Enabling Grids for E-science in Europe

Information Society Technologies

Grid Tutorial

GRID MIDDLEWARE

HANDOUTS FOR STUDENTS

Document identifier:	**doc-identifier**
EDMS id:	
Date:	September 1, 2004
Work package:	
Partner(s):	
Lead Partner:	
Document status:	**DRAFT**
Author(s):	Jeff Templon, David Groep, Kors Bos, Fokke Dijkstra, Flavia Donno, Leanne Guy, Mario Reale, Ricardo Rocha, Elisabetta Ronchieri, Massimo Sgaravatto, Heinz & Kurt Stockinger, Antony Wilson, Antonio Delgado Peris, Patricia Méndez Lorenzo, Flavia Donno, Andrea Sciabà, Simone Campana, Roberto Santinelli, Sjors Grijpink
File:	**tutorial**

Abstract: These handouts are provided for people to learn how to use the LCG-2 middleware components to submit jobs on the Grid, manage data files and get information about their jobs and the testbed. It is intended for people who have a basic knowledge of the Linux/UNIX operating system and know basic text editor and shell commands.

CONTENTS

1 INTRODUCTION 4
1.1. CONVENTIONS USED IN THESE HANDOUTS 4
2 GETTING ACCESS TO THE GRID 5
2.1. INTRODUCTION 5
2.2. GETTING A CERTIFICATE 5
2.2.1. WHAT IS A CERTIFICATE? 5
2.2.2. SETTING UP THE AUTHENTICATING ENVIRONMENT 6
2.2.3. EXERCISES 8
2.2.4. GETTING A CERTIFICATE 8
2.2.5. REGISTRATION AUTHORITIES, DO I NEED ONE? 10
2.3. REGISTERING IN A GRID VIRTUAL ORGANISATION 11
2.3.1. REQUESTING YOUR ACCOUNT 11
2.4. REGISTERING IN OTHER VIRTUAL ORGANISATIONS 12
2.4.1. EXCERCISES 12
2.4.2. GETTING A PROXY 13
2.4.3. GETTING THE EXERCISES 13
3 JOB SUBMISSION 15
3.1. INTRODUCTION 15
3.2. EXERCISE JS-1: “HELLO WORLD” 16
3.2.1. INTERMEZZO: THE JOB DESCRIPTION LANGUAGE 20
3.3. EXERCISE JS-2: LIST THE CONTENT OF THE CURRENT DIRECTORY ON THE WORKER NODE; GRID-MAP FILE 21
3.4. EXERCISE JS-3: PING A HOST FROM A NODE; THE SUBMISSION OF SHELL SCRIPTS TO THE GRID 23
3.5. EXERCISE JS-4: RENDERING OF SATELLITE IMAGES USING DEMTOOLS 24
3.6. EXERCISE JS-5: USING POVRAY TO GENERATE VISION RAY-TRACER IMAGES 26
3.7. EXERCISE JS-6: CHECKSUM ON A LARGE INPUT SANDBOX TRANSFERRED FILE 27
3.8. EXERCISE JS-7: A SMALL CASCADE OF “HELLO WORLD” JOBS 28

3.9. EXERCISE JS-8: MPI JOBS 29
3.9.1. THE GRAPHICAL USER INTERFACE 30
4 DATA MANAGEMENT 31
4.1. INTRODUCTION 31
4.1.1. EDG DATA MANAGEMENT TOOLS 33
4.2. EXERCISE DM-1: DISCOVER GRID STORAGE 33
4.3. EXERCISE DM-2: FILE REPLICATION WITH THE EDG REPLICA MANGER 35
4.4. EXERCISE DM-3: USING THE REPLICA CATALOG 38
4.5. EXERCISE DM-4: ACCESSING A GRID FILE FROM A JOB 40
4.6. EXERCISE DM-5: REPLICA OPTIMISATION WITH THE EDG REPLICA MANAGER 43
4.7. EXERCISE DM-6: TAKING A LOOK AT THE .BROKERINFO FILE 45
4.8. EXERCISE DM-7: USE CASE - READ DATA ON THE GRID 47
4.9. EXERCISE DM-8: USE CASE - COPY AND REGISTER JOB OUTPUT DATA 48
5 INFORMATION SYSTEM 49
5.1. INTRODUCTION 49
5.2. THE LOCAL GRIS 50
5.3. THE SITE GIIS 52
5.4. THE BDII 54
A THE GLUE SCHEMA 59
A.1. ATTRIBUTES FOR THE COMPUTING ELEMENT 59
A.2. ATTRIBUTES FOR THE STORAGE ELEMENT 63
A.3. ATTRIBUTES FOR THE CE-SE BINDING 65
B JOB STATUS DEFINITION 66

CHAPTER 1

INTRODUCTION

This document leads you through a number of increasingly sophisticated exercises covering aspects of job submission, data management and information systems. It is assumed that you are familiar with the basic Linux/UNIX user environment (bash, shell etc.) and that you have obtained a security certificate providing access to the LCG-2 testbed. This document is designed to be accompanied by a series of presentations providing a general overview of Grids and the LCG tools. Solutions to all the exercises are available online. We do not give exact host names of machines in the testbed since they change over time.

1.1. CONVENTIONS USED IN THESE HANDOUTS

The following conventions are used in these handouts [1]:

Bold	is used for statements and functions, identifiers, and program names.
italic	is used for file and directory names when they appear in the body of a paragraph as well as for data types and to emphasise new terms and concepts when they are introduced.
`Constant Width`	is used in examples to show the contents of files or the output from commands.
`Constant Bold`	is used in examples to show command lines and options that should be types literally by the user. (For example, **`rm foo`** means to type "rm foo" exactly as it appears in the text or the example.)
""	are used to identify a code fragment in explanatory text. System messages and symbols are quoted as well.
$	is the UNIX shell prompt.
<>	surrounds optional elements in a description of program syntax. (The brackets themselves should never be typed, unless otherwise noted.)
. . .	stands for text (usually computer output) that's been omitted for clarity or to save space.

[1] See D. Dougherty and A. Robbins, *sed & awk, Second Edition*. O'Reilly & Associates, Inc., 1997, 1990.

CHAPTER 2

GETTING ACCESS TO THE GRID

2.1. INTRODUCTION

2.2. GETTING A CERTIFICATE

2.2.1. WHAT IS A CERTIFICATE?

While you are using computer systems that are scattered all over the world, the administrators of all those machines will want to know who is using their machines and storage. In the past, you had to contact each site administrator separately, and you would get a username and a password for every new site. By providing this combination, the administrator could be sure who was using the system. But the user was obliged to remember as many passwords as there were sites. This cumbersome way of working is not suitable for the Grid, where you will be accessing many different sites without you even knowing.

On the Grid, you will be using a certificate. This certificate binds together your identity (name, affiliation, etc.) and a unique piece of digital data called a public key that is explained below. A third party that is trusted by all sites in the LCG-2 test bed digitally signs the combination of your name and the public key.

The use of a public key to authenticate yourself is based on a special mathematical trick, called *asymmetric cryptography*. If you would pick two large (prime) numbers and multiply them, it is virtually impossible to factorise the product into the two numbers again. The individual prime numbers are used to generate an encryption and a decryption function and the product of the two, and then the two numbers are destroyed. If you only have the encryption function, it is impossible to derive the decryption functions from it (and vice versa). So, if you distribute the encryption function called public key widely (e.g. you put it on the web) but keep the decryption function private, everyone can send you encrypted messages, but only you can read them and even the sender cannot get the message back!

This method is quite useful if you want to authenticate yourself to a remote site without revealing any personal information: if the remote site knows your public key, it can encrypt a challenge (e.g. a random number) using this key and ask you to decrypt it. If you can, you obviously own the private key and therefore you are who you say you are but still the remote site has to know all the public keys of every one of its customers.

It all becomes simpler if we introduce a trusted third party, a human that can authenticate people in persons called a *Certification Authority (CA)*. When you go to a CA you bring along your public key and an identifier your full name and possibly an affiliation. Now the CA has to make sure by some other means that you are indeed who you claim to be. The CA may ask for a passport or drivers license, it

could contact your boss to verify your affiliation, make a phone call to your office, etc. When the CA is reasonably convinced of your identity, it will take your public key and your identifier and put those together in a certificate. As a proof of authentication, the CA will then calculate a digest (hash) of the combination of the two and encrypt it with the private key of the CA. Everyone can recalculate the digest, decrypt the signature using the public key of the CA and verify that these two are the same. If you show up at a remote site that only knows your name (identifier) and trust the CA that you got your certificate from, the site known that whoever can decrypt the challenge sent corresponds to the name they have in their list of allowed users.

2.2.2. SETTING UP THE AUTHENTICATING ENVIRONMENT

In reality, applying for a certificate may take you a day of two remember that it requires action by real human beings. For that reason certificates have already been generated for you for use during this tutorial. The only thing you have to do is get it and install it in the proper directory.

In this tutorial you will be working from a *User Interface (UI)*. So, first you have to login to the UI (*ui.matrix.sara.nl*). In the information map you can find the user name and password for login in to the UI of the LCG-2 Matrix cluster at SARA, e.g.:

```
$ ssh demo39@ui.matrix.sara.nl
demo39@ui.matrix.sara.nl's password:
Last login: Tue May 25 16:02:05 2004 from aude.nikhef.nl
*****************************************************************************
Welcome to the SARA NL-Grid Matrix User interface
- For information on use see http://www.sara.nl
- If you have problems or questions please contact grid.support@sara.nl
*****************************************************************************

The Matrix cluster has just been upgraded to the LCG-2 software, and
the cluster is operational again.
$ ls
Grijpink-Sjors
```

In your home directory you see a directory with your name, containing the files produced by the certificate generation process:

```
$ cd Grijpink-Sjors/
$ ls -la
total 32
drwxr-xr-x    2 demo39   demo          4096 May 25 14:33 .
drwx------    3 demo39   demo          4096 May 25 17:30 ..
-rw-r--r--    1 demo39   demo          2451 May 25 14:33 020cf596-c437ca
-rw-r--r--    1 demo39   demo           244 May 25 14:33 certreq6499.cnf
-rw-r--r--    1 demo39   demo          2451 May 25 14:33 certreq6499.txt
-r--------    1 demo39   demo            39 May 25 14:33 pw.txt
-r--------    1 demo39   demo           951 May 25 14:33 userkey.pem
-rw-r--r--    1 demo39   demo          2064 May 25 14:33 userrequest.pem
```

Note the protection set on your private key file *userkey.pem*. They are very restrictive and are set thus for a reason: your possession of the private key is the only proof remote sites have that they are indeed taking to you. If you would give that key to someone else (or if it gets stolen), you will be held liable for any damage that may be done to the remote site! In any case, if the user key is world readable or worse, is cannot be used by the Grid.

