS) that aim to manage and expose user tracings. In this paper, we present the integration of these standards into the UNICORE Grid middleware that lays the foundation for valuable tools in the area of accounting and monitoring. We present the development of Grid extensions for the LLview application, which allows to monitor the utilization (e.g. usage of cluster nodes per users) of Grid resources controlled by Grid middleware systems such as UNICORE.

1 Introduction

Over the last years many production Grid projects and e-Science infrastructures such as DEISA, D-Grid, EGEE, TeraGrid and OSG have begun to offer services for the usage of computational resources to end-users. These infrastructures indicate an increasing number of application projects that require access to computational resources such as supercomputer, desktop Grids or clusters. The access to resources within these infrastructures is usually provided by Grid systems such as UNICORE [2], gLite [3], or Globus Toolkit-based services [29]. Projects such as OMII-Europe [26] or the Grid Interoperation Now (GIN) community group of the Open Grid Forum (OGF) have begun to work towards interoperability between these different Grid middleware systems.

With regard to the on-demand provisioning and usage of resources within these Grids members of a Virtual Organization (VO) [23] are very interested to examine the utilization across these resources that can be realizing by tracing the usage of Grid resources for each users. This in turn lays the foundation to charge users for the use of the consumed resources. Furthermore, VOs are typically interested to monitor resource activity within their Grid and e-Science infrastructures. The absense of a widely accepted standard for tracing resource usage within these e-Science infrastructures via the Grid middleware led to different technologies related to accounting, billing and monitoring in the past. For example, Distributed Grid Accounting (DGAS) [15] in gLite (EGEE), or SweGrid Accounting System (SGAS) [17] in SweGrid and Globus Toolkit-based Grids. Both use proprietary interfaces to exchange and trace resource usage records that lay the foundation for accounting and billing. Furthermore, several monitoring technologies evolved that provide an overview of the current resource usage on systems, for instance Ganglia [21], or Inca [25].

In order to provide interoperability for resource accounting, billing, and monitoring across different Grid and e-Science infrastructures (e.g. DEISA, EGEE, or TeraGrid) the Usage Record Format (URF) [10] work performed within the OGF introduced a common standardized format for tracing the usage of Grid users. Such records lay the foundation for sharing of usage information among Grid sites and a wide variety of technologies. This includes information about resource consumption such as the usage of nodes and processors per end-users. In addition, the Resource Usage Service (RUS) [5] working group of the OGF define standard interfaces for inserting and retrieving such URF specific pieces of information. Currently, the OMII - Europe project augments the Grid middleware systems UNICORE, gLite (via DGAS) and Globus Toolkit (via SGAS) with these set of interfaces that will lead to the mentioned interoperability across Grid borders in the future.

This interoperability lead to several thousands of processors and it is not feasible to monitor the usage of system and batch load with command line tools, because the lists or tables in their output are becoming too large and complex. Therefore, the LLview [24] monitoring application was developed and extended to monitor the utilization of these resources, e.g. within the DEISA infrastructure. In this paper we present the development of generators that are able to store OGF URF-compliant usage records for each Grid user that utilizes a computational resource. Furthermore, we introduce the integration of a Web Services Resource Framework (WS-RF) [18] compliant RUS service into the Grid middleware UNICORE in order to expose these usage records through a standard interface. In order to provide an example use case scenario, we describe the LLview monitoring application that gives a quick and compact summary of usage records, including several statistics and graphical information.

The remainder of this paper is structured as follows. In Section 2 we introduce the integration of RUS interfaces and URFs into UNICORE. Section 3 presents the LLview monitoring application that use these standardized interfaces and formats. The paper ends with related work in Section 4 and concluding remarks.

2 Dynamic Resource Usage Architecture of UNICORE

In recent years, the UNICORE 5 [2] evolved to a full-grown and well tested Grid middleware system that is used in daily production at supercomputing centers and research facilities worldwide. Furthermore, it serves as a solid basis in many European and International research projects (e.g. OMII - Europe, Chemomentum [22], and A-WARE [12]) that use existing UNICORE components to implement advanced features and support scientific applications from a growing range of domains [2]. UNICORE is open source under BSD license and available at sourceforge [16] and represents the major middleware of the DEISA supercomputing Grid infrastructure.

