NORDUGRID-TECH-22
18/8/2008

The gLite gateway plugins of ARCLIB and related command line tools

Technical description and user manual

Mattias Ellert[1]
Iván Márton[2]
Péter Stefán[3]

[1] mattias.ellert@fysast.uu.se
[2] martoni@niif.hu
[3] stefan@niif.hu

Contents

1 Introduction

The purpose of this document is to describe gLite [1] specific plugins of ARCLIB [2] and to present the related command line utilities built on top of them. The gateway library together with these tools enables seemless interoparability with gLite computing elements (CE). It is possible, that relevant sections of this document will be moved into the ARC1 user guide and into the ARCLIB technical documentation. Interoperability with gLite storage elements (SE) is provided by another subset of ARC library. This data library provides native interfaces to GridFTP [3] and SRM [4] which are the main bulding blocks of a gLite storage element. Storage interopability is out of scope of this library.

2 Overview

Interoperability has been one of the most important goals in recent grid middleware development. This term is often used in the sense of accessing resources operated by one kind of grid middleware (e.g. gLite) from a user interface operated by another kind (e.g. ARC). There are two main ways to achieve this result:

- either to use a special infrastructure element, the gateway, which performs a full data structure and protocol translation between the two interoperating grid middleware solutions,
- or to explicitly implement the interface of the other side in the client.

The latter solution has been chosen by ARC developers, the gLite gateway library module has been implemented in the ARCLIB and the gateway functionality will be transparently offered via the usual ARC command line tools.

An overview of the implemented soulution is presented in Figure 1. The gLite gateway functionality integrated into the ARCLIB and thus into the ARC client, allows the user to transparently access gLite computing resources via the usual ARC command line tools. The user will be able to express the job requirements in the job description document using any of the xRSL [5], JDL or JSDL [6] languages. The ARC client library will perform an automatic translation. The gLite modules perform authentication, credential delegation and job submission to a CREAM2 [7] CE. CREAM2 CE may perform the necessary input file download from different storage elements. The library module gets job information through CREAM2 and sends management commands, like killing a job or undelegating credentials. Note that not all the functionality described in the overview figure is implemented yet, for the current limitations see Section 6.1.

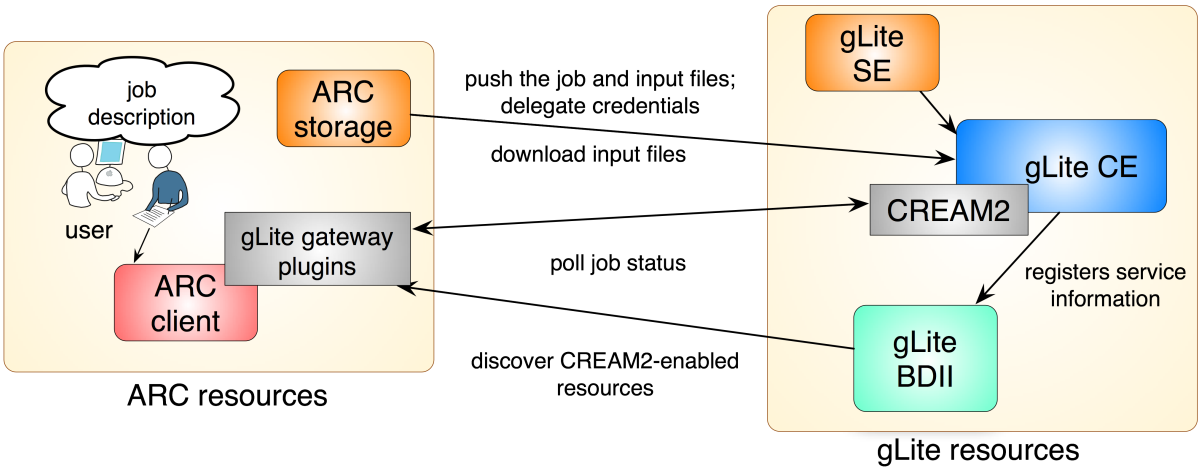

Figure 1: *This chart illustrates how the ARC1→gLite gateway library module based approach works.*

2.1 The supported gLite CE interface: CREAM2

The gLite computing element (CE) offers several interfaces. Some of them are only suitabe for the gLite workload management system and were not meant to be used by 3rd party clients. Fortunately, gLite comes with a Web Service based computing element interface, known as the CREAM CE [7]. The CREAM has undergone several major revisions. The current CREAM version has recently entered the EGEE certification process, nevertheless its full-scale deplomeyment on the EGEE worldwide grid has not taken place yet. CREAM2 is currently available on several preproduction and testing grid sites. KnowARC has chosen the CREAM version 2 (CREAM2) as the target gLite computing element interface.