The private key is also be protected with a pass phrase (a difficult name for a password of arbitrary length). You can find the password in the file *pw.txt*. You can change the pass phrase anytime you like.

Now you can install the file *userkey.pem* in a directory where it can be found with the LCG-2 management tools. This directory is the *.globus* directory and residents in your home directory:

```
$ cd ~
$ mkdir .globus
$ cp -p Grijpink-Sjors/userkey.pem ~/.globus/
$ cp -p Grijpink-Sjors/userrequest.pem ~/.globus/
$ ls -la ~/.globus/
total 12
drwxr-xr-x    2 demo39   demo          4096 May 25 17:41 .
drwx------    4 demo39   demo          4096 May 25 17:41 ..
-r--------    1 demo39   demo           951 May 25 14:33 userkey.pem
-rw-r--r--    1 demo39   demo          2064 May 25 14:33 userrequest.pem
```

You can obtain your certificate from the following webpage:

```
http://certificate.nikhef.nl/medium/certlist.html
```

Change to the *.globus* directory and install your usercertificate:

```
$ cd ~/.globus/
$ wget http://certificate.nikhef.nl/medium/details-16da7552/newcerts/0194.p
em
--17:50:23--  http://certificate.nikhef.nl/medium/details-16da7552/newcerts/
0194.pem
           => `0194.pem'
Resolving certificate.nikhef.nl... done.
Connecting to certificate.nikhef.nl[192.16.185.28]:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 5,071 [text/plain]

100%[===================================================>] 5,071
2.42M/s    ETA 00:00

17:50:23 (2.42 MB/s) - `0194.pem' saved [5071/5071]

$ mv 0194.pem usercert.pem
$ ls -la
total 20
drwxr-xr-x    2 demo39   demo          4096 May 25 17:50 .
drwx------    4 demo39   demo          4096 May 25 17:41 ..
-rw-r--r--    1 demo39   demo          5071 May 25 10:06 usercert.pem
-r--------    1 demo39   demo           951 May 25 14:33 userkey.pem
-rw-r--r--    1 demo39   demo          2064 May 25 14:33 userrequest.pem
```

You can always see what is in a certificate using the **openssl** command. This is a toolkit for handling certificates, keys and requests. The table below lists a few useful commands:

show the contents of a certificate:

```
openssl x509 -text -noout -in <usercert.pem>
```

show the contents of a certificate request:

```
openssl req -text -noout -in <userrequest.pem>
```

writes a new copy of the private key with a new pass phrase:

```
openssl rsa -in private_key_file -des3 -out new_private_key_file
```

In principal you are done now to start with the exercises for working with the Grid (e.g. job submission, data management . . .). But the certificates you have obtained for this tutorial are only useful for the duration of the tutorial (plus some extra days). In reality you have to make a request for a certificate and register with a *Virtual Organisation (VO)*. So, the next sections will show you how to do this in order to familiarise you with these procedures.

Important: The "default" directory for your certificates is *$HOME/.globus*. Since there is now a certificate in it, you would overwrite the files whilst doing the upcoming exercises. In order to make sure that that posses no problems you should copy the *.globus* to a new directory:

```
$ cp -rp ~/.globus ~/.globus.original
```

Whenever you now feel like you messed up the directory, you can recover by:

```
$ rm -f ~/.globus/usercert.pem
$ rm -f ~/.globus/userkey.pem
$ rm -f ~/.globus/userrequest.pem
$ cp -p ~/.globus.original/* ~/.globus/
```

2.2.3. EXERCISES

1. Look in your certificate directory, and look inside your certificate using the openssl command. What is your subject name?

2. Make sure that the files in your .globus.original directory are the same as in your .globus directory afterwards.

3. Remove the files in your .globus directory and copy the original ones from .globus.original.

4. Store your tutorial certificate in the .globus directory and try the exercises in the section Getting a Proxy . Remember to come back here.

2.2.4. GETTING A CERTIFICATE

This section will try to familiarise you with the procedure of making a certificate request. For the tutorial there have already been certificates being created for you, but these are only valid for the duration of the tutorial. In this section we will show you how to request for a certificate useful in *real life*, and in the exercises let you request a dummy certificate from the Grid Certification Authority The exact procedure is different for every CA and there is one per country. For real life regular use of the Grid from the Netherlands, you need a *medium-security* CA certificate from the DutchGrid CA. For use with the national grid projects and the EU projects, DutchGrid is running the CA. The web site for this CA is http://www.dutchgrid.nl/ca and on this page you find a link to a web form that will help you to generate a certificate request as shown in Figure 2.1 (nearly all CAs have such a web form). When you fill all information and make your way through the certification details, you can in the end download a shell script that you can run on the user interface machine. The shell script is called *makerequest.sh* by default and is usually written to your home directory.

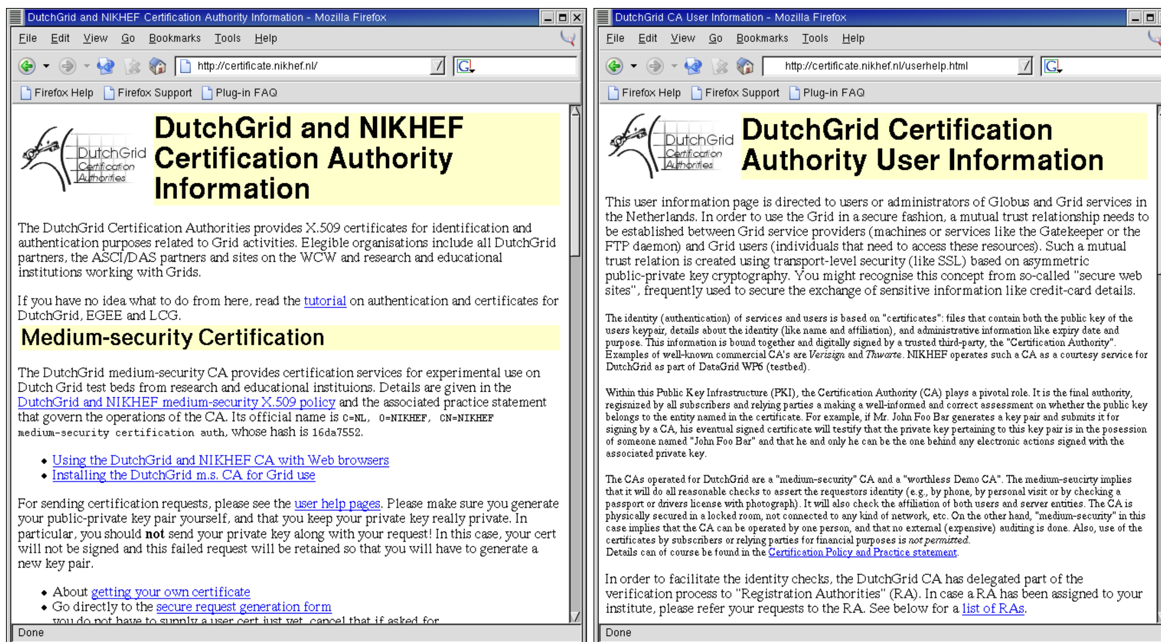

Figure 2.1: The NIKHEF CA webpage (left), the user help webpage (right).

The direct link to the user request pages is at http://certificate.nikhef.nl/userhelp.html, read the document and fill the request forms similar to the one shown above (Figure 2.1).

When you run the shell script (run it only once!), it will generate a new, unique public and private key and write a certificate request to a file in your *.globus* directory. It is this request that you have to submit to the CA for certification. A regular certificate request is mailed automatically to the CA, so make sure that your machine can actually send mail.