More recently, the first prototype of the Web service-based UNICORE 6 evolved that is based on emerging standard technologies such as the WS-RF, proclaimed as an official standard by OASIS at April 2006. The adoption of standards into Grid middleware such as UNICORE provides basic interoperability among the different systems and thus make the change from one middleware to another easier and more transparent to the scientists so that they can concentrate on their scientific workflows. The dynamic resource usage architecture described here is based on UNICORE 6 and relies on recent work performed within the RUS and UR working groups of OGF.

2.1 Augmenting UNICORE with a RUS-compliant Interface

UNICORE provides seamless, secure and intuitive access to distributed Grid resources (e.g. supercomputer, clusters) by interacting with the underlying local batch subsystem or Resource Management System (RMS). Therefore, RMSs such as Torque, PBSPro, LSF or LoadLeveler are connected via the UNICORE *Target System Interface (TSI)*. The fundamental idea of the dynamic resource usage architecture of UNICORE is to record Usage Record Format (URF) compliant documents securely for a site running a UNICORE TSI and to allow the distribution of them to interested parties, in a manner that meets to confidentiality requirements of sites and users. This is achieved by the integration of a Resource Usage Service (RUS) specification [5] compliant interface into UNICORE as shown in Figure 1. In more detail, it represents a higher-level service on top of the UNICORE Atomic Services (UAS) described by Riedel et al. in [6]. The UAS consist of a *Target System Factory (TSF)* that is used to create an instance of the *Target System Service (TSS)* and thus implements the WS-RF factory pattern [18]. By traversing the enhanced UNICORE gateway [4], end-users can use the TSS to submit jobs which descriptions are compliant with the emerging standard Job Submission and Description Language (JSDL) [1]. Afterwards, the *Job Management Service (JMS)* can be used to control the job, while the *Storage Management Service (SMS)* and the *File Transfer Service (FTS)* are used for staging job related files in and out of the UNICORE environment. The JSDL based job description is parsed and interpreted by the enhanced Network Job Supervisor (NJS) [7] at the backend that also performs the authorization of users by using the enhanced UNICORE User Database (UUDB).

The RUS interface uses the functionality of the UAS underlying NJS backend to execute an extension within the TSI that is capable of providing up-to-date URFs as shown in Figure 1. Hence, the invocation of a RUS operation (e.g. *ExtractUsageRecords()*) leads to the definition and submission of an internal job to the NJS. After successful authorization, the rather abstract job definition for the extension execution is translated into non-abstract job descriptions, a process named as incarnation, by using the Incarnation Database (IDB) at the NJS. Finally, the execution request is forwarded via the TSI to its extension without using the RMS for scheduling on the HPC resource. The execution of the extension represents a URF generator that will be described in more detail in the next section. However, the execution outcome is a XML document with URFs that are transfered back to the NJS and then exposed via the RUS interface to service consumers.

