Monitoring UNICORE jobs executed on Desktop Grid resources

Jozsef Kovacs
MTA SZTAKI
Budapest, Hungary
smith@sztaki.hu

Filipe Araujo, Serhiy Boychenko
CISUC
Dept. of Informatics Engineering
University of Coimbra, Portugal
filipius@dei.uc.pt, serhiy@student.dei.uc.pt

Matthias Keller
University of Padeborn
Padeborn, Germany
mkeller@uni-paderborn.de

Andre Brinkmann
JGU Mainz
Mainz, Germany
brinkman@uni-mainz.de

***Abstract*— The EDGI project builds a federation of Desktop Grid sites, which provide volunteer and institutional compute resources. EDGI offers these resources to gLite, ARC, and UNICORE user communities. The jobs can be directly bridged to the Desktop Grids via the standard gLite, ARC, or UNICORE interfaces. The paper focuses on the UNICORE side, especially on the monitoring aspect. A bridging mechanism has been implemented in a modified UNICORE computing element to forward jobs towards the Desktop Grid servers. This component is part of the EDGI infrastructure, where all the bridge-related components and resources are continuously monitored by a central monitoring system. The monitoring system provides information about the traffic among the sites in the infrastructure. The paper gives a short overview of the EDGI infrastructure, especially the monitoring system and introduces the technical details about how the monitoring is integrated and supported by the modified UNICORE computing element. The introduced solution is part of the EMI software stack to make it easily accessible for UNICORE infrastructure providers.**

I. Introduction

The European Grid Infrastructure (EGI) creates and provides an eScience infrastructure for European research communities. Those communities, reaching from high energy physics to humanities, can choose from different grid technologies to execute their computations: Grids of cluster computers, like the German D-Grid, supercomputers, like the Distributed European Infrastructure for Supercomputing Applications (DEISA), or desktop grids consolidated in the International Desktop Grid Federation (IDGF). Each of the technologies has different properties concerning access limitations, amount and types of available resources, and privacy properties. UNICORE [1] supports both clusters and supercomputers, which are also known as service grids (SGs). Extending UNICORE to support desktop grids (DGs) is the final step to enable UNICORE to operate with all available grid resources. This extension enables European eScience communities, especially UNICORE VOs, to access all available grid resources from one middleware.

Desktop grids collect idle resources from loosely coupled desktop computers. Each DG client only has to install a small client software to run computations. DGs exploit that compute capacities of modern computers are rarely exhausted while performing daily tasks, like reading mails and websites or writing texts. The DG client software now allows the utilization of idle capacities for scientific computations. The desktop computers form either an institutional/private/campus or a voluntary/public desktop grid. The first scenario aggregates idle resources out of computer pools of an university or out of employees' computers of a company. The second scenario motivates the general public to install DG client software on their home computers and therewith to donate compute resources for science similar to popular projects like SETI@home [2].

Since there is no high-speed network between the desktop computers, only a subset of applications is applicable to them, like parameter studies (PS). Parallel applications requiring a high-speed interconnect should still be executed on service grids. However, PS currently running on service grids can be outsourced to DGs, leaving more available resources for researchers in need of highly interconnected nodes. The power of the DG is the potential to collect and utilize millions of desktop computers owned by citizens. A public desktop grid can collect hundreds of thousands or even millions of computers from the volunteers worldwide, depending on how appealing the supported applications are for the citizens. In case of a campus desktop grid, the typical size is several thousands since they are collected within an institute/company.

The aim of the European Desktop Grid Initiative (EDGI) is to deploy desktop grid and cloud services for EGI user communities that are heavy users of distributed computing infrastructures (DCIs) and require an extremely large multinational e-infrastructure. In order to achieve this goal, software components of ARC, gLite, UNICORE, BOINC, XWHEP, ATTICS, and 3G Bridge have been integrated into a platform, which can move jobs from service grids to desktop grids. Therefore, EDGI extends ARC, gLite, and UNICORE grids with gateways to volunteer and institutional DG systems. EDGI also integrates a bridge between service grids and Eucalyptus, OpenStack, and OpenNebula cloud environments to be able to support QoS for the DG environments. This enables EDGI to explore new provisioning models in order to ensure a harmonized transition from service grids to desktop grids. EDGI will also provide a workflow-oriented science gateway to enable user communities to access the EDGI infrastructure more easily. The EDGI project established the IDGF