If for some reason you cannot send mail directly, copy-and-paste the file *certreqXXX.txt* into your favourite mail client and send the mail to <ca@dutchgrid.nl>. The mail looks like this:

```
Certificate request for medium certification

  From: David Groep
  Email address: davidg@nikhef.nl
  Contact info: NIKHEF, room H157, Kruislaan 409, Amsterdam, +31 20 592 2179
  Date: 20031120-1020
  Dir: .
  Pwd: /user/davidg/.globus-lcg

Certificate Request:
    Data:
        Version: 0 (0x0)
        Subject: O=dutchgrid, O=users, O=nikhef, CN=David Groep

...
aPznCjlI0WAUCrnP47Hj+P5RTx9PVZNTA95H0B/foF1HwXL46wfkwc4Y8QqGAcuG
B99JoIZx9ZXGVAwYb7eU1r2sl3VC8fsCh6PwWX4gMy6wFl9xlCL9EpFI+wLzC/oK
6fE3EQm+oEqA489G0FwHjlWnFFFFaA==
-----END CERTIFICATE REQUEST-----
```

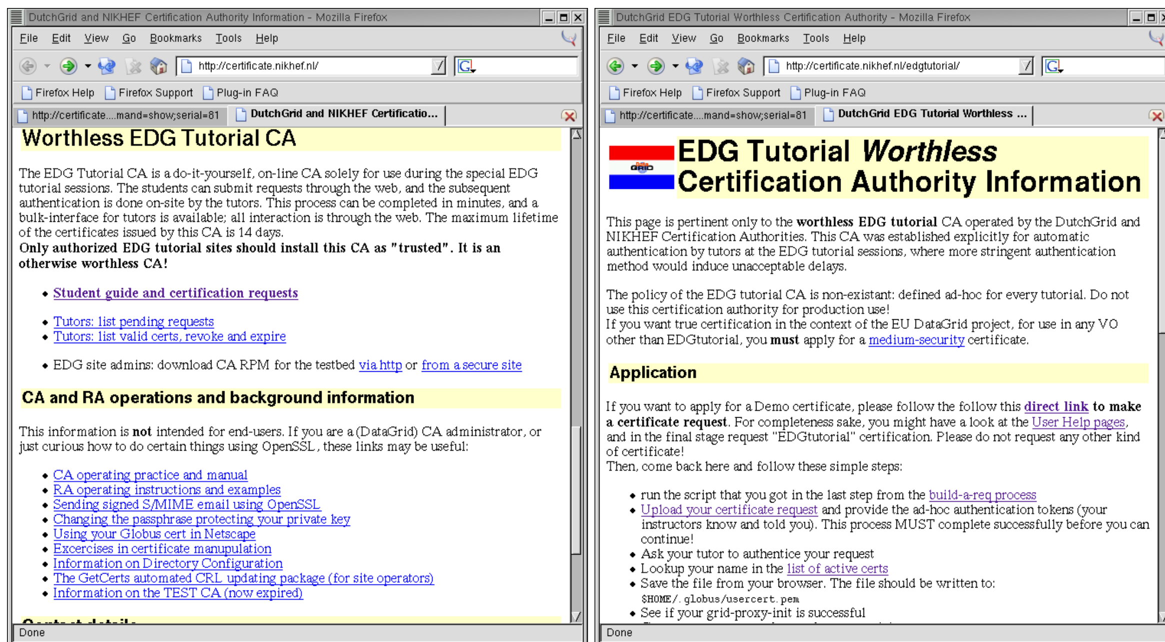

Figure 2.2: Webpages for applying for a Demo certificate.

After a short while, you get a certificate back from the CA. Some CAs send the certificate by e-mail to you, others request you retrieve it yourself from a web site. The normal DutchGrid CA will mail it back to you, but the Tutorial CA wants you to retrieve it yourself from a web site. In any case, you store it in a file called *usercert.pem*, in the same directory where you found the *userrequest.pem* file. If you lost the mail, go to the web address and download your certificate:

```
http://certificate.nikhef.nl/medium/certlist.html
```

It does not matter how much bogus is in this file, as long as you keep the fragment between `BEGIN CERTIFICATE` and `END CERTIFICATE` intact:

```
-----BEGIN CERTIFICATE-----
MIIE1DCCA7ygAwIBAgIBQjANBgkqhkiG9w0BAQQFADBSMQswCQYDVQQGEwJOTDEP
MA0GA1UEChMGTklLSEVGMTIwMAYDVQQDEylOSUtIRUYgbWVkaXVtLXNlY3VyaXR5

YjNS8HW/xZ+BvK0hHiIneVccvotJhl35u/qITZK0ExehHIu4UTr1YgaYxOpieIbg
wzUZncH+lVaDME4JcFAOgc5xrA5q+RJeLg8rmbtTvViiK7VEZxyOeg==
-----END CERTIFICATE-----
```

2.2.5. Registration Authorities, do I need one?

For large CAs, it is very difficult to contact everyone personally. Therefore, the task of authenticating people has been devolved onto *Registration Authorities (RA)*s. Like a CA, a RA is a real person, maybe the head of your personnel department, or your team leader. The RAs do not sign certificates themselves, but tell a CA that a particular person belongs to a particular certificate and that they should sign the request. The task of an RA is simple, and many RAs can be appointed for one CA. On the other hand, running a proper CA is a complex task, requiring a secure environment and personnel.

When you request a certificate via the web, you may have to specify which RA is closest to you. When you upload your certificate request using your web browser, you also select the RA for your own lab.

EXERCISE

1. Apply for a Demo certificate using the Worthless EDG Tutorial CA (See Figure 2.2):
   `http://certificate.nikhef.nl/`
   click on: **Student guide and certification requests**
2. Follow the steps laid out on (See Figure 2.2):
   `http://certificate.nikhef.nl/edgtutorial/`
3. Check if your certificate appears in the "list of active certs".

2.3. Registering in a Grid Virtual Organisation

If you want to use the EGEE or DataGrid Grid for real, you should register with a Virtual Organisation (VO). This may be your experiment (LHCb, Babar) or your community (dteam, EarthOb). Also you thereby agree to the Acceptable Use Policy (of course you do, but realise that you are now legally responsible for your actions :-). To do this, you must authenticate with your certificate to a web site, and thus you would have your certificate available inside your web browser.

The file you have on disk is suitable for Grid use, but needs to be converted to a different format for web browsers. This format is called `PKCS#12`, and files have the extension *.p12*. This format is special in the sense that the file contains both your public and your private key, and the combination is again protected with a pass phrase (here called export password).

The **openssl** programme is again used to convert between the different formats:

```
$ cd $HOME/.globus
$ openssl pkcs12 -export -in usercert.pem -inkey userkey.pem \
> -out packed-cert.p12
```

The file *packed-cert.p12* now contains both your certificate and your private key, and can be imported in Mozilla or Internet Explorer in this tutorial we will use Mozilla (also installed on the UI), but Internet Explorer will work as well. The certificate in Mozilla can be reached from:
`Edit -> Preferences -> Privacy & Security -> Certificates -> Manage Certificates`
In the `Certificate Manager` You can now import your certificate by pressing the import certificate button.

Mozilla will protect its certificate store with a password as well. Enter a good password in the dialogue. In the file browser window you will subsequently get, go to your .globus directory and select the packed-cert.p12 file. Again, you will have to provide a password, this time the export password you gave to **openssl** when you created the `PKCS#12` file. You also have to think of a nickname for this identity. We suggest you use your username on the User Interface machine. You have now successfully imported your certificate and you can close the Mozilla security window.

2.3.1. Requesting your account

You are now ready to sign the Guidelines and apply for an account. You can get to the registration page from the main LCG web site `http://lcg-registrar.cern.ch/` and selecting Registration, or go directly to (see Figure 2.3):

```
https://lcg-registrar.cern.ch/cgi-bin/register/account.pl
```

Or if you are part of “NL-Grid” you can go to (see Figure 2.3):
`https://register.matrix.sara.nl/cgi-bin/register.py`

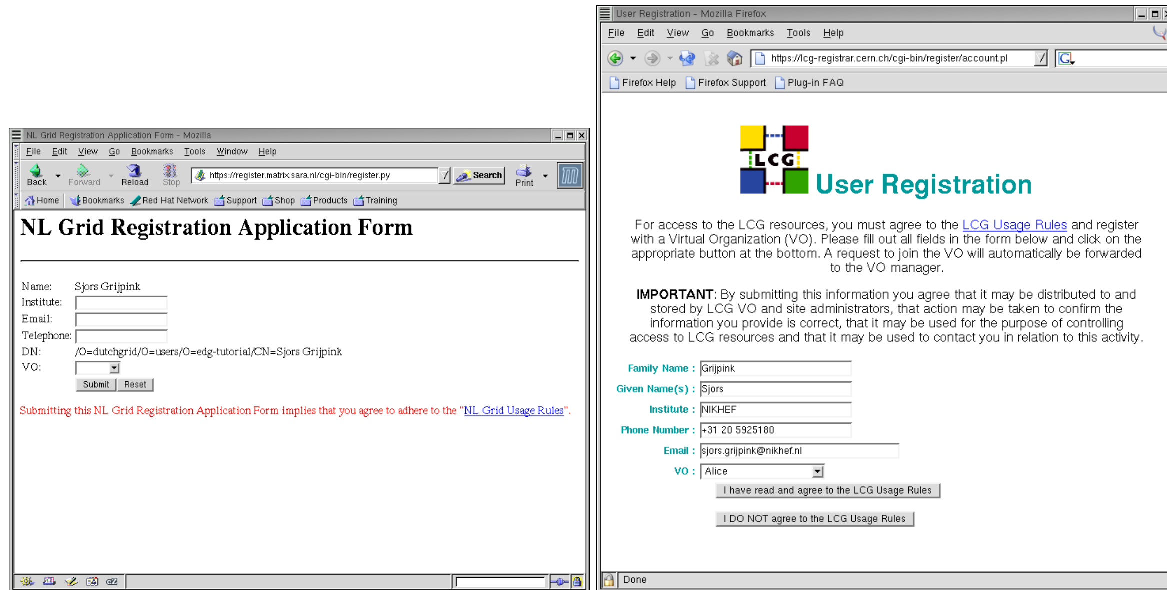

Figure 2.3: NL Grid registration form (left), LCG registration form (right)

Press OK whenever asked (the web site is protected with a certificate from the CERN CA, which is not recognised by default in Mozilla). Using your personal certificate, you can authenticate to the web site.

You will see that all the data from your certificate is already filled in. In real life, you would now have selected your affiliation to the Grid project you belong to and apply for a real account... but that has already been done for you by the tutors, so do NOT press on the agree button but instead quit your browser.

2.4. Registering in other Virtual Organisations

The web registration above only applies for EGEE, EDG and LCG. In you joined a national project like VL-E, please contact the VO directly, e.g., using the mail address given on the project web site.