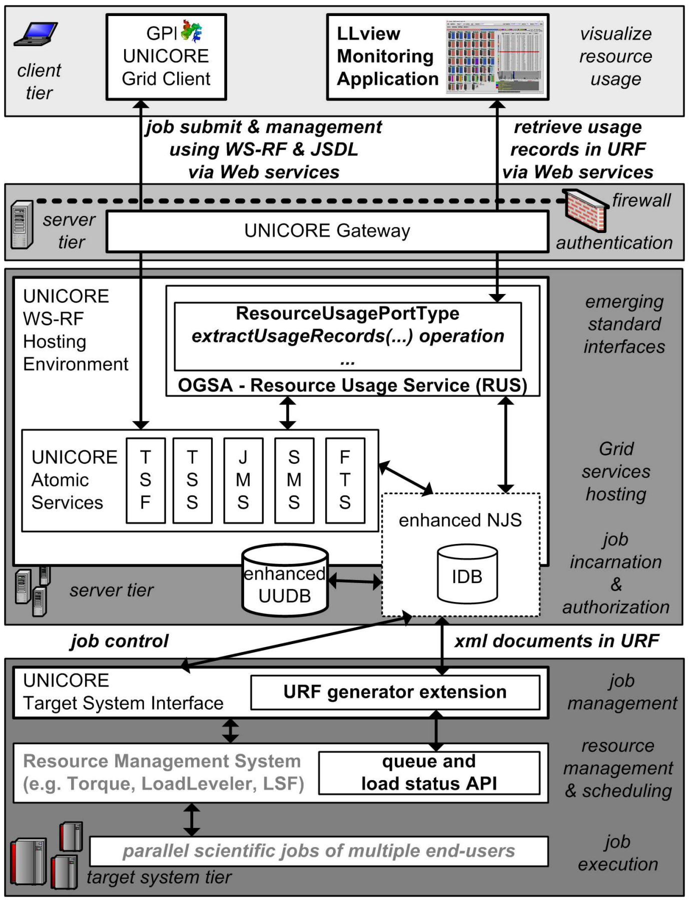

client tier
GPI UNICORE Grid Client
LLview Monitoring Application
visualize resource usage
job submit & management using WS-RF & JSDL via Web services
retrieve usage records in URF via Web services
firewall
server tier
UNICORE Gateway
authentication
UNICORE WS-RF Hosting Environment
ResourceUsagePortType
extractUsageRecords(...) operation
...
OGSA - Resource Usage Service (RUS)
emerging standard interfaces
UNICORE Atomic Services
TSF
TSS
JMS
SMS
FTS
Grid services hosting
enhanced NJS
IDB
enhanced UUDB
server tier
job incarnation & authorization
job control
xml documents in URF
UNICORE Target System Interface
URF generator extension
job management
Resource Management System (e.g. Torque, LoadLeveler, LSF)
queue and load status API
resource management & scheduling
parallel scientific jobs of multiple end-users
target system tier
job execution

Figure 1: The UNICORE RUS interface provides up-to-date URF-compliant information about the resource usage and used by the LLview monitoring tool.

2.2 Extending UNICORE with URF Generators

As shown in Figure 1, the TSI is the component of UNICORE that is interfacing with the RMS system which is installed on a target system. Hence, computational jobs that are submitted via the TSS of UNICORE will be forwarded via the enhanced NJS to the TSI to submit it finally to the underlying RMS for scheduling and execution on the High Performance Computing (HPC) resource. In more detail, UNICORE provides a dedicated TSI component for each available RMS such as Torque, LoadLeveler, PBSPro, LSF and others as shown in Figure 2.

Since UNICORE typically interacts with the underlying RMS for the control of computational jobs, such RMSs must be adapted in order to get accurate up-to date usage records. In this context, the gathering of information about resource usage and thus the URF generator is also dependent from the installed RMS system. To provide an example, we describe briefly how pieces of information are gathered from the LoadLeveler RMS that is installed on supercomputer JUMP within DEISA. As shown in Figure 2 a small C-program uses the data access C-API of LoadLeveler to get precise information about node usage, including running and waiting jobs. This c-program is integrated into UNICORE via an extension at the TSI that is called via the NJS by the RUS service. The retrieved information is a URF-compliant XML document with up-to-date information from computing resource. Of course, there have been also activities started developing other URF generators for different RMSs such as Torque/PBRPro and LSF.