In particular, the gateway library was developed against version glite-ce-cream_R_1_8_3_0[4].

CREAM2 was chosen for the following reasons:

- CREAM2 has a Web Service interface that fits the Web Service based ARC.
- CREAM2 enables direct access to the gLite computing element without the need of going through the gLite workload management system.
- CREAM2 contains numerous improvements when compared to the earlier CREAM versions.
- Unlike other gLite CE interfaces it supports direct job status queries.
- It is an official interface to gLite computing elements and as EGEE evolves it is expected that more and more computing elements will be deployed with CREAM2 interface.
- CREAM2 offers a convenient way of handling input and output files through accessing the input and output sandbox via GridFTP.

In the near future CREAM will offer yet another interface which will be based on the BES [8], JSDL and GLUE [9] OGF standards. This CREAM BES interface will coexist with CREAM2 too.

3 External dependencies and installation instruction

The ARCLIB gLite plugins have only one dependency: GridFTP components of the Globus Toolkit [10] are needed to enable data staging functionality to those CREAM2 computing elements which offer input and output sandbox over a GridFTP enabled storage. Unfortunately, the gateway library only works together with a strongly patched Globus verion. The NorduGrid patched Globus can be dowloaded from the NorduGrid download area[5]. Further information about the NorduGrid Globus patches can be read on the NorduGrid page[6]. It is worth mentioning that the user would need a VOMS [11] proxy certificate in order to use command line tools against a CREAM2 service. This VOMS proxy can be generated by using standard VOMS client tools. It is only the VOMS enabled proxy certificate that is needed here, apart from this no VOMS components are needed.

The ARC→gLite gateway plugins library and the temporary command line utilities are incorporated into ARC releases. The source code is available from the ARC1 SVN repository[7] and binary packages are provided in the NorduGrid download area. Build and installation instructions are given in the README file distributed together with the software. A step-by-step guide for Debian binary installation can be found in Appendix A *Installation from Debian binary packages*.

4 ARCLIB gLite gateway plugins

The ARCLIB gLite gateway plugins constist of the wrapper classes, the job description handling facility and the gLite specific `CREAMClient` base class. The wrapper classes convey the function provided by the `CREAMClient` to ARCLIB. The job description handling framework makes a general job request representation

[4] `http://grid.pd.infn.it/cream/field.php?n=Main.ReleaseNotes`
[5] `http://download.nordugrid.org/software/globus/`
[6] `http://wiki.nordugrid.org/index.php/Globus_Libraries`
[7] `http://svn.nordugrid.org/repos/nordugrid/arc1/`

available for the other classes. `CREAMClient` base class implements the interface towards CREAM2 computing elements.

This client object has an easy-to-use and intuitive interface. It uses VOMS proxy certificates to build the secure channel to the server and also to sign that of the server.

Different functions throw *CREAMClientError* exceptions that should be caught in the application.

The following sections present the set of functions provided by the library.

4.1 Wrapper classes

The gLite gateway library can be integrated into ARCLIB as a plugin. To expose the CREAM2 functionality into the generic ARCLIB target object functions, the following three wrapper classes have been implemented: the `JobController`, the `Submitter` and the `TargetRetriever`.

The `JobControllerCREAM` wrapper class provides the basic functions related to job management, such as getting job information or killing and cleaning the job on the remote CE.

The `SubmitterCREAM` instantiates the gLite client library and calls the CREAM2-specific submitter function in there.

Class `TargetRetrieverCREAM` performs a gLite BDII database query and extracts possible CREAM2 target objects.

Unfortunately, handling CREAM2 target objects does not perfectly fit into the philosophy of ARCLIB. There are several functions, such as proxy credential delegation or CREAM2-specific job information retrieval, that behave differently on an ARC and on a CREAM2 target. The wrapper classes have to provide solutions for these special cases.