For VL-E, use the DutchGrid Support system, for the time being, at

`http://www.dutchgrid.nl/Support/`

`<support@dutchgrid.nl>`

2.4.1. Excercises

1. Convert your certificate and private key into a `PKCS#12` file.
2. Can you get to the VO registration page?

2.4.2. GETTING A PROXY

When you have a certificate you can now request a key to be allowed to do the exercises that follow in manual. The key you will get will be valid for several hours, long enough for a hands-on afternoon at least. First you have to get onto a machine that understands grid commands. Such computers are called the User Interface (UI) machines and you may have one in your own home institute for which you have an account. In any case, your tutorial organizer should have provided an account for you to use during the course. Your instructor will tell you which account you can use and what your password is. Now one can get a ticket that allows you to use the testbed. The following commands are available:

grid-proxy-init	to get a key, a pass phrase will be required
grid-proxy-info	all gives information of the ticket in use
grid-proxy-destroy	destroys the ticket for this session
grid-proxy-xxx -help	shows the usage of the command grid-proxy-xxx

Examples:

```
$ grid-proxy-init
Your identity: /O=dutchgrid/O=users/O=nikhef/CN=Sjors Grijpink
Enter GRID pass phrase for this identity:
Creating proxy ...................................... Done
Your proxy is valid until: Fri May 14 23:33:50 2004
$ grid-proxy-info -all
subject   : /O=dutchgrid/O=users/O=nikhef/CN=Sjors Grijpink/CN=proxy
issuer    : /O=dutchgrid/O=users/O=nikhef/CN=Sjors Grijpink
type      : full
strength  : 512 bits
path      : /tmp/x509up_u556
timeleft  : 11:59:40
$ grid-proxy-destroy -dryrun
Would remove /tmp/x509up_u556
```

2.4.3. GETTING THE EXERCISES

Some material for the exercises has been prepared in advance and you can copy it (e.g. with `wget`) to your home directory on the UI machine from:
http://www.nikhef.nl/~h19/exercises.tgz

Example of what you may see on the screen:

```
$ wget http://www.nikhef.nl/~h19/exercises.tgz
--09:44:01--  http://www.nikhef.nl/%7Eh19/exercises.tgz
           => `exercises.tgz.1'
Resolving www.nikhef.nl... done.
Connecting to www.nikhef.nl[192.16.199.5]:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1,549,148 [application/x-tar]

100%[================================>] 1,549,148     11.28M/s    ETA 00:00

09:44:01 (11.28 MB/s) - `exercises.tgz' saved [1549148/1549148]
```

```
$ ls
exercises.tgz
$ tar zxvf exercises.tgz
DMexercise4/
DMexercise4/values
DMexercise4/gsiftp.jdl
[...]
JSexercise7/
JSexercise7/HelloWorld.jdl
JSexercise7/submitter.sh
```

CHAPTER 3

JOB SUBMISSION

3.1. INTRODUCTION

The components of the *Workload Management System (WMS)* are shown in Figure 3.1. It consist of a *User Interface (UI)*, a *Resource Broker (RB)* with an associated *Information Index (II)* and a *Job Submission System (JSS)*, the *Globus Gatekeeper (GK)* with its associated *Local Resource Management System (LRMS)*, a *Worker Node (WN)* and a *Logging and Bookkeeping (LB)* system. The Resource Broker, the Job Submission System and the Logging and Bookkeeping system are central services in the Grid and do not have to be geographically at the same place as the user and/or the User Interface machine. The Gatekeeper and the Local Resource Management System are services that each center providing compute and storage resources to the grid will have. On the current testbed these are the particle physics laboratories like CNAF in Bologna or NIKHEF in Amsterdam, just to name two of them. Logically the Gatekeeper, the Local Resource Management System and the Worker Nodes are called a *Computing Element (CE)*. As depicted in Figure 3.1 the step to run a job are:

1. User submits the job from the UI to the RB. The RB does the matchmaking to find out where the job might be executed. After having found such a Computing Element the Job is transferred to the Job Submission System. At the JSS a file is created in *Resource Specification Language (RSL)*. Also at this stage the *Input SandBox* is created in which all files are specified that are needed by the job.
2. This RSL file, together with the Input SandBox, is then transferred to the Gatekeeper of the Computing Element and the Gatekeeper submits the job to the Local Resource Management System.
3. The LRMS will then send the job to one of the free Worker Nodes of the Computing Element.
4. When the job has finished, the files produced by the job are available on the LRMS. The job manager running on the CE notifies the Resource Broker that the job has completed.
5. The RB subsequently retrieves those files specified in the *OutputSandBox*.
6. The RB sends the results (the OutputSandBox) back to the user on the User Interface machine.
7. Queries by the user on the status of the job are sent to the Logging and Bookkeeping Service.

Users access the GRID through a UI machine that – by means of a set of Python scripts allows the user to submit a job, monitor its status, and retrieve the output from the worker node back to a local directory

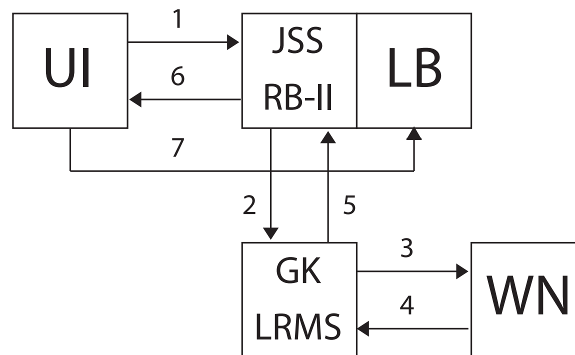

Figure 3.1: The main Work Load Management System (WMS) components and their operation.

on the UI machine. To do so, a simple *Job Description Language (JDL)* file is compiled. In this file all parameters to run the job are specified.

The relevant commands for submitting a job, querying its status, retrieving the output, and cancelling a job are:

edg-job-submit <job.jdl>	submits a job for which the description is in *job.jdl*
edg-job-status <jobId>	returns the status of a job with job identifier jobId
edg-job-get-output <jobId>	returns the place where tho output of the job can be found
edg-job-cancel <jobId>	cancels the job with job identifier jobId
edg-job-xxx --help	shows the usage of command **edg-job-xxx**

In the next sections a series of exercises will be presented to familiarise you with the WMS of the Grid.

3.2. Exercise JS-1: "Hello World"

In this example you will run a simple "Hello World" job on the Grid. The job is described in the Job Description Language (JDL) in the *HelloWorld.jdl* file, which is in the *JSexersise1* directory [1]:

```
[JSexercise1]$ cat HelloWorld.jdl
Executable = "/bin/echo";
Arguments = "Hello World";
Stdoutput = "message.txt";
StdError = "stderror";
OutputSandbox = {"message.txt","stderror"};
```

This is close to the minimum job description to be able to run a job on the grid. The Executable in this case is a simple Unix **echo** command and the Argument to this command is the "Hello World" text. You have to specify at least two files: an output file and a file where the possible error messages go. The OutputSandbox contains the files that will go with the job to the node where the program will be executed and migrated back to the user when the execution has finished.

[1] To obtain the exercises and directories see 2.4.3..

EXCERCISE

1. Run this job on the grid using the **edg-job-submit** command.
2. Read and try to understand the output on the screen.
3. Request the status of the job using the **edg-job-status** command.
4. Get the output from this job using the **edg-job-get-output** command when in the Output Ready state.
5. Check that the run has run correctly by looking into the *message.txt* and *stderror* files.

To submit the "Hello World" job to the LCG-2 Grid, you must have a valid proxy certificate in the User Interface machine. Obtain one by issuing:

```
$ grid-proxy-init
```

and submit the job:

```
$ edg-job-submit HelloWorld.jdl
```

If the submission is successful, the output is similar to:

```
Selected Virtual Organisation name (from UI conf file): alice
Connecting to host boswachter.nikhef.nl, port 7772
Logging to host bosheks.nikhef.nl, port 9002


*********************************************************************************************
                               JOB SUBMIT OUTCOME
 The job has been successfully submitted to the Network Server.
 Use edg-job-status command to check job current status. Your job identifier
 (edg_jobId) is:

 - https://boswachter.nikhef.nl:9000/lceOtWQxqrECtTH6LQB2tw


*********************************************************************************************
```

In case of failure, an error message will be displayed instead, and an exit status different from zero returned.

The command returns to the user the job identifier (*jobId*), which defines uniquely the job and can be used to perform further operations on the job, like interrogating the system about its status, or cancelling it. The format of the jobId is:

```
https://Lbserver_address[:port]/unique_string
```

where `unique_string` is guaranteed to be unique and `Lbserver_address` is the address of the Logging and Bookkeeping server for the job, and usually (but not necessarily) is also the Resource Broker.

Note: The jobId does NOT identify a web page.

After a job is submitted, it is possible to see its status and its history, and to retrieve logging information about it. Once the job is finished the jobs output can be retrieved, although it is also possible to cancel it previously. The status information of our “Hello World” job can now be obtained by issuing:

```
$ edg-job-status https://boswachter.nikhef.nl:9000/lceOtWQxqrECtTH6LQB2tw
```

a possible output could then be:

```
*************************************************************
BOOKKEEPING INFORMATION:

Status info for the Job : https://boswachter.nikhef.nl:9000/lceOtWQxqrECtTH6
Current Status:     Scheduled
Status Reason:      Job successfully submitted to Globus
Destination:        lcgce01.triumf.ca:2119/jobmanager-lcgpbs-long
reached on:         Wed May 19 10:22:26 2004
*************************************************************
```

where the current status of the job is showed, along with the time when that status was reached, and the reason for being in that state (which may be especially helpful for the **ABORTED** state). The possible job states are summarised in Appendix B. Finally, the **destination** field contains the ID of the CE where the job has been submited.

Note: Much more information is provided if the verbosity level is increased by using **-v1** or **-v2** with the command. See [R6] for detailed information on each of the fields that are returned then.