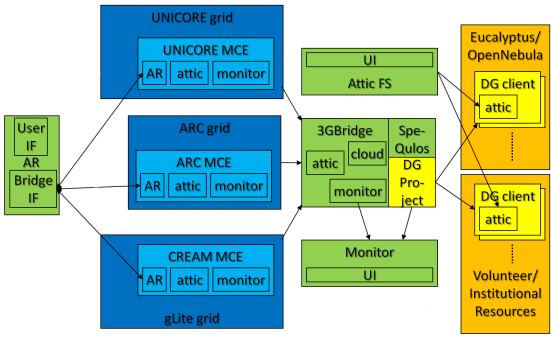

Fig. 1. The EDGI software infrastructure.

organization to coordinate DG-related activities in Europe, both for solving technical issues as well as to attract volunteer DG resource donors by disseminating results of the EDGI and EGI projects. IDGF and EDGI collaborate with EGI, EMI, NorduGrid, the UNICORE Forum, and interested NGIs.

The rest of the paper is organized as follows. Section 2 gives a short overview of the EDGI infrastructure. Section 3 describes the technical challenges of extending UNICORE as a UNiform Interface to Computing REsources to support multiple DGs. Section 4 describes the EDGI monitoring infrastructure. Finally, Section 5 concludes the work described in this paper.

II. Overview of the EDGI infrastructure

EDGI is a project supported by the FP7 Capacities Programme. This section will give a short overview of the key components of the EDGI infrastructure [3], which are developed and maintained in the project. The aim of the infrastructure shown in Figure 1 is to provide a gateway to desktop grid resources for those users who are familiar with gLite, ARC, or UNICORE type service grids and do not intend to move their environment to another type of grid. Adding a huge number of desktop grid resources can significantly increase the overall capacity of a service grid system.

Desktop grid systems can be built based on institutional or volunteer resources. In the former case, an institution (e.g., a university) is maintaining the resources, while in the latter case, the resources are provided by private PC owners. In DG systems, applications must be ported (i.e., modified according to the requirements of a DG software like BOINC) and validated (to make sure they do not cause any harm on the resource) before being deployed on the server. After the deployment, which includes registering the application with all of its binaries, work units can be submitted. In DG systems, volunteers trust the applications provided by the attached DG when letting the application run on their PCs. To store the validated applications centrally, a new *Application Repository* (denoted "AR" in Figure 1) has been introduced. The most important information related to an application are a detailed description, binaries, example input files, supported service grids and desktop grids. Users can also select an application and submit it to a well-known service grid.

The specially prepared *Modified Computing Element* (MCE inside gLite, ARC, or UNICORE in Figure 1) is responsible for forwarding the job to a desktop grid. Before doing that, it checks whether the application originates from the AR (through the "Bridge IF") and then whether the application is supported (i.e., registered) on the target desktop grid. Access to a desktop grid is provided by the *Generic Grid-Grid bridge* (3G Bridge) [4] component (depicted in the middle of Figure 1) running on the server of the target desktop grid system. The 3G Bridge is responsible for inserting the job as a work unit into the desktop grid system (BOINC or XtremWeb), for keeping track of its progress, and for reporting the status and results back to the service grid computing element.

The EDGI infrastructure is also able to utilize cloud computing resources (shown in the upper-right corner of Figure 1) by instantiating virtual machines to speed up computation when actual resources are considered unable to provide enough capacity. The dynamic scheduling of cloud resources is part of the newly developed SpeQuloS [5] software, which resides on the DG server. The Attic P2P file system [6] is used to store huge input files for jobs executed many times to decrease the load on the desktop grid server. Monitoring [7] in EDGI collects information about service and desktop grid components and provides a web-based portal to inspect the utilization of the overall infrastructure.

III. Implementation

This section describes the extension of UNICORE to access desktop grid resources. After starting with a brief UNICORE description, the implementation is described and critical technical details are outlined (please see [8] for a more detailed description).