4.2 Job description translators

When addressing interoperability it is of paramount importance to transparently address grid job descriptions written in different job description languages by translating them automatically. In the gLite gateway library code this functionality is implemented in class `JobDescription`. This is a generic class that takes a job description as input in any supported format (currently xRSL, JDL, JSDL), converts and stores it in a JSDL-like internal job description format. Then different operations, like getting the description in other formats or getting job-related information, can be performed using the description-independent functions of this class.

`JobDescription` has tree back-end classes, sometimes are referred to as back-end modules or translator modules, corresponding to the three supported job description languages above. The back-end classes are used for parsing and also for generating the job descriptions. These classes do not store information and are not intended to be used directly, instead, the generic class uses them.

There is a class called `JobDescriptionOrderer` which tries to find out the format of the input file by pattern matching; the generic class uses this piece of information to choose the appropriate back-end.

4.3 CREAMClient base class

The client object uses the VOMS proxy certificate to build the secure channel to the server and also to sign that of the server.

To establish SOAP connection via the secure channel, the library uses the native ARC HED[8] Message Chain Component (MCC), that provides full integration with the ARC core and corresponding services. This replaces the former gSOAP-based [12] communication libraries, thus completely eliminating gSOAP dependency.

[8] `https://www.knowarc.eu/documents/Knowarci_D1.1-1_07.pdf`

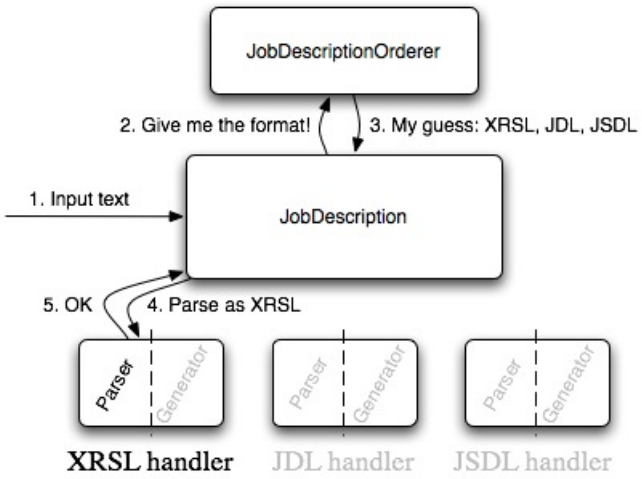

Figure 2: *The figure shows how the different classes participating in the job description translation process interrelate in an example about xRSL parsing.*

CREAMClient(Arc::URL, Arc::MCCConfig)[9]

The constructor of the class performs the necessary initialization work. It receives two arguments: the service URL and the ARC message chain component configuration file to establish the communication channel to the server. These pieces of information are indispensable to create the client and to communicate with the remote site.

setDelegationId(std::string)

Besides the constructor this is the other generic-purpose function that can be used in CREAM2 client applications. This function sets the previously registered and referred delegation ID on the client. The function is simple: it sets a private variable and returns with no value.

createDelegation(std::string)

The `createDelegation` function performs the whole delegation registration process. It sends a *getProxyReq* message having the requested delegation ID, signs the received certificate and sends it back in a *putProxy* SOAP message to the server. These three steps constitute the delegation registration process.

As almost every funtion, this one has no return value either. If there are any problems during the communication, either locally in the channel or at the remote site, the function throws *CREAMClientError* described previously. If there are no exceptions thrown, then the command is considered to have successfully completed.

destroyDelegation(std::string)

This method has the reverse functionality of function `createDelegation`. It sends the *destroy* message of CREAM2 to the remote server.

submit(std::string)

The `submit` function is the most complex part of the class. It translates the job description received as the argument, registers the job with a *JobRegisterRequest* message, uploads the locally stored files, if necessary, then launches the job execution on the server by sending a *JobStartRequest* message. Finally it returns with a `creamJobInfo` object that contains a *jobId* (job indetifier), a *creamURL* (as the service URL), an *ISB*

[9]For class references, see: `http://www.knowarc.eu/download/D1.2-2_documentation.pdf`

(reference to the Input Sandbox) and an *OSB* (Output Sandbox) member to describe the registered job. In the `glitesub` command (See Section 5.2 for details) these pieces of information are stored in the information file.

stat(std::string)

This method queries the job status from the gLite CREAM2 server. The return value is the job status translated to the ARC terminology. The possible values are presented in the section of the `glitestat` command line tool (see Section 5.3).

cancel(std::string)

The `cancel` function sends a *JobCancelRequest* SOAP message to the server and handles the emerging exceptions. It can be used for canceling a remotely registered and possibly running job.

purge(std::string)

The `purge` function sends a *JobPurgeRequest* message to the server and throws an exception in case of any emerging problems.