After our job has finished (it reaches the **Done** status), its output can be copied to the UI with the command **edg-job-get-output**:

```
$ edg-job-get-output https://boswachter.nikhef.nl:9000/lceOtWQxqrECtTH6LQB2t

Retrieving files from host: boswachter.nikhef.nl ( for https://boswachter.ni
khef.nl:9000/lceOtWQxqrECtTH6LQB2tw )

****************************************************************************
                        JOB GET OUTPUT OUTCOME

 Output sandbox files for the job:
 - https://boswachter.nikhef.nl:9000/lceOtWQxqrECtTH6LQB2tw
 have been successfully retrieved and stored in the directory:
 /tmp/jobOutput/h19_lceOtWQxqrECtTH6LQB2tw

****************************************************************************
```

By default, the output is stored under */tmp/jobOutput*, but it is possible to specify in which directory to save the output using the −**dir** <**path_name**> option.

MORE ON EXERCISE JS-1: "HELLO WORLD" ON A DIFFERENT SOURCE

In this exersise you will run the same HelloWorld job but now on a pre-selected site. There exists a command that returns the result from the match making job the RB does on the basis of the job description in JDL file. From the list of Computing Elements that could run your job you can select one and use one of the options of the job submission command to send your job to that site.

The relevant commands for this exercise are:

edg-job-submit --help	to find out all options of the job-submit command
edg-job-list-match <job.jdl>	returns the CEs where the job could run

EXCERCISE

1. Find out which option of the **edg-job-submit** command you can use to choose your favorite CE to run your job.
2. Find out were your job could possibly run by using the **edg-job-list-match** command.
3. Choose your favourite CE and submit the "HelloWorld.jdl" job to this site.
4. Check the status of your job and verify your job was indeed run at the site of your choice.
5. When the data is ready, get your output back and check that the job was executed correctly.

It is possible to see which CEs are eligible to run a job specified by a given JDL file using the command **edg-job-list-match**:

```
$ edg-job-list-match HelloWorld.jdl

Selected Virtual Organisation name (from UI conf file): alice
Connecting to host boswachter.nikhef.nl, port 7772

***************************************************************************
                         COMPUTING ELEMENT IDs LIST
 The following CE(s) matching your job requirements have been found:

                   *CEId*
 farm012.hep.phy.cam.ac.uk:2119/jobmanager-lcgpbs-Lq
 farm012.hep.phy.cam.ac.uk:2119/jobmanager-lcgpbs-Mq
 farm012.hep.phy.cam.ac.uk:2119/jobmanager-lcgpbs-Sq
 ...
 zeus02.cyf-kr.edu.pl:2119/jobmanager-lcgpbs-long
 zeus02.cyf-kr.edu.pl:2119/jobmanager-lcgpbs-short
 tbn18.nikhef.nl:2119/jobmanager-pbs-qlong
***************************************************************************
```

The `-r <CE>` option is used to directly send a job to a particular CE. The drawback is that the BrokerInfo functionality will not be carried out. That is, the BrokerInfo file, which provides information about the evolution of the job, will not be created. The CE is identified by `<CE>`, which is a string with the following format:

```
<full_hostname>:<port_number>/jobmanager-<service>-<queue\name>
```

where `<full_hostname>` and `<port>` are the hostname of the machine and the port where the Globus Gatekeeper is running, `<queue_name>` is the name of one of the queue of jobs available in that CE, and the `<service>` could refer to the LRMS, such as (*lsf*, *pbs*, *condor*), but can also be a different string as it is freely set by the site administrator when the queue is set-up.

3.2.1. INTERMEZZO: THE JOB DESCRIPTION LANGUAGE

In LCG-2, job description files (*.jdl* files) are used to describe jobs for execution on the Grid. These files are written using a Job Description Language (JDL). The JDL adopted within the LCG-2 Grid is the *Classified Advertisement (ClassAd) language*[R12] defined by the *Condor Project* [R13], which deals with the management of distributed computing environments, and whose central construct is the *ClassAd*, a record-like structure composed of a finite number of distinct attribute names mapped to expressions. A ClassAd is a highly flexible and extensible data model that can be used to represent arbitrary services and constraints on their allocation. The JDL is used in LCG-2 to specify the desired job characteristics and constraints, which are used by the match-making process to select the resources that the job can use.

The fundamentals of the JDL are given in this section. A detailed description of the JDL syntax is outside the scope of this handout, and can be found in [R14] and [R15]. The JDL syntax consists on statements like:

```
attribute = value;
```

Note: The JDL is sensitive to blank characters and tabs. NO blank characters or tabs should follow the semicolon at the end of a line.

In a job description file, some attributes are mandatory, while some others are optional. Essentially, one must at least specify the name of the executable, the files where to write the standard output and the standard error of the job (they can even be the same file). For example:

```
Executable = "test.sh";
StdOutput = "std.out";
StdError = "std.err";
```

If needed, arguments can be passed to the executable:

```
Arguments = "hello 10";
```

Files to be transferred between the UI and the WN before (Input Sandbox) and after (Output Sandbox) the job execution can be specified:

```
InputSandbox = {"test.sh","std.in"};
OutputSandbox = {"std.out","std.err"};
```

Wildcards are allowed only in the **InputSandbox** attribute. The list of files in the Input Sandbox is specified relatively to the current working directory. Absolute paths cannot be specified in the **OutputSandbox** attribute. Neither the Input Sandbox nor the Output Sandbox lists can contain two files with the same name (even if in different paths) as when transferred they would overwrite each other.

Note: The executable flag is not preserved for the files included in the Input Sandbox when transferred to the WN. Therefore, for any file needing execution permissions a `chmod u+x` operation should be performed by the initial script specified as the **Executable** in the JDL file (the `chmod u+x` operation is done automatically for this script).

The environment of the job can be modified using the **Environment** attribute. For example:

```
Environment = {"CMS_PATH=$HOME/cms",
               "CMS_DB=$CMS_PATH/cmdb"};
```

To express any kind of requirement on the resources where the job can run, there is the **Requirements** attribute. Its value is a Boolean expression that must evaluate to true for a job to run on that specific CE. For that purpose all the GLUE attributes of the IS can be used. For a list of GLUE attributes, see Appendix A.

To run on a CE using PBS as the LRMS, whose WNs have at least two CPUs and the job can run for at least two hours then in the job description file one could put:

```
Requirements = other.GlueCEInfoLRMSType == "PBS" &&
               other.GlueCEInfoTotalCPUs > 1 &&
               other.GLUECEPolicyMaxCPUTime > 7200;
```

The WMS can be also asked to send a job to a particular CE with the following expression:

```
Requirements = other.GlueCEUniqueID ==
               "lxshare0286.cern.ch:2119/jobmanager-pbs-short";
```

If the job must run on a CE where a particular experiment software is installed and this information is published by the CE, something like the following must be written:

```
Requirements = Member("CMSIM-133",
                      other.GlueHostApplicationSoftwareRunTimeEnvironment);
```

Note: The **Member** operator is used to test if its first argument (a scalar value) is a member of its second argument (a list). In this example, the **GlueHostApplicationSoftwareRunTimeEnvironment** attribute is a list.

Note: Requirements on attributes of a CE are written prefixing **other.** to the attribute name in the Information System schema.

It is also possible to use regular expressions when expressing a requirement. Let us suppose for example that the user wants all his jobs to run on CEs in the domain *cern.ch*. This can be achieved putting in the JDL file the following expression:

```
Requirements = RegExp("cern.ch", other.GlueCEUniqueId);
```

The opposite can be required by using:

```
Requirements = (!RegExp("cern.ch", other.GlueCEUniqueId));
```

The choice of the CE where to execute the job, among all the ones satisfying the requirements, is based on the *rank* of the CE; namely, a quantity expressed as a floating-point number. The CE with the highest rank is the one selected.

The user can define the rank with the **Rank** attribute as a function of the CE attributes, like in the following (which is also the default definition):

```
Rank = other.GlueCEStateFreeCPUs;
```

3.3. Exercise JS-2: List the content of the current directory on the worker node; grid-map file

A complication of working on the grid is that one cannot easily type in Unix commands in the shell one is running or testing a program as we are used to do when developing code on a Linux desktop. On the

grid your program may not be running on your machine and/or in a completely different shell. In this example we will learn how to get information about the environment when your program is running on a remote worker node. This exercise lets you do the simplest of all: list the files in the home directory (`$HOME`) on the worker node where the program is running.

We therefore submit here (after **grid-proxy-init**) a job using the executable **/bin/ls**, and we redirect the standard output to a file (JDL attribute: `Stdoutput = "ListOfFiles.txt";`), which is retrieved via the Output Sandbox mechanism to a local directory on the User Interface machine. The result of the file listing command will be the list of files on the `$HOME` directory of the local user account on the Worker Node to which we are mapped. We can issue `edg-job-submit <JobId>` and `edg-job-get-output <JobId>` (after the job is in the **Done (Success)** status) to get the output.

EXCERCISE

1. study the *listOfFiles.jdl* file in this exercise and see if you understand the specifications including the **RetryCount** and **Requirements**.
2. submit the job to the grid and retrieve the output after the job has finished.
3. verify that you understand the output.