A. UNICORE Ecosystem

The following section describes the distributed architecture of UNICORE. UNICORE is divided into three layers: client, service, and system. Users and their applications can access UNICORE services via different client tools. The UNICORE services, associated with the service layer, are loosely coupled Java Web services. They have to be registered in at least one well-known *Service Registry*. The concrete resources, clusters, or storage systems are UNICORE's *target systems* and are part of the system layer. The services on top of these target systems provide a system independent UNiform Interface to (COmputing) REsources (UNICORE).

UNICORE/X is the core component of a UNICORE site. It hosts the webservices for job and storage management. The *Target System Service (TSS)* is created by a factory (TSF) and provides access to native resources. It uses the backend component, the eXtended Network Job Supervisor (XNJS), sometimes referred to as the job management and execution engine of UNICORE. For job submission a *Job Management Service* is created and provided. The *Job Management Service (JMS)* handles the control flow of a single job: submit, abort, pause, and monitor. To execute these requests the XNJS is used. The *Storage Management Service (SMS)* offers unified

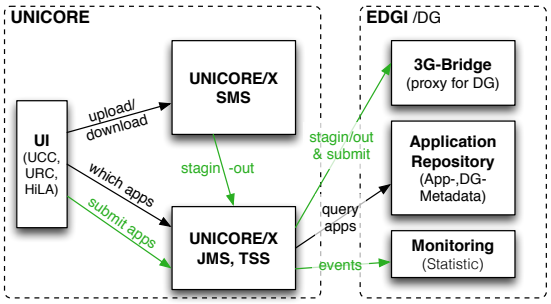

Fig. 2. Architecture and Interaction of UNICORE and EDGI components.

access to arbitrary storage systems. The main functionality is implemented inside the backend XNJS component. Finally, the *File Transfer Service (FTS)* enables concurrent file transfer operations.

These services are known as UNICORE atomic services (UAS) and are indirectly accessible through a *Gateway* that ensures security. Information about available services are stored in a special *Registry Service*.

The *Gateway* component is the single entry point to a UNICORE site. It authenticates all incoming requests and forwards them to their intended destination services. Replies are sent back to the clients. To authenticate users and assign roles to them, the Gateway queries the XUUDB, which is a user database mapping user credentials (certificates, distinguished names) to access privileges for each service and to a set of attributes, like Unix logins, roles, or projects.

The *Registry Service* provides a central point of registration for different services of a UNICORE installation. At least one registry is needed to run an installation.

The *eXtended Network Job Supervisor (XNJS)*, also called UNICORE engine, contains most of the core functionalities of a UNICORE site, which is used by most UASs. It uses the *Target System Interface (TSI)*, which enables a unified access to target systems for resource and storage handling. The XNJS handles the control flow of a job, the staging to and from local storage systems and submitting to local resource management systems. This is done through the TSI, which implements the access to native management and storage systems.

The Target System Interface (TSI) implements access to local systems. In a typical grid environment, a resource management software (RMS) schedules a job and a shared file system provides access to the working directory for the job. Currently, two types of TSI implementations exists: a java and a perl implementation. The java implementation is mainly used for test purposes. It executes a shell process and stores data locally. The perl implementation runs a set of perl scripts with different implementations for accessing Torque, SGE, OpenCCS, or PBS.

B. Architecture and Implementation

This section describes the changes required to enable DG job submission. Figure 2 shows UNICORE and EDGI components and their interactions. With the User Interface (UI), a user can upload his input data to an SMS, query a UNICORE/X server for available applications, and submit jobs using those applications and input data. The UNICORE/X server will download the input data from the SMS and submit the job to the 3G Bridge, which forwards the job to the target DG. After the job terminates on a DG client, the UNICORE/X server downloads the output data from the DG and stores it on the given target storage. Later, the user can download the results from the target UNICORE storage.

Typically, an UNICORE/X site represents a single cluster. The extension keeps this architecture by wrapping a single DG installation. Therewith, querying each UNICORE/X for application or resource information results in individual data for each DG. Additionally to submit a job, a user can choose a DG installation. Wrapping a target system is usually done by implementing a TSI. We decided to implement a java based TSI, which includes a full reimplementation of the TSI for accessing remote components like the application repository and the 3G Bridge.