5 ARC command line tools for gLite CREAM2 CE

The well known ARC commands (`ngsub`, `ngget`, see The NorduGrid/ARC User Guide[13]!) will provide seemless access and integration with gLite computing elements. Full transparency will be achieved through the gLite gateway ARCLIB plugins. An ordinary user will use the same set of commands regardless of what type of CE is accessed. The user will be able to specify grid jobs described both in gLite (JDL) and ARC (JSDL, xRSL) job description languages. The client will perform automatic translation between the different job description languages.

Unfortunately, the interoperability features offered by the gLite gateway library are not yet integrated into the ARC commands. Also, the current version of ARCLIB does not offer the full spectrum of language transformations yet.

Currently, only a set of temporary command line tools exposing the available features of the library are provided. These temporary commands make the already existing interoparabilily features usable for ordinary users.

There are seven different commands for using the gLite gateway functions. In this section these commands will be presented in details through examples. Job descriptions and services shown are just simple examples, their arguments might be different in real-life usage.

Every tool has a short manual page being accessible by using the `man` on-line manual reader command; some more help can be achieved by using the <command> -? command line option.

5.1 glitedelegate

In order to use gLite resources user must have a VOMS proxy. This proxy can be generated by command *voms-proxy-init*.

For example:

```
$ voms-proxy-init -voms knowarc.eu
```

Once a VOMS proxy has been generated, everything is in place to submit a new job. If one wants submit a job to a gLite server, first has to own a valid delegation on the remote site. This can either be an old, but still valid delegation identifier, or can be a new one. There is a command line tool to register a delegation on the CE to be used.

The usage of command **glitedelegate** is straighforward. All needed is a working CREAM2 delegation service URL, and a locally unique arbitrary delegation ID. This delegation ID will be associated to the remote resource. If you are not sure whether your favourite delegation ID is occupied or not, then try to delete it before registering it again (see Section 5.6 for further details). The delegation command has the following syntax:

```
$ glitedelegate <delegation ID> <delegation service URL>
```

For example:

```
$ glitedelegate test_delegation \
        https://cream.grid.upjs.sk:8443/ce-cream/services/gridsite-delegation
```

5.2 glitesub

The current command only supports direct job submission to a known CE, that is, the user must know service endpoints. These kind of endpoints are usually stored in LDAP/BDII databases. Ordinary `ldapsearch` can be used to find such an endpoint. Below comes an example:

```
$ ldapsearch -x -h lxbra2305.cern.ch -p 2170 -b mds-vo-name=local,o=grid \
        '(GlueCEUniqueID=*cream*)' | grep GlueCEInfoContactString | \
        awk '{print $2}' | sort | uniq
```

The **glitesub** command implements the actual job submission in three phases: registration of a job, uploading the local files to the input sandbox URL received during the registration, and starting the job.

The **glitesub** command requires a valid delegation. See Section 5.1 how to create one. By the way, this is the only command which needs the delegation ID.

The next step is to prepare the necessary input files and the job description.

The **glitesub** command syntax is the following (please note that the service URL, job description, info file are mandatory and can only be given in the specified order):

```
$ glitesub -D <delegation ID> <service URL> <job description> <info file>
```

Here the delegation ID is the identification registered previously. The service URL will be different than in the delegation example above, because this is not the URL of the delegation, but the execution service. The job description must be in JDL format like in the example below. The info file stores information about the job submitted. It should be a non-existing file otherwise it will be overwritten. This file contains the job ID, the input sandbox URL, the output sandbox URL and the service URL.