THE GRID-MAPFILE MECHANISM

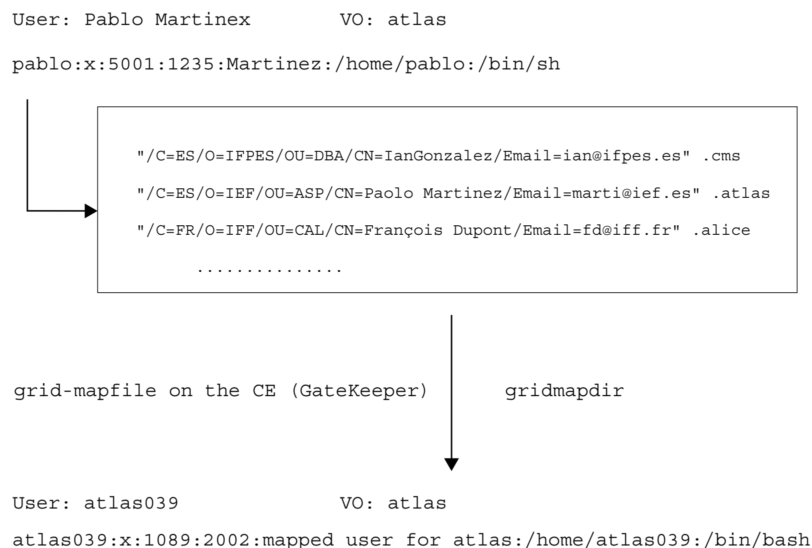

Figure 3.2: The grid-mapfile mechanism

Every user is mapped onto a local user account on the various Computing Elements all over the Grid. This mapping is controlled by the */etc/grid-security/grid-mapfile* file on the Gatekeeper machine and is

known as the *grid-mapfile* mechanism: Every user (identified by their personal certificate's subject) must be listed in the *grid-mapfile* file and associated to one of the pooled accounts available on the CE for the locally supported *Virtual Organisation (VO)*he belongs to (see Figure 3.2). The *grid-mapfile* mechanism, which is part of *Grid Security Infrastructure (GSI)*, requires that each individual user on the Grid is assigned to a unique local User ID. The *accounts leasing* mechanism allows access to take place without the need for the system manager to create an individual user account for each potential user on each computing element. On their first access to a LCG-2 site, users are given a temporary "leased" identity (similar to temporary network addresses given to PCs by the *Dynamic Host Configuration Protocol (DHCP)* mechanism). This identity is valid for the task duration and need not to be freed afterwards. If the lease still exists when the user reenters the site, the same account will be reassigned to him (See http://www.gridpp.ac.uk/gridmapdir/).

3.4. EXERCISE JS-3: PING A HOST FROM A NODE; THE SUBMISSION OF SHELL SCRIPTS TO THE GRID

In this exercise we ping a host from a Worker Node, to exercise again the execution of simple operating system commands on the nodes. In this particular case we will execute the **ping** command in two ways: directly calling the **/bin/ping** executable on the node and by executing a simple shell script (*pinger.sh*) which does the same thing. This will teach us how to use shell scripts on the grid.

EXERCISE

1. Execute the **ping** command on host *www.cern.ch* on the User Interface machine to see what it does. Depending on the installation ping ends by itself or has to be stopped with Ctrl-c. (If you want to find out more about the ping command type `man ping`)
2. Have a look at the *pinger1.jdl* file and try to understand what it does.
3. Submit the "pinger1" job on the grid and retrieve the output when the job has finished and see if the output is what you expected.
4. Now have a look at the *pinger2.jdl* file and notice the differences.
5. Submit the "pinger2" job on the grid and retrieve the output when the job has finished and verify if the output is the same.

As you may have noticed, the executable in the second job was the bash shell itself. The parameters for this executable are then the **ping** command and the **hostname**. In this case one uses a Unix fork and executes the command in the new shell. This may be usefull in case you know your script only works within a specific shell. Without a fork this should also work but then you have to be sure your script can run in the default shell of the worker node. In that case the executable becomes the *pinger.sh* script and the argument to be passed to the executable is just the **hostname**.

In the first case we directly call the ping executable (the JDL file is *pinger1.jdl*):

```
Executable     = "/bin/ping";
Arguments      = "-c 5 lxshare0220.cern.ch";
```

```
RetryCount     = 7;
StdOutput      = "pingmessage1.txt";
StdError       = "stderror";
OutputSandbox = {"pingmessage1.txt","stderror"};
Requirements = other.GlueHostOperatingSystemName == "Redhat";
```

Whereas in the second case we call the **bash** executable to run a shell script, giving as input argument both the name of the shell script and the name of the host to be pinged, as required by the shell script itself (the JDL file is *pinger2.jdl*):

```
Executable     = "/bin/bash";
Arguments      = "pinger.sh lxshare0220.cern.ch";
RetryCount     = 7;
StdOutput      = "pingmessage2.txt";
StdError       = "stderror";
InputSandbox   = "pinger.sh";
OutputSandbox = {"pingmessage2.txt", "stderror"};
Requirements = other.GlueHostOperatingSystemName == "Redhat";
```

where the *pinger.sh* shell script, to be executed in bash, is the following one:

```
#!/bin/bash
/bin/ping -c 5 $1
```

EXERCISE

1. Make your own *pinger3.jdl* file where you make *pinger.sh* the executable.
2. Run this new job on the grid and verify the output
3. Make you own *student.sh* script in which you don't ping a host but executes some other commands like for example **/bin/pwd** or **/usr/bin/who**.
4. Submit this script to the grid make sure that the output you get back is what you expected.

3.5. EXERCISE JS-4: RENDERING OF SATELLITE IMAGES USING DEMTOOLS

In this exercise we demonstrate the use of the grid for and Earth Observation application. We will use the DEMTOOLS program on the grid, which is a satellite images rendering program. Staring from files in the *.dem* format (Digital Elevation Model, usually acquired by high resolution remote sensing satellites) produces graphical virtual reality images in the *.wrl* file format, which can then be browsed and rotated using the **lookat** command, after output retrieval.

We need to specify on input the satellite remote sensing data stored in the 2 files containing satellite views of:

- Mont Saint Helens: *mount_sainte_helens_WA.dem*;
- Grand Canyon: *grand_canyon_AZ.dem*.

After the job s execution we need to specify the name of the 2 produced images we want back to our UI machine.

EXERCISE

1. Have a look at the *demtools.jdl* file and make sure you understand how it works. Note that we need to require that the DEMTOOLS software is installed on the CE for the job to be able to run.
2. Use the **edg-job-list-match** command to find out yourself to which sites this job can be submitted.
3. Submit the job to the grid and check that indeed to RB does the matching properly and sends it to one of the CEs that confirm the requirement specified in the *.jdl* file.
4. Check the execution job and copy the output to your home directory when the job is finished.
5. Use the lookat browser to view the Grand Canyon and mount St.Helena.

The JDL file (*demtools.jdl*) is the following one:

```
Executable     = "/bin/sh";
StdOutput      = "demtools.out";
StdError       = "demtools.err";
InputSandbox   = {"start_demtools.sh",
                   "mount_sainte_helens_WA.dem",
                   "grand_canyon_AZ.dem"};
OutputSandbox = {"demtools.out",
                   "demtools.err",
                   "mount_sainte_helens_WA.ppm",
                   "mount_sainte_helens_WA.wrl",
                   "grand_canyon_AZ.ppm",
                   "grand_canyon_AZ.wrl"};
RetryCount     = 7;
Arguments      = "start_demtools.sh";
Requirements   = Member("DEMTOOLS",
                 other.GlueHostApplicationSoftwareRunTimeEnvironment);
```

Note that we need to expressly require that the destination CE should have the DEMTOOLs software installed using the **Requirements** descriptive in the JDL file.

The launching shell script (*start_demtools.sh*) used is the following one:

```
/usr/local/bin/dem2ppm mount_sainte_helens_WA.dem \
    mount_sainte_helens_WA.ppm
/usr/local/bin/dem2vrml -r 2 mount_sainte_helens_WA.dem \
    mount_sainte_helens_WA.wrl}
/usr/local/bin/dem2ppm grand_canyon_AZ.dem \
    grand_canyon_AZ.ppm}
/usr/local/bin/dem2vrml -r 2 grand_canyon_AZ.dem \
    grand_canyon_AZ.wrl
```

To check the effective presence of available CEs for the job to be correctly executed, as usual, we can issue a `edg-job-list-match demtools.jdl`. After we checked (issuing a `edg-job-status <JobId>`) that the Job reached the **OutputReady** status, we can retrieve the output locally on the UI machine (issuing a `edg-job-get-output <JobId>`). Finally, to visualize the produced graphical output file, we can issue:

```
$ lookat grand_canyon_AZ.wrl
$ lookat mount_sainte_helens_WA.wrl
```

3.6. Exercise JS-5: Using POVRAY to generate vision ray-tracer images

Yet another application for the grid: the processing of X-ray. We will run the POVRAY program on the grid (http://www.povray.org) which starting from a file in the *.pov* format (in this example an Xray image of a pipe duct) creates a Vision Ray-Tracer image in the *.png* format.

Exercise

1. Look at the *povray_pipe.jdl* and make sure you understand what it does. Note that the job execution itself is done in a shell script *start_povray_pipe.sh* and that the POVRAY software is required to be present on the node.
2. Look at the *start_povray_pipe.sh* script and make sure you understand what it does.
3. Look for yourself which CE s are able to accept this job
4. Submit the job to the grid and copy the output to your home directory after the job has finished
5. Look at the *pipeset.png* file with **Mozilla** or **xv** tool. Note that for **xv** you have to make sure that the `$DISPLAY` parameter is set to your desktop machine.

The JDL file (*povray_pipe.jdl*) is the following one:

```
Executable     = "/bin/sh";
StdOutput      = "povray_pipe.out";
StdError       = "povray_pipe.err";
InputSandbox   = {"start_povray_pipe.sh", "pipeset.pov"};
OutputSandbox = {"povray_pipe.out",
                  "povray_pipe.err",
                  "pipeset.png"};
RetryCount     = 7;
Arguments      = "start_povray_pipe.sh";
Requirements   = Member("POVRAY-3.1",
                 other.GlueHostApplicationSoftwareRunTimeEnvironment);
```

The executable to be used in this case is the **sh** shell executable, giving as an input argument to it the name of the shell script we want to be executed (*start_povray_pipe.sh*):

```
#!/bin/bash
mv pipeset.pov OBJECT.POV
/usr/local/bin/x-povray /usr/local/lib/povray31/res640.ini
mv OBJECT.png pipeset.png
```

We can finally, after having retrieved the output of the job, examine the produced image using **Mozilla** or using **qiv** (after having exported the `$DISPLAY` variable to our current terminal).