The *staging process* is slightly different from a standard UNICORE system, because data has to be transferred additionally to and from the DG head node. The XNJS initiates through the TSI a stage-in process. This creates a working directory for the job and downloads every input file into this directory. In contrast to the stage-out process, this stage-in process is unchanged and works by using existing code supporting every UNICORE file transfer protocol.

Afterwards, the XNJS triggers a job submission and the new TSI transforms the UNICORE job request into a 3G Bridge request. This request cannot contain input data, which is probably hundreds of megabytes large. Thus, the 3G Bridge expects an alternative way to fetch the input data, e.g., via HTTP. The 3G Bridge is typically collocated with the DG management software on the DG head node, so the 3G Bridge moves these files to the correct place. To enable fetching input files, the filespace-directory, in which UNICORE creates working directories, is exposed via a HTTP-server. To inform the 3G Bridge about the download location, the input file location of the request is replaced by the corresponding remote URL.

If nothing unexpected happens, the status will change from PENDING to RUNNING and FINISHED. The new TSI observes this by polling the 3G Bridge. If FINISHED is reached, the 3G Bridge provides a list of output data with HTTP URLs. This data is downloaded to the job's working directory. This functionality is hooked in prior to UNICORE's normal stage-out process to enable a fully supported file transfer. Thus, the XNJS believes the computation is not finished until the output file has been downloaded from the remote HTTP server. After every file transfer is finished, the job and its files are deleted on the DG site, and the normal XNJS functionality continues: stage-out after job computation. This implementation preserves UNICORE/X stage functionality and supports different target and source storage systems. It also cleans the 3G Bridge or the DG head node up.

The *Incarnation Database (IDB)* is basically an xml-file

read by the XNJS. The IDB is used to describe native or local properties of the target system, for which the UASs provide an uniform interface. The IDB has roughly three parts describing the kind of local resources, CPU architecture, cluster size, the available applications, and specific execution environments used by applications, e.g., MPI. The solution updates the IDB at UNICORE/X start time with information from the application repository, which relates to the currently accessed DG. This can also be triggered from command line and can be continuously updated if an interval is specified. This eases the administrative effort, because newly available applications for DGs are automatically available for UNICORE users.

The *XMLLogReport* class handles the EDGI internal monitoring system (Section IV-E), but does not need to communicate with external components, because the monitoring system follows a pull model for observed nodes (Section IV-A). For this, a directory is exposed by an HTTP server. The `XMLLogReport` class writes monitoring events in files, handling the special XML format for the monitoring and cleaning up outdated files.

URL-passthrough is a concept realized in EDGI for glite to bypass local staging processes. Publicly available data can be downloaded directly from the DG clients. Otherwise, the data has to be downloaded to a UNICORE/X side, then to a DG head node, and from there DG clients could fetch them. This would result in a waste of bandwidth and potentially cause network bottlenecks on the UNICORE/X site or the DG head node. For bypassing, the URL has to stick to the following format: bypass://*[protocoll]*:///*[host,path to file]*:*[md5]*:*[size]*.

IV. Monitoring

A. Overview

The EDGI monitoring system provides information about EDGI resources, including historical and real time data of the connected grid systems. In particular, besides the UNICORE grids that we cover in this paper, monitoring must also integrate the following technologies: BOINC, XtremWeb, gLite, ARC and cloud technologies.

Figure 3 gives an overview of the components of the EDGI monitoring system. It depicts two sites as large rectangles: the central and a foreign site. The (single) central site is responsible for the following functions: 1) collecting, 2) storing and 3) displaying data. To collect data from (multiple) remote sites (1), the monitoring system uses "foreign probes". To centrally gather these data, it uses an additional probe at the central site. (2) Then, we store the data that goes through the probes in a local MySQL database and in a round-robin database (RRD)[1]. This latter database keeps data with different degrees of granularity depending on its age. For instance, it will keep all data of the last day, but it will use increasing levels of consolidation (thus loss) for weekly, monthly and yearly views. (3) To display data from the databases, we currently use a Java Server Faces web site run by a GlassFish[2] Application Server.