Below is a JDL file example. Note that the *VirtualOrganisation* and *QueueName* attributes should be the same as your virtual organisation name and the corresponding queue name on the server side.

```
[
Executable = "/bin/hostname";
StdInput = "std.in";
StdOutput = "std.out";
StdError = "std.err";
BatchSystem = "pbs";
VirtualOrganisation = "knowarc.eu";
InputSandbox = {"std.in"};
OutputSandbox = {"std.out","std.err"};
QueueName = "knowarc.eu";
OutputSandboxDestURI = { "gsiftp://localhost/std.out", "gsiftp://localhost/std.err" };
]
```

An example how to use this command:

```
$ glitesub -D test_delegation \
       https://cream.grid.upjs.sk:8443/ce-cream/services/CREAM2 description.jdl job.info
```

After successful completion the command reports success and tells the user where the job related data are stored. Otherwise the message written to the output contains the reason of the failure.

5.3 glitestat

The **glitestat** command contacts the CREAM2 service on the service URL and queries status of a job specified by the job ID given in the info file.

Usage:

```
$ glitestat <info file>
```

For example:

```
$ glitestat job.info
```

The **glitesub** command implements job status mapping between gLite and ARC job status semantics. The possible responses and their meanings are the following:

- ACCEPTING - the job submission is completed but the job is not yet scheduled
- SUBMITTING - scheduling in progress
- INLRMS:Q - the job is already at the local resource manager system and it is queued
- INLRMS:R - the job is waiting at the local resource manager system for running
- INLRMS:S - the job is running
- KILLED - the job was terminated
- FINISHED - the job has finished
- FAILED - the job had some failure
- FAILES - the job had some failure at LRMS level
- EXECUTED - the job has been finished and there is no state information available

5.4 glitekill

The **glitekill** command can be used for killing a remote job running in the underlying batch system. The only necessary argument needed is the information file made by **glitesub** (see Section 5.2 for further details). This command initiates stopping the job in the batch system. You can check the effect by using **glitestat** (see Section 5.3 for its usage).

Usage:

```
$ glitekill <info file>
```

For example:

```
$ glitekill job.info
```

5.5 gliteclean

Either after killing a job or because some remote error occurs, garbage files may remain on the server. These files can be removed by using command **gliteclean**. If you intentionally kill a job on a computing node, the job related stuff might also remain as garbage in the remote queue. This command also removes the job from the underlying batch system on the gLite computing element. It is important to note that this command removes the local information file as well. It might be useful to run this command after killing a job (see Section 5.4).

Usage:

```
$ gliteclean <info file>
```

For example:

```
$ gliteclean job.info
```

5.6 gliteundelegate

After the job execution is completed, it is possible to unregister and delete the delegation entry from the remote site by using command **gliteundelegate**. It can also be used when one needs to register a delegation ID but is unsure whether it is already used or not. Put the delegation service URL into the argument!

Usage:

```
$ gliteundelegate <delegation ID> <delegation service URL>
```

For example:

```
$ gliteundelegate test_delegation \
        https://cream.grid.upjs.sk:8443/ce-cream/services/gridsite-delegation
```

5.7 Skeleton of a temporary example gLite client

Finally, here is an example code which shows how easy and simple it is to write a new client using the `CREAMClient` class (see Section 4.3). This code sample requests the job status of a previously submitted remote job, and writes it to the standard output. The client is written in C++.

```
#include "CREAMClient.h"
#include <iostream>

int main(int argc, char* argv[]){
  Arc::URL url( SERVICE_URL );
  Arc::MCCConfig cfg;

    Arc::Cream::CREAMClient gLiteClient(url,cfg);
    try {
        std::cout << "Job status: " << gLiteClient.stat( JOBID ) << std::endl;
    } catch (CREAMClienException& cce) {
        std::cerr << "ERROR: " << cce.what() << std::endl;
        return 1;
    }
    return 0;
}
```

It is of course mandatory to set *SERVICE_URL* and *JOBID* strings to their real values to obtain the proper functionality.

6 Open issues

In order to achieve the full scale transparent interoperability width the gLite computing element offered through the ARC client, a couple of issues still need to be resolved. Most of these are related to the incomplete integration of the gLite gateway library into the ARCLIB framework, even though it is ready for production use. There are also some conceptual challenges due to the incompatible nature of the two systems, which require workarounds.