Since we require a special software executable (*/usr/local/bin/x-povray/usr/local/lib/povray3/res640.ini*), which is identified by the Grid Run Time Environment tag called "POVRAY-3.1", we notice here that we need to specify it in the Requirements classAd, in order to consider (during the matchmaking done by the Broker to select the optional CE to send the job to) only those CEs which have this software installed. This has been done in the last line of the JDL file.

3.7. EXERCISE JS-6: CHECKSUM ON A LARGE INPUT SANDBOX TRANSFERRED FILE

In this exercise you will transfer via the Input Sandbox a file whose bit wise checksum is known to a worker node. On the worker node you will check that the file was transferred correctly by performing a checksum again. You will use the shell script *ChecksumShort.sh*, which exports in an environmental variable (`$CSTRUE`) the value of the Checksum for the file before the transfer. In the script the Checksum is performed again on the worker node by issuing the **cksum** command on the file and the result is stored in the `$CSTEST` variable. When `$CSTEST` is equal to `$CSTRUE` the file was not corrupted during the transfer.

EXERCISE

1. Go to the directory *JSexercise6* and perform the checksum locally on the file *short.dat*, which is present in that directory. Make sure you understand the **cksum** command. More information about this command you get by typing `cksum --help`.
2. Look at the *ChecksumShort.jdl* file and note that no arguments need to be passed with this job and that only the shell script *ChecksumShort.sh* will be executed.
3. Look at the *ChecksumShort.sh* file and note that indeed all parameters needed to run the job are specified in here. Try to understand what this script does and try to predict what you will see in the output file after the job has finished.
4. Submit the job to the grid, check its status and retrieve the output and verify that the answer is what you expected.
5. Change the value for the checksum and then submit the job again and verify you understand the answer that comes back now.

The JDL file (*ChecksumShort.jdl*) is the following one:

```
Executable     = "ChecksumShort.sh";
StdOutput      = "std.out";
StdError       = "std.err";
InputSandbox   = {"ChecksumShort.sh", "short.dat"};
OutputSandbox  = {"std.out", "std.err"};
Arguments      = "none";
```

If everything worked fine, and the GridFTP InputSandbox transfer was OK, the *std.out* should read:

```
True checksum:'2933094182 1048576 short.dat'
Test checksum:'2933094182 1048576 short.dat'
Done checking.
Goodbye. [OK]
```

3.8. Exercise JS-7: A small cascade of "Hello World" jobs

Very often you do not want to submit just one job but a whole series of jobs with the same executable but with a different input files each time (e.g. when you want to run a reconstruction file on all data files from the month January). In this exercise you will submit a small cascade of "Hello World" jobs. This exercise will show how to allow you to look up the status of any of the jobs you submitted or to retrieve the output of any of those jobs without having to remember the JobId's of each individual job that was submitted.

EXERCISE

1. Look at the *submitter.sh* script and note that in this case we don't submit this script to the grid but instead run this script locally and it submits jobs to the grid. Note that the scripts takes 2 parameters, the first one is the number of times you want to submit the "HelloWorld" job and the second one is the name of the file where you want to store the JobId's in.
2. Note that in the script the **-o** option of the **edg-job-submit** command is used. Find out what this parameter does.
3. Have a look at the *HelloWorld.jdl* and note that it is the same simple job as in JS exercise 1.
4. Run the script by typing **`./submitter.sh 4 HelloWorld.jid`**
5. Look in the *HelloWorld.jid* file and make sure you understand what it contains.
6. Make sure you understand how this works. Note, if you run the job again you may want to use a different name for the file with the JobId's (e.g. *HelloWorld.jid2* etc).

The shell script (*submitter.sh*) that loops a given amount of times submitting a single job each occasion is the following one:

```
#!/bin/bash
i=0
while [ $i -lt $1 ]
  do edg-job-submit -o $2 HelloWorld.jdl
  i=`expr $i +1`
done
```

From the UI we can now issue the command:

```
$ ./submitter.sh 4 HelloWorld.jobids
```

The *HelloWorld.jid* file is a file containing all JobIds for the 4 submitted Jobs. To get info on the Jobs status we can issue `edg-job-status -i HelloWorld.jid`. To collectively retrieve the output we can issue a `edg-job-get-output -i HelloWorld.jid` and then examine the content of the files on the local temporary directories on the UI, where files are retrieved.

3.9. EXERCISE JS-8: MPI JOBS

Running parallel programs has become very important in recent years because of the large increase in the number of available processors. Using message passing between multiple processes with the MPI library is a common way to construct parallel programs. Usually multiple copies of a program are started that know about their rank within this group. This rank can then be used to decide about the distribution of the work, and special routines are used to communicate between the processes. When using the LCG-2 middleware it is also possible to run jobs that make use of this *MPI* message passing library.

In order to run a parallel MPI job the **JobType** attribute has to be specified, and set to *MPICH* (mpich is a public domain version of the mpi library). When specifying `JobType="MPICH"` the number of nodes to use must also be specified using the attribute **NodeNumber**.

An example is:

```
JobType = "MPICH";
NodeNumber = 4;
```

When specifying `JobType="MPICH"` the job will be sent to a CE that supports MPICH. This requirement is automatically added to the job. Note, however, that the support for MPI jobs is currently very limited within the LCG-2 grid infrastructure.

The executable given with the Executable attribute will be run in parallel using multiple processors on one or more worker nodes.

EXERCISE

1. Look at the *linpack4.jdl* file in the JSExercise8 directory. Try to understand the contents of the file.
2. Submit the job to the grid.
3. Look at the output and determine the Gigaflop rate for the run.
4. Look at the *linpack1.jdl* file, and try to understand the contents of this file.
5. Submit the job to the same CE you used for the previous job.
6. Look at the output and determine the Gigaflop rate for the run. Do you understand the difference?

In the directory JSExercise8 an example MPI job is available. The job will run the linpack parallel benchmark over 4 processors. The benchmark measures the time needed to solve a large linear system. The *linpack4.jdl* file looks like follows:

```
JobType = "MPICH";
NodeNumber = 4;
Executable = "xhpl";
StdOutput = "linpack.out";
StdError = "std.err";
OutputSandbox = {"linpack.out","std.err"};
InputSandbox = {"xhpl","hpl4/HPL.dat"};
Environment = "P4_GLOBMEMSIZE=128000000";
```

Note the **JobType** and **NodeNumber** attributes. The Enviroment variable is needed to increase the amount of memory available for shared memory communication within a multi-processor node.

The executable **xhpl** and input file *hpl4/HPL.dat* will be transferred tot the worker nodes and run. Because the processor layout is specified in the *HPL.dat* file we need separate input files for single and four processor runs.

Within the outputfile the GFlop rate for the four processor run will be shown like:

```
============================================================================
T/V                N    NB     P     Q               Time             Gflops
----------------------------------------------------------------------------
WR12L8R4       10000   240     2     2              64.59          1.032e+01
----------------------------------------------------------------------------
||Ax-b||_oo / ( eps * ||A||_1  * N        ) =        0.1014441 ...... PASSED
||Ax-b||_oo / ( eps * ||A||_1  * ||x||_1  ) =        0.0240047 ...... PASSED
||Ax-b||_oo / ( eps * ||A||_oo * ||x||_oo ) =        0.0053655 ...... PASSED
============================================================================
```

3.9.1. The Graphical User Interface

The EDG WMS GUI is a Graphical User Interface composed of three different applications:

- a Job Description Language Editor (**edg-wl-ui-jdleditor.sh**);
- a Job Monitor (**edg-wl-ui-jobmonitor.sh**);
- a Job Submitter (**edg-wl-ui-jobsubmitter.sh**).

The three GUI components are integrated although they can be used as standalone applications so that the JDL Editor and the Job Monitor can be invoked from the Job Submitter, thus providing a comprehensive tool covering all main aspects of job management in a Grid environment: from creation of job descriptions to job submission, monitoring and control up to output retrieval.

Note: To be able to use these applications set the `$DISPLAY` environment variable appropriately.

Details on the EDG WMS GUI are not given in this guide. Please refer to [R18] for a complete description of the functionalities provided by the GUI, together with some example screen shots.

CHAPTER 4

DATA MANAGEMENT

4.1. INTRODUCTION

The Data Management services are provided by the *Replica Management System (RMS)* of the *European DataGrid (EDG)* [R9]. In a Grid environment, the data files are replicated, possibly on a temporary basis, to many different sites depending on where the data is needed. The users or applications do not need to know where the data is located. They use logical names for the files and the Data Management services are responsible for locating and accessing the data. The Input/Output Sandbox is a mechanism for transferring small data files needed to start the job or to check the final status over the Grid. Large data files are available on the Grid and known to other users only if they are stored on *Storage Elements (SEs)* and registered in the Replica Management System catalogues. In order to optimise data access and to introduce fault-tolerance and redundancy, data files can be replicated on the Grid. The EDG *Replica Manager (RM)*, the *Replica Location Service (RLS)* and the *Replica Metadata Catalogue (RMC)*are the tools available for performing these tasks. Only anonymous access to the data catalogues is supported: the user proxy is not used to control the access to them.