[1]http://www.mrtg.org/rrdtool/, visited on February 2^{nd} 2012.
[2]http://glassfish.java.net/, visited on February 2^{nd} 2012.

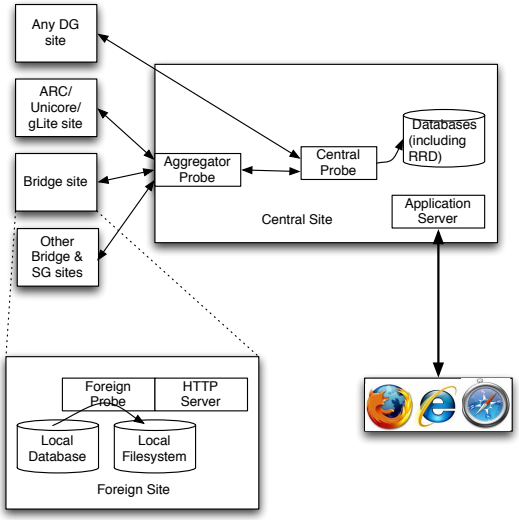

Fig. 3. Modules and components of the monitoring environment.

B. The Data

We get data from several different resources. In some cases, we need to query a BOINC or XtremWeb database, to get, for example, the number of available cores, the number of jobs that are running, or the number of FLOPs available. In this case, we just extract and filter the data into a different format: we periodically read from a MySQL database, write it in XML, for later storage in an RRD database. For other grids, we may have to produce data, based on job-related events, like job submission, completion, or failure, to know exactly what happened to each one of these jobs. This will enable us to compute the overall number of jobs that finished successfully or that failed, for instance. To have a better understanding of this latter case, we illustrate in Table I a number of events that two components of the EDGI infrastructure produce: the first one is the modified Computing Element, which submits jobs to the second one, the 3G Bridge, which finally submits jobs to BOINC or XtremWeb for execution. As we can see from the table, we can determine exactly what is happening to each job.

C. XML Creation

To make the data available, we use XML files. In practice, these serve as the glue that binds the monitoring system together. We publish the XML files on the resources sides and periodically download these files to the central site using the HTTP protocol. The advantage of HTTP is that it completely decouples communication between the foreign probes and the central probe. System administrators must configure their HTTP servers to enable the central probe to read the entire list of files available in the directory. The names of the XML files include the creation UNIX epoch. The central probe uses these timestamps to keep track of the files it has already read. It first reads the entire content of the directory, then determines which files are new, and, finally, it downloads them one by

TABLE I
EVENTS OF INTEREST IN EDGI

Event	Format
Modified Computing Element	
Start	dt=2010-01-06_13:30:30 event=job_entry job_id=xxxxxxxxxxxx application=yyyyyyyy input_grid_name=zzzz
Status (Scheduled, Waiting, Finished, ...)	dt=2010-01-06_13:30:30 event=job_status job_id=xxxxxxxxxxxx status=Running
Submission	dt=2010-01-06_13:30:30 event=job_submission bridge_id=bridge_URI job_id=xxxxxxxxxxxx job_id_bridge=zzzzzzzzzzzzzz status=Submitted
3G Bridge	
Start	dt=2010-01-06_13:30:30 event=job_entry job_id=zzzzzzzzzzzzz application=yyyyyyyy
Status (Scheduled, Running, Finished)	dt=2010-01-06_13:30:30 event=job_status job_id=zzzzzzzzzz status=Running— Scheduled—Finished
Submission	dt=2010-01-06_13:30:30 event=job_submission job_id=zzzzzzzzzz job_id_dg=yyyyyyyy output_grid_name=xw—boinc/xyz.com

one. For the sake of saving space, this scheme requires periodic cleaning of XML files on the local sites. System administrators can, e.g., use a cron daemon and file timestamps to regularly clean old files.

To provide an idea of how do these files look like, we give an example in Figure 4. This particular file contains events regarding job submission to a 3G Bridge, from an MCE. For convenience of the presentation, we split some job identifiers.