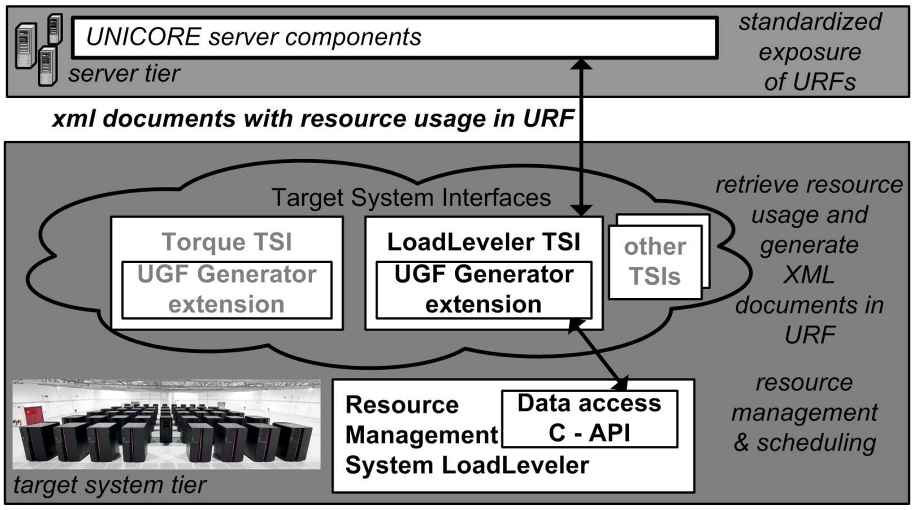

UNICORE server components
server tier
standardized exposure of URFs
xml documents with resource usage in URF
Target System Interfaces
Torque TSI
UGF Generator extension
LoadLeveler TSI
UGF Generator extension
other TSIs
retrieve resource usage and generate XML documents in URF
Resource Management System LoadLeveler
Data access C - API
resource management & scheduling
target system tier

Figure 2: UNICORE architecture with new resource usage tracing capabilities.

3 Monitoring Grid Resource Usage with LLview

Today large scale scientific research relies on the collaborative usage of Grid resources such as supercomputers or clusters provided within Grid and e-Science infrastructures. Hence, the Grid resource is shared among a wide variety of end-users such as single individuals or user groups that represent the members of a VO. To provide a sophisticating infrastructure for such end-users the availability and reliability of the Grid resources is of major importance. In this context, monitoring is absolutely necessary for resource providers as well as for the management of VOs. Monitoring refers to the process of observing Grid resources to track their status for purposes such as problem solving or load evaluations. The work described within this paper emphasize monitoring that provides a view on the real existing physical resources (nodes and cpus) that include the monitoring of Grid jobs submitted via a Grid middleware. Hence, it rather focuses on Grid cluster monitoring instead of Grid services monitoring.

Beside billing and accounting, a RUS interface of a Grid middleware such as UNICORE lays the foundation for resource level monitoring. In this context, the retrieved usage records in the standardized URF represent the up-to-date status of the VO resources. It is important that this up-to-date status is well presented to end-users and thus it is feasible to create visual images within a Graphical User Interface (GUI) from complex datasets with URFs instead of pure tables. This is necessary because the human mind is used to make inferences from this imagery in order to get a better insight of the usage records or to get an overall picture of the current load situation. One example of such a GUI for monitoring resource usage within Grids is the following LLview application that is able to act as a service consumer of a RUS service.