6.1 Current limitations

Full ARC client integration would mean that the temporary commands developed on top of the `CREAMClient` presented in Section 4.3 will be overridden by the ordinary ARC commands built on top of the ARCLIB. To achieve this, the `Wrapper` class integration of the gLite gateway plugins library should be finalized and exposed to extensive testing.

Some of the `Wrapper` class methods and data structures should be extracted from the information available in the `CREAMClient` class. In particular, detailed job representation needed for job monitoring is an open area.

The `CREAMClient` class will be extended to support operations such as *suspend* available both in the CREAM2 interface and the ARCLIB's jobcontrol component.

Due to the incomplete ARCLIB integration, currently only direct job submission to a manually selected CE is possible. The ARCLIB integration will combine the existing resource discovery (see Section 4.1) and job submission functionality of the gLite gateway plugins with the brokering module of ARCLIB, which is currently under development. The combination of these ARCLIB modules will provide intelligent brokered job submission over gLite and ARC computing element.

For a fully transparent integration, automatic translation among JSDL/xRSL and JDL should also be provided. A user should be able to specify her job request in any of these languages and the gateway should perform the necessary translations. The existing language translator module currently is not capable of generating JDL output. Therefore, the user still needs to use JDL to submit jobs to CREAM2.

Further investigation is needed on how a CREAM2 service handles data staging and whether the data movement related ARC libraries can be used to manage data staging on the client side, or development of special data staging wrapper scripts executed on the computing element is needed.

A set of temporary workaround command line clients have been developed (the glite* commands; see Section 5) due to a discovered OpenSSL version conflict. The conflict appeared when we tried to integrate the `CREAMClient` gateway library into the rest of ARCLIB. A tedious and time-consuming debugging process took place. It revealed the following: a VOMS proxy certificate delegation method was implemented in the `CREAMClient` gateway library. During this process it was needed to sign a certificate and the original OpenSSL was used for this purpose. In the communication library used and taken from the ARC1 framework there is a built-in Message Chain Component (MCC) which already utilizes the Globus Toolkit OpenSSL functions to enable establishing secure communication. Linking the `CREAMClient` library directly against the standard OpenSSL libs caused congesting SSL functions. In order to resolve the OpenSSL conflict and to make possible the complete ARCLIB integration a Globus patch has been worked out. Once the OpenSSL conflict is resolved, the ARCLIB integration will be completed, this way overriding the temporary glite* commands.

6.2 Conceptual problems

During the development of the gateway library a few conceptual incompatibilities were found. These are typical when one of the systems offers richer capabilities not available in the other. These conceptual gaps can only be bridged by developing customized case-by-case workarounds. Still, the interoperation will always result in some information or functionality loss in these cases.

Below are the most important challenges for which solution is still sought:

Expressing file staging in job description

The xRSL offers richer capability to describe input and output data of a job than JDL. Especially, in xRSL it is possible for input files downloaded from storage servers to have their names changed. This is not possible in JDL. So in xRSL user might ask for this:

```
(inputfiles=(gsiftp://interop.dcgc.dk/storage/datafile1.txtinput.txt))
```

This is not possible to express in JDL. It only allows the file to be downloaded as `datafile1.txt`. This could be solved by a wrapper script that renames files according to the xRSL specication. This however does not solve the problem completely. For example, if a user wants to concatenate standard outputs from two jobs, she would write something like the following in xRSL:

```
(inputfiles=(gsiftp://interop.dcgc.dk/storage/job1/output.txtinput1.txt)
            (gsiftp://interop.dcgc.dk/storage/job2/output.txtinput2.txt))
```

This cannot be solved just using the renaming method, because both input files would be retrieved before the renaming and they would therefore overwrite each other.

A possible solution is to recognize jobs of this type and simply barring them from using gLite resources or to let such jobs fail with an error message explaining that the user has to express their job in a different way.

RunTime Environments related issues

Many grid jobs rely on software being available at the execution location. The requirement as well as the initialization of such software is expressed through the use of Run-Time Environments (RTEs). While the library is capable of translating the RTE request itself, it is not capable of translating the name of the RTE. No standardized cross-grid nomenclature has been adopted, nor are the authors aware of any such activity. A temporary solution would be to have a remotely accessible list of RTEs and their names in each job description language. This could then be used for translating jobs.

A Installation from Debian binary packages

This section contains instructions on how to install the gateway libraries and glite* command line tools on Debian Linux.