D. The Probe Concept

We run three sorts of probes: "foreign", "aggregator" and "central". The foreign probe runs in the monitored sites, such as BOINC, XtremWeb, 3G Bridge, etc.. The central probe is common to all monitoring data paths. It receives data like "80 cores" or "24 jobs" referring to some specific resource. The aggregator probe keeps track of data that is still changing across the different grids. For instance, if the 3G Bridge submitted a job to BOINC, we must keep its identifier to know when it finishes. The aggregator keeps track of the life cycle of jobs and sends consolidated data to the central probe using XML itself. In this example, when the job finishes, the aggregator probe can add one to the number of jobs that finished for that specific 3G Bridge, for that specific application, and output this fact in the next XML file it writes.

We offer two mechanisms to publish the XML data: one of them are "foreign probes", the other are Java and C libraries that developers can use in their source code. Usually these probes search for specific text patterns in log files or run SQL queries in databases to get data, e.g., job start, job finish, or number of CPU cores. If users want to read data from gLite resources, from BOINC, or from XtremWeb they should install the appropriate probes. For instance, in BOINC and XtremWeb, the main (but not single) goal of the probe is to collect data regarding the desktop grid alone, e.g., the number of running jobs, or the number of available CPU cores. In the 3G Bridge, the metrics are slightly different, as we want data regarding jobs that cross the bridge. Overall, we have the probes for BOINC and XtremWeb, the gLite modified Computing Element (MCE) probe, the 3G Bridge probe, the aggregator probe and the central probe. The probes have been developed in Java. Some of them work as stand-alone daemons, while others run under the control of a cron daemon. This means that a system administrator must manually control the former, while the latter reads a number of metrics and sleeps until the end of the next period. Table II enumerates which probes are daemons and which ones are not.

```
<?xml version="1.0" encoding="UTF-8"?>
<report timestamp="1328540459014"
            timezone="GMT" version="1.1">
  <metric_data>
    <dt>2012-02-06 15:59:57</dt>
    <event>job_entry</event>
    <job_id>https://grid40.lal.in2p3.fr:9000/
                  jw65o5NitPSFQPxhExvHgA
    </job_id>
    <application>guineapig-pp-1.1.1</application>
    <input_grid_name>
        gLite/demo.vo.edges-grid.eu
    </input_grid_name>
  </metric_data>
  <metric_data>
    <dt>2012-02-06 15:59:59</dt>
    <event>job_submission</event>
    <job_id>https://grid40.lal.in2p3.fr:9000/
                  jw65o5NitPSFQPxhExvHgA
    </job_id>
    <job_id_bridge>
          ad956fa0-0c06-4d94-ba20-db01e6cc2e0f
    </job_id_bridge>
    <status>Submitted</status>
    <output_grid_name>
        http://xw.lri.fr:4322
    </output_grid_name>
  </metric_data>
  <metric_data>
    <dt>2012-02-06 16:00:12</dt>
    <event>job_status</event>
    <job_id>https://grid40.lal.in2p3.fr:9000/
            jw65o5NitPSFQPxhExvHgA
    </job_id>
    <status>Running</status>
  </metric_data>
</report>
```

Fig. 4. Example of an XML file generated by a probe

TABLE II
EXECUTION MODE OF THE PROBES

Probe	Daemon/Cron
BOINC	Cron
XtremWeb	Cron
MCE/gLite	Cron
3G Bridge	Cron
Aggregator	Daemon
Central Probe	Daemon

E. XML Libraries

Although the monitoring probes still run in an important fraction of the EDGI infrastructure, they have important disadvantages, because they need complex cron daemon configurations and must run in different systems. Additionally, they require hard to maintain rpm and debian packages. Since the standard in EDGI monitoring is, in fact, settled by the format of XML files, one alternative to run previously prepared probes is to make the grid resources write the appropriate XML themselves. We started to change this with the ARC middleware. Despite producing the same XML output as the monitoring probes, ARC uses its own code to get a similar result. This makes monitoring simpler and more robust.

With UNICORE, we took the ARC model further ahead. We created a Java library that the UNICORE developers hooked to their code. This library allows UNICORE to directly call Java methods whenever events of interest happen, e.g., a new job is submitted to the 3G Bridge. In this way, neither we depend on external processes, nor we force grid developers to learn the internal details of the monitoring format.