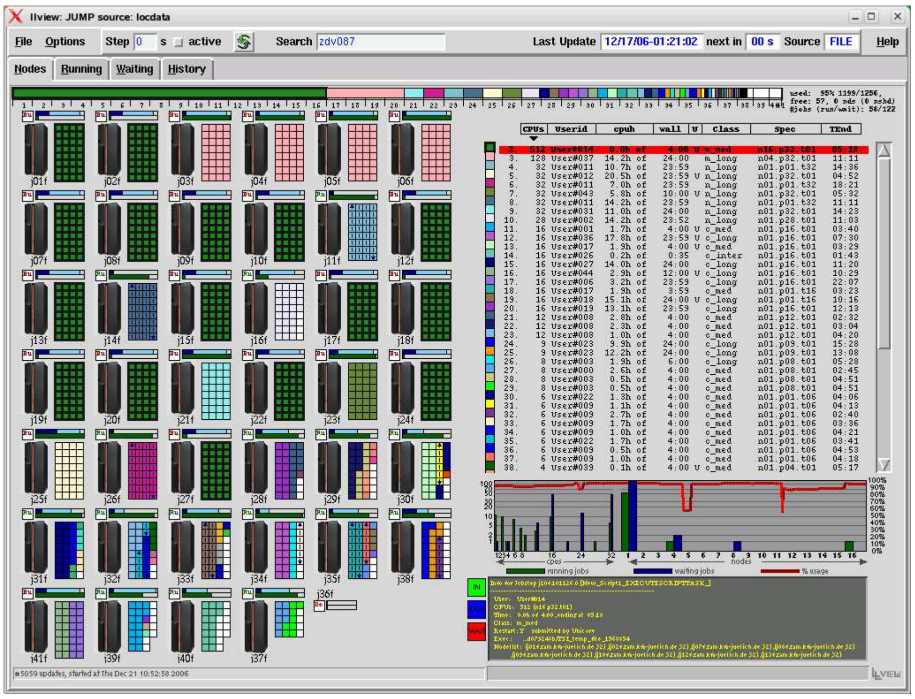
3.1 LLview Monitoring Application GUI

The LLview monitoring application [24] is a known tool in the area of system management and used by scientists for resource reservation estimations as well as support people at user help desks to resolve problems. In addition, administrators use LLview to get an load status overview of the computational resource they administrate. It represents a visualization of the mapping between running jobs and nodes of clusters controlled by a batch system. It offers a wide variety of illustrations in only one window, including efficient supervision node usage, running and waiting jobs, several statistics, a history of jobs as well as reservations. This fully configurable application provides interactive mouse sensitive information about resource usage as shown in Figure 3 via the red line. Note that in this figure the userids are just named with numbers for confidentiality reasons, but it can be also configured to show the login names of end-users that submitted jobs on the Grid resource.

LLview was initially designed to work without a Grid service provided by a Grid middleware. Therefore, there are four different modes in which the LLview client GUI can access data from the server part of LLview named as *llqxml*. The different modes for data access can be selected in the Option panel of LLview and the currently used mode will be displayed in the status bar of the LLview GUI (see upper right corner in Figure 3). The four different modes are as follows.

First, the LLview GUI can access the data directly if the client runs on the same machine or the LLview is configured to use a ssh-connection to the corresponding machine. Therefore, LLview is implemented in perl that is accepted on the most supercomputer and clusters today. However, in this mode LLview executes the llqxml server part at every update step. Second, the usual way is to distribute the data by a Web server to support clients running on local desktops. In this case, LLview accesses the data from the Web server with a pre-configured username/password authentication method. A perl script named as *getllqxml.pl* can be used as a crontab script for regular update of the XML file on the Web server. This script is available in the util directory of the LLview distribution.