1. Put the Debian repository into the APT source files (`/etc/apt/sources.list`). This repository in Luebeck is maintained as long as the KnowARC project continues.

```
$ echo "deb http://pc02.inb.uni-luebeck.de:8080/~moeller/debian/stable ./" \
       >> /etc/apt/sources.list
```

2. Update the package list and install the necessary dependencies:

```
$ aptitude update
$ aptitude install voms globus globus-dev globus-doc
```

3. Add the Globus libraries to the system path:

```
$ echo "/opt/globus/lib" > /etc/ld.so.conf.d/globus.conf
$ ldconfig
```

4. Install ARC1 client.

```
$ aptitude install nordugrid-arc1-client
```

5. Create the vomses directory

```
$ mkdir -p /opt/glite/etc/
```

and copy these lines into the vomses file located at /opt/glite/etc/vomses

```
"gin.ggf.org"  "kuiken.nikhef.nl" "15050" "/O=dutchgrid/O=hosts/OU=nikhef.nl/CN=kuiken.nikhef.nl" "gin.ggf.org"
"knowarc.eu"    "arthur.hep.lu.se" "15001" "/O=Grid/O=NorduGrid/CN=host/arthur.hep.lu.se"          "knowarc.eu"
"atlas"         "voms.cern.ch"     "15001" "/DC=ch/DC=cern/OU=computers/CN=voms.cern.ch"           "atlas"
"nordugrid.org" "voms.uninett.no"  "15015" "/O=Grid/O=NorduGrid/CN=host/voms.ndgf.org"          "nordugrid.org"
```

6. Create $HOME/.globus directory and move the user certificate-key pair into there.

References

[1] "gLite, Lightweight Middleware for Grid Computing," Web site. [Online]. Available: http://glite.web.cern.ch/glite/

[2] M. Ellert *et al.*, *WS-based ARC Clients*, The NorduGrid Collaboration, NORDUGRID-TECH-20, document in preparation.

[3] W. Allcock *et al.*, "Data management and transfer in high-performance computational grid environments," *Parallel Comput.*, vol. 28, no. 5, pp. 749–771, 2002.

[4] A. Sim, A. Shoshani and others, "The Storage Resource Manager Interface (SRM) Specification v2.2," May 2008, GFD-R-P.129. [Online]. Available: http://www.ggf.org/documents/GFD.129.pdf

[5] O. Smirnova, *Extended Resource Specification Language*, The NorduGrid Collaboration, NORDUGRID-MANUAL-4. [Online]. Available: http://www.nordugrid.org/documents/xrsl.pdf

[6] A. Anjomshoaa *et al.*, "Job Submission Description Language (JSDL) Specification, Version 1.0 (first errata update)," July 2008, GFD-R.136. [Online]. Available: http://www.gridforum.org/documents/GFD.136.pdf

[7] C. Aiftimiei *et al.*, "Job Submission and Management Through Web Services: the Experience with the CREAM Service," in *Proc. of CHEP 2007, J. Phys.: Conf. Ser. 119 062004*, R. Sobie, R. Tafirout and J. Thomson, Ed. IOP, 2008. [Online]. Available: http://dx.doi.org/10.1088/1742-6596/119/6/062004

[8] I. Foster *et al.*, "OGSA™ Basic Execution Service Version 1.0," August 2007, GFD-R-P.108. [Online]. Available: http://www.ogf.org/documents/GFD.108.pdf

[9] "GLUE Working Group," Web site. [Online]. Available: http://forge.gridforum.org/sf/projects/glue-wg

[10] I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit," *International Journal of Supercomputer Applications*, vol. 11, no. 2, pp. 115–128, 1997, available at: http://www.globus.org.

[11] R. Alfieri *et al.*, "From gridmap-file to VOMS: managing authorization in a Grid environment," *Future Gener. Comput. Syst.*, vol. 21, no. 4, pp. 549–558, 2005.

[12] R.A. van Engelen and others, "gSOAP." [Online]. Available: http://www.cs.fsu.edu/~engelen/soap.html

[13] *The NorduGrid/ARC User Guide*, The NorduGrid Collaboration, NORDUGRID-MANUAL-6. [Online]. Available: http://www.nordugrid.org/documents/userguide.pdf