The core of this library is the `XMLLogReport` class. The library also contains a few helper classes, namely the `ReportEntry` and the `OutputThread`. The former represents the data, while the latter, which extends the class `Thread`, periodically outputs the new data into XML (each 10 minutes by default), while deleting the old files (one week old by default). The `XMLLogReport` class contains the following public methods:

- `startReporting()`. This method initiates the `OutputThread` if it is not running yet.
- `stopReporting()`. Make the reporting thread finish after the next reporting event.
- `setReportsPath()`. Set the file system path where the thread should output the XML files to.
- `setCleanupAge()`/`getCleanupAge()`. Set or get the age of files that should be deleted on cleanup.
- `jobSubmission()`. Report the submission of a job. This data includes the identifier and the id of the job in the 3G Bridge. This allows the monitoring system, more precisely, the aggregator, to correlate all the identifiers of the same job across the entire grid infrastructure.
- `jobEntry()`. This method serves to register the entrance of a new job in the UNICORE grid.
- `jobChangeStatus()`. This method should be called when the infrastructure changed the status of some job or when it became aware of job status change on some other infrastructure.
- `output()`. Forces a report output. All the events that were created by the previous methods are outputted to a XML files upon invocation of this method.
- `cleanup()`. Deletes all files that are older than the threshold defined in `setCleanupAge()` or older than a default threshold if no other threshold has been set.

We also provided a `main()` method that uses the Java library for the sake of testing it.

V. CONCLUSION

We have described the motivation for and the implementation of a UNICORE extension as well as the corresponding monitoring environment, which directly supports desktop grids from within this service grid middleware. The extension will be integrated into the EMI environment. The stability of the extensions is ensured by several unit tests and also by EDGI's work package to test the developed infrastructure. Thus, an active cycle between test engineers and developers is established.

Additional concurrent project efforts port a lot of application to run on desktop grids, so that the UNICORE community can choose from a rich set of applications. These new resources will boost research, not only for those who use the additional desktop grid resources, but also by leaving more capacity for the rest of the users. In summary, the release quality, the broad application spectrum, and newly available resources make this extension a valuable contribution to the UNICORE community.

ACKNOWLEDGMENTS

The research leading to these results is funded by the European Union Seventh Framework Program (FP7/2007-2013) under grant agreement no 261556.

REFERENCES

[1] A. Streit, P. Bala, A. Beck-Ratzka, K. Benedyczak *et al.*, "Unicore 6 - recent and future advancements," *Berichte des Forschungszentrums Jülich*, vol. 65, 2010.

[2] D. Anderson, "Boinc: A system for public resource computing and storage." in *Proceedings of the 5th IEEE/ACM International GRID Workshop*, Pittsburgh, USA, 2004, pp. 1–7.

[3] P. Kacsuk, J. Kovacs, Z. Farkas, A. Marosi, and Z. Balaton, "Towards a powerful european dci based on desktop grids," *Journal of Grid Computing*, vol. 9, pp. 219–239, 2011.

[4] Z. Farkas, P. Kacsuk, Z. Balaton, and G. Gombás, "Interoperability of boinc and egee," *Future Generation Computer Systems*, vol. 26, no. 8, pp. 1092 – 1103, 2010.

[5] S. Delamare, G. Fedak, D. Kondo, and O. Lodygensky, "Hybrid distributed computing infrastructure experiments in grid5000: Supporting qos in desktop grids with cloud resources," in *Proceedings of the Grid500 Spring School*, 2011.

[6] I. Kelley and I. Taylor, "Bridging the data management gap between service and desktop grids," *Data Management*, pp. 13–26, 2008.

[7] F. Araujo, D. Santiago, D. Ferreira, J. Farinha, L. M. Silva, P. Domingues, E. Urbah, O. Lodygensky, A. Marosi, G. Gombas, Z. Balaton, Z. Farkas, and P. Kacsuk, "Monitoring the edges project infrastructure," in *Proceedings of the 3rd Workshop on Desktop Grids and Volunteer Computing Systems (PCGrid)*, Rome, Italy, May 2009.

[8] M. Keller, J. Kovacs, and A. Brinkmann, "Desktop grids opening up to unicore," in *Proceedings of the UNICORE Summit*, Torun, Poland, 2011, pp. 67–76.