Furthermore, LLview provides a mechanism to record data and replay recent usage statistics. Therefore LLview is able to read a tar file which contains XML files in a proprietary format. Such tar files can be recorded by a separate perl script *getwwwdata.pl* that is also available in util directory. In addition, LLview can read XML files from a directory. Finally, the fourth recently developed mode is an interface to a RUS compliant Grid service that will be described in more detail in the next paragraph. This mode can be used to seamlessly integrate the LLview application into Grid and e-Science infrastructures.



```
Step 0 s   Search zdv087   Last Update 12/17/06-01:21:02   next in 00 s   Source FILE

used:  95% 1199/1256,
free: 57, 0 nds (0 nshd)
#jobs (run/wait): 58/122

     CPUs Userid      cpuh      wall  U Class     Spec          TEnd
  2.  512 User#014   0.0h of   4:00  U m_med     n16.p32.t01   05:18
  3.  128 User#037  14.2h of  24:00    m_long    n04.p32.t01   11:11
  4.   32 User#011  10.7h of  23:59    n_long    n01.p01.t32   14:36
  5.   32 User#012  20.5h of  23:59  U n_long    n01.p32.t01   04:52
  6.   32 User#011   7.0h of  23:59    n_long    n01.p01.t32   18:21
  7.   32 User#043   5.8h of  10:00  U n_long    n01.p32.t01   05:32
  8.   32 User#011  14.2h of  23:59    n_long    n01.p01.t32   11:11
  9.   32 User#031  11.0h of  24:00    n_long    n01.p32.t01   14:23
 10.   28 User#002  14.2h of  23:52    n_long    n01.p28.t01   11:03
 11.   16 User#001   1.7h of   4:00  U c_med     n01.p16.t01   03:40
 12.   16 User#036  17.8h of  23:59  U c_long    n01.p16.t01   07:30
 13.   16 User#017   1.9h of   4:00  U c_med     n01.p16.t01   03:29
 14.   16 User#026   0.2h of   0:35    c_inter   n01.p16.t01   01:43
 15.   16 User#027  14.0h of  24:00    c_long    n01.p16.t01   11:20
 16.   16 User#044   2.9h of  12:00  U c_long    n01.p16.t01   10:29
 17.   16 User#006   3.2h of  23:59    c_long    n01.p16.t01   22:07
 18.   16 User#017   1.9h of   3:59    c_med     n01.p01.t16   03:23
 19.   16 User#018  15.1h of  24:00  U c_long    n01.p01.t16   10:16
 20.   16 User#019  13.1h of  23:59    c_long    n01.p16.t01   12:13
 21.   12 User#008   2.8h of   4:00    c_med     n01.p12.t01   02:32
 22.   12 User#008   2.3h of   4:00    c_med     n01.p12.t01   03:04
 23.   12 User#008   1.0h of   4:00    c_med     n01.p12.t01   04:20
 24.    9 User#023   9.9h of  24:00    c_long    n01.p09.t01   15:28
 25.    9 User#023  12.2h of  24:00    c_long    n01.p09.t01   13:08
 26.    8 User#003   1.9h of   6:00    c_long    n01.p08.t01   05:28
 27.    8 User#000   2.6h of   4:00    c_med     n01.p08.t01   02:45
 28.    8 User#003   0.5h of   4:00    c_med     n01.p08.t01   04:51
 29.    8 User#003   0.5h of   4:00    c_med     n01.p08.t01   04:51
 30.    6 User#022   1.3h of   4:00    c_med     n01.p01.t06   04:06
 31.    6 User#009   1.1h of   4:00    c_med     n01.p01.t06   04:13
 32.    6 User#009   2.7h of   4:00    c_med     n01.p01.t06   02:40
 33.    6 User#009   1.7h of   4:00    c_med     n01.p01.t06   03:36
 34.    6 User#009   1.0h of   4:00    c_med     n01.p01.t06   04:21
 35.    6 User#022   1.7h of   4:00    c_med     n01.p01.t06   03:41
 36.    6 User#009   0.5h of   4:00    c_med     n01.p01.t06   04:53
 37.    6 User#009   1.0h of   4:00    c_med     n01.p01.t06   04:18
 38.    4 User#039   0.1h of   4:00  U c_med     n01.p04.t01   05:17

Info for Jobstep j10f101126.0 [New_Script1_EXECUTESCRIPTTASK_]
-------------------------------------------------------
 User:   User#014
 CPUs:   512 (n16.p32.t01)
 Time:   0.0h of 4:00 ,ending at 05:18
 Class:  m_med
 Restart: Y    submitted by Unicore
 Exec:     ...d079248b/TSI_temp_file_1568854
 Nodelist: (j01f.zam.kfa-juelich.de,32) (j02f.zam.kfa-juelich.de,32) (j07f.zam.kfa-juelich.de,32) (j08f.zam.kfa-juelich.de,32)
           (j09f.zam.kfa-juelich.de,32) (j10f.zam.kfa-juelich.de,32) (j12f.zam.kfa-juelich.de,32) (j13f.zam.kfa-juelich.de,32)

5059 updates, started at Thu Dec 21 10:52:58 2006
```

Figure 3: LLview displays resource usage statistics, nodes and job status.

3.2 Interactions between LLview and RUS services

The augmentation of LLview with a RUS client as shown in Figure 4 allows for the extraction of up-to-date information about the current load situation at a Grid site. In particular, the XML-based documents with URFs from the generator at a Grid site can be queried via the emerging standard RUS interface and in turn visualized within the GUI. Therefore, the document with URFs must be parsed and should provide all necessary information to visualize the resource usage like within Figure 3. This is possible by using the schemas of the URFs and map the different tags to the particular parts within the GUI. To provide a small example, Figure 4 shows a small piece of a document with usage records that are compliant with the OGF URFs and used in LLview.

The RUS client within the perl-based LLview is based on the SOAP:Lite [19] package that represents a perl implementation for the Simple Object Access Protocol (SOAP) [11] and is capable of invoking Web service operations at a Grid site that offers a RUS interface. In this context it seems reasonable to consider the lifetime of the information as well as performance implications. The RUS interface itself is standardized within the specification [5] however the implementation itself is Grid-specific. For instance, the RUS interface for UNICORE described within this paper supports two modes. One mode is to query the URF generator at the TSI for each request in the RUS interface while another mode remains a cached copy for a defined period (e.g. 1 minute) using the lifetime management mechanisms in UNICORE 6. To conclude, the information displayed within LLview is either the up-to-date situation or older with a maximum of the defined time period. Furthermore, the time period for updates can be configured within LLview in order to control the amount of Web service requests.

Finally, the LLview application must be seamlessly integrated into Grids by using the same certificates that are also used within usual Grid clients (e.g. UNICORE GPE clients [13]) that are used for job submit. That means the secure access to the RUS interface within a Grid middleware is handled via standardized X.509 certificates. To provide an example in UNICORE, end-users can only invoke RUS operations if the UNICORE gateway authenticated them based on their certificates. Furthermore, an end-user must be authorized via the UUDB in order to retrieve usage records.

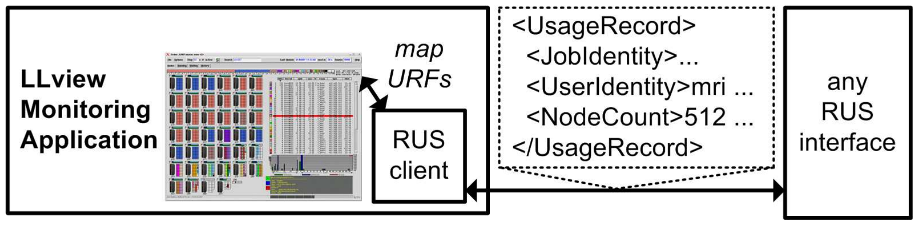


Figure 4: LLView monitoring application with the recently developed RUS client.

4 Related Work

The approach described in this paper is similar to others used in the field. Its primary advantage is the use of standard interfaces (e.g. WS-RF and RUS) as well as standardized content information (URF). This includes flexibility of the URF information and a robust higher-level service implementation that provides basic security (SSL and UUDB). A slightly different approach to monitoring represents the cluster monitor system Ganglia described by M.L. Massie in [21]. A. Coooke et al. describe in [9] the monitoring framework of the EGEE infrastructure that is named as R-GMA wherein all data appear if it is centrally available within one relational database. Another monitoring tool is the Inca Reporting Framework described by S. Smallen et al. in [25]. Its main difference to our approach is that it does not gather cluster or queuing data and rather focuses on software stack validation and site certification. There are many other tools that provide functionality within Grids in the context of monitoring and all have their special characteristics such as the network monitoring in Nagios [28] or the automatic emailing system within Hawkeye [20]. However, other systems are Clumon [14] and MonaLisa [30]. Finally, the Monitoring and Discovery System (MDS) 4 of the Globus Toolkit is described by J. Schopf et al. in [27] and focuses on Grid service monitoring instead of resource monitoring. It relies on standard schemas for information representation such as the GLUE schema [8] and provides a Web-based user interface called WebMDS. The integration of other cluster monitoring systems is possible via information providers.

5 Conclusion and Future Work

The integration of a RUS interface into UNICORE and the usage of these services by the LLview monitoring application provides a true benefit for Grid administrators and customer service management as well as end-users to plan job submits. Also, the standardized exposure of URF-compliant usage records via a standardized interface lays the foundation for accounting and billing across borders of the wide variety of interoperable Grids that exists today. The developed LLview RUS client presented here is able to extract usage records in URF from UNICORE 6 and thus allows for monitoring the load within UNICORE Grids. Use case scenarios of this new LLview application include resource level monitoring within D-Grid and DEISA when UNICORE 6 becomes the production middleware for these infrastructures in the future. Finally LLView runs at the John von Neumanns Institute for Computing (NIC) continously to show visitors the current load on the systems. Future work exists in the context of extending LLview to show the status of multiple Grid clusters in one GUI.

Acknowlegments

This work is partially funded by the OMII - Europe project under EC grant RIO31844-OMII-EUROPE, duration May 2006 - April 2008.

References

1. A. Anjomshoaa et al., Job Submission and Description Language 1.0, OGF, 2006
2. A. Streit et al. UNICORE - From Project Results to Production Grids, Elsevier, Grid Comp. and New Frontiers of High Perf. Proc., pages 357–376, 2005
3. gLite Middleware, http://glite.web.cern.ch/glite
4. R.Menday, The Web Services Architecture and the UNICORE Gateway. In Proceedings of the International Conference on Internet and Web Applications and Services (ICIW) 2006, Guadeloupe, French Caribbean, 2006
5. OGF RUS-WG, http://forge.gridforum.org/projects/rus-wg
6. M. Riedel et al., Standardization Processes of the UNICORE Grid System. In Proc. of 1st Austrian Grid Symposium 2005, Linz, pages 191-203
7. B. Schuller et al., A Versatile Execution Management System for Next Generation UNICORE Grids. In Proc. of the 2nd UNICORE Summit at EuroPar 2006
8. GLUE SCHEMA 1.3, http://glueschema.forge.cnaf.infn.it/Spec/V13
9. A. Cooke et al., R-GMA: An Information Integration System for Grid Monitoring, In proc. of the 11th Int. Conference on Cooperative Information Systems, 2003
10. OGF UR-WG, http://forge.gridforum.org/projects/ur-wg
11. M. Gudgin et al. SOAP version 1.2 Part 1: Messaging Framework, W3C 2003
12. A-Ware project, http://www.a-ware.org
13. R. Ratering et al., GridBeans: Supporting e-Science and Grid Applications. In 2nd IEEE e-Science conference 2006, Amsterdam
14. CLUMON System, http://clumon.ncsa.uiuc.edu/
15. R. Piro et al. An Economy-based Accounting Infrastructure for the DataGrid, Proc. of the 4th Int. Workshop on Grid Comp., Phoenix, 2003
16. UNICORE Grid middleware, http://www.unicore.eu
17. T. Sandholm et al. A service-oriented approach to enforce grid resource allocation, Int. Journal of Cooperative Inf. Systems, Vol.15, 2006
18. OASIS Web Services Resource Framework TC, http://www.oasis-open.org/committees/tc_home.phpwg_abbrev=wsrf
19. R.J.Ray et al. Programming Web Services with Perl, ISBN:0596002068
20. Hawkeye Condor Monitoring, http://www.cs.wisc.edu/condor/hawkeye/
21. M.L. Massie et al., The Ganglia Distributed Monitoring System: Design, Implementation, and Experience, Parallel Computing, 30(7), 2004.
22. Chemomentum Project, http://www.chemomentum.org
23. I. Foster et al. The Anatomy of the Grid - Enable Scalable Virtual Organizations. In F. Berman, G.C. Fox, and A.J.G. Hey, editors, Grid Computing - Making the Global Infrastructure a Reality, pages 171-198, John Wiley & Sons Ltd, 2003
24. LLview Application, http://www.fz-juelich.de/zam/llview
25. S. Smallen et al., The INCA Test Harness and Reporting Framework. In proceedings of Supercomputing 2004, November 2004.
26. Open Middleware Infrastructure Institute for Europe, http://www.omii-europe.org
27. J. Schopf et al. Monitoring and Discovery in a Web Services Framework: Functionality and Performance of Globus Toolkit MDS4, HPDC 2006
28. NAGIOS System, http://www.nagios.org
29. I. Foster et al. Globus Toolkit 4: Software for Service-Oriented Systems, IFIP Int. Fed. for Inf. Processing, LNCS 3779, pages 2-13, 2005
30. I. Legrand et al. MonALISA: An Agent based, Dynamic Service System to Monitor, Control and Optimize Grid based Applications, CHEP 2004, Interlaken, 2004