The files in the Grid are referenced by different names:

- *Grid Unique IDentifier (GUID)*; a file can always be identified by its GUID, which is assigned at data registration time and is based on the *Universal Unique IDentifier (UUID)* standard to guarantee unique IDs. A GUID is of the form:
  guid:<unique_string>
  and all the replicas of a file will share the same GUID. An example GUID is:
  `guid:3cb13190-ab23-11d8-bc9c-d39c21caf9ab`
- *Logical File Name (LFN)*; in order to locate a Grid accessible file, the human user will normally use a LFN. LFNs are usually more intuitive, human-readable strings, since they are allocated by the user as GUID aliases. Their form is:
  lfn:<any_alias>
  An example LFN is:
  `lfn:importantResults/Test1240.dat`
- *Storage URL (SURL)*; the SURL is used by the RMS to find where a replica is physically stored, and by the SE to locate it. The SURL is of the form:
  sfn:// <SE_hostname><SE_Accesspoint><VO_path><filename>
  An example SURL is:

`sfn://tbed0101.cern.ch/flatfiles/SE00/dteam/generated/2004-02-26/`
`file3596e86f-c402-11d7-a6b0-f53ee5a37e1d`

- *Transport URL (TURL)*; the TURL gives the necessary information to retrieve a physical replica, including hostname, path, protocol and port (as any conventional URL); so that the application can open and retrieve it. The TURL is of the form:
  <protocol>:// <SE_hostname><SE_Accesspoint><VO_path><filename>
  An example TURL is:
  `gsiftp://tbed0101.cern.ch/flatfiles/SE00/dteam/generated/2004-02-26/`
  `file3596e86f-c402-11d7-a6b0-f53ee5a37e1d`

While the GUID or LFN refer to files and not replicas, and say nothing about locations, the SURLs and TURLs give information about where a physical replica is located. Figure 4.1 shows the relation between the different file names.

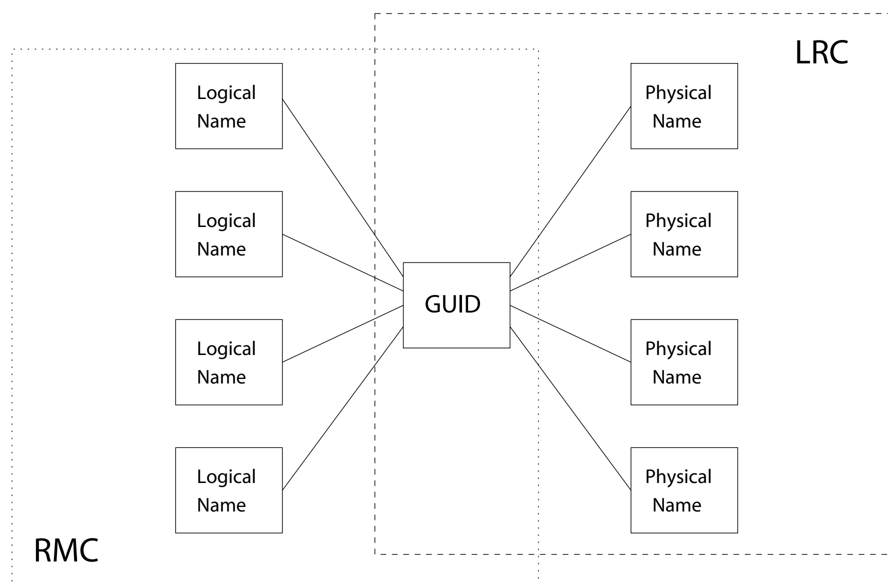

Figure 4.1: The logical File Name to GUID mapping is maintained in the Replica Metadata Catalogue, the GUID to Physical File Name mapping in the Replica Location Service.

The main services offered by the Replica Management System are the Replica Location Service and the Replica Metadata Catalogue. The RLS maintains information about the physical location of the replicas (mapping with the GUIDs). It is composed of *Local Replica Catalogues (LRCs)* which hold the information of replicas for a single *Virual Organisation (VO)*. The RMC stores the mapping between GUIDs and the respective aliases (LFNs) associated with them, and maintains other metadata information (sizes, dates, ownerships ...). For the moment these catalogues are centralized and there is one RLS (with its LRC and RMC) per VO.

The last component of the Data Management framework is the *Replica Manager (RM)*. The Replica Manager presents a single interface for the RMS to the user, and interacts with the other services. In the LCG-2, this interface is integrated with the User Interface.

4.1.1. EDG Data Management Tools

In this chapter, exercises will be presented to make you familiar with the EDG Data Management tools. These are high level tools used to upload files to the grid, replicate data and locate the best replica available. The Data Management tools are:

- **edg-replica-manager** (**edg-rm**)
- **edg-local-replica-catalog** (**edg-lrc**)
- **edg-replica-metadata-catalog** (**edg-rmc**)

The examples presented in the next sections will mainly concern **edg-rm**. For in depth details on how to use all the client tools mentioned above, please refer to [R19], [R20], [R21].

Apart from those presented above, there are two more commands in the UI: the **edg-replica-location-index** (**edg-rli**) and the **edg-replica-optmization** (**edg-ros**). These two commands will allow the user to interact with the Replica Location Index and the Replica Optimisation Services, when they are in work. These services are planned for the LCG architecture, but they are not in use yet (and their commands are therefore not useful at the moment). Information about these two commands can be found in [R23] and [R24].

4.2. Exercise DM-1: Discover Grid storage

In general, all Storage Elements registered with the Grid publish, through the Grid Information System, the location of the directory where to store files. This directory on the SE is usually VO dependent (each defined VO has its own one) but the location can also be hidden by the underlying Storage Element or Storage Resource Manager. For LCG, a Storage Element is mainly implemented by a Storage Resource Manager and thus both terms are used in the remainder of this document.

The are several ways to retrieve information about Storage Elements and their attributes. One way is to directly query the Information System and you will learn how to do so in the exercises about Information Systems (see Chapter 5). Here we will use the Data Management tool **edg-rm** which has the following syntax:

```
edg-replica-manager [options] command [command-options]
```

where the **--vo** <**vo_name**> option specifies the virtual organization of the user (this option is mandatory —without it the command will not work), and <**cmd_name**> is the particular command that the RM must perform. Most commands have both an extended and an abbreviated name form. Other general **edg-rm** options are: **--log-debug**, **--log-info** and **--log-off**, which are used for enabling or disabling bug-level or info-level logging; and the **--config** <**file**> option, which is used to read the specified configuration file, instead of the default *$EDG_LOCATION/etc/edg-replica-manager/edg-replica-manager.conf*.

Note: In the current release, if a local file called *edg-replica-manager.conf* exists, the RM will use it as configuration file even if it is not specified by the user with the --config option.

In what follows some usage examples are given. For details on the options of each command, please use the **--help** option with **edg-rm**. If the name of a command is also given, then specific information about that command is presented. The user can also consult the manpages and [R19].

To retrieve information about SEs we can issue:

```
$ edg-rm --vo alice printInfo
```

EXERCISE

1. Issue the command to retrieve information about Storage Elements.
2. Read and try to understand the output on the screen.
3. Find out which Storage Elements are active right now.
4. Find out how many Storage Elements support the tutorial VO “tutor” (i.e. how many SEs you can use).

A possible (reduced) output looks as follows:

```
$ edg-rm --vo=tutor pi
VO used              : tutor
default SE           : tbn17.nikhef.nl
default CE           : tbn18.nikhef.nl
Info Service         : MDS

RMC endpoint         : http://rlsalice.cern.ch:7777/alice/v2.2/edg-replica-met
LRC endpoint         : http://rlsalice.cern.ch:7777/alice/v2.2/edg-local-repli
ROS endpoint         : no information found : No Service found edg-replica-opt

List of CE ID's       : lcgce02.gridpp.rl.ac.uk:2119/jobmanager-lcgpbs-long
                       tbn18.nikhef.nl:2119/jobmanager-pbs-qshort
[...]
                       gw39.hep.ph.ic.ac.uk:2119/jobmanager-lcgpbs-short
                       gw39.hep.ph.ic.ac.uk:2119/jobmanager-lcgpbs-infinite

CE at long :
                      name : long
                        ID : lcgce02.gridpp.rl.ac.uk:2119/jobmanager-lcgpbs-
                  closeSEs : lcgads01.gridpp.rl.ac.uk,lcgse01.gridpp.rl.ac.u
                       VOs : alice,atlas,cms,lhcb,dteam,babar,dzero
CE at qshort :
                      name : qshort
                        ID : tbn18.nikhef.nl:2119/jobmanager-pbs-qshort
[...]
                  closeSEs : gw38.hep.ph.ic.ac.uk
                       VOs : alice,atlas,cms,lhcb,dteam

List of SE ID's       : grid002.mi.infn.it
                       gridkap02.fzk.de
                       tbn17.nikhef.nl
[...]
                       castorgrid.ific.uv.es
                       gw38.hep.ph.ic.ac.uk
SE at INFN-MILANO-LCG2 :
                      name : INFN-MILANO-LCG2
```

```
                      host : grid002.mi.infn.it
                      type : disk
               accesspoint : /flatfiles/SE00
                       VOs : alice,atlas,cms,lhcb,dteam,infngrid
            VO directories : alice:alice,atlas:atlas,cms:cms,lhcb:lhcb,dteam
                 protocols : gsiftp,rfio
SE at FZK-LCG2 :
                      name : FZK-LCG2
                      host : gridkap02.fzk.de
[...]
                       VOs : atlas,ific,dteam
            VO directories : atlas:atlas,ific:ific,dteam:dteam
                 protocols : gsiftp,rfio
SE at IC-LCG2 :
                      name : IC-LCG2
                      host : gw38.hep.ph.ic.ac.uk
                      type : disk
               accesspoint : /stage/lcg2-data
                       VOs : alice,atlas,cms,lhcb,dteam
            VO directories : alice:alice,atlas:atlas,cms:cms,lhcb:lhcb,dteam
                 protocols : gsiftp,rfio
```

The output presents information about all possible Computing Elements (CEs) as well as Storage Elements that are registered with the information service. We are mainly interested in the Storage Elements and their storage locations. The number of SEs given shows you how many SEs you can possibly use. Note that you also need to find an SE that allows for your VO (tutor). The SE can be implemented in several ways which has an impact on the directory where files are stored:

1. The SE is a conventional disk server with a GridFTP interface. That is a simple solution that does not require additional SE software but has several disadvantages are regards space management on the disk.
2. On the Storage Element, a particular SRM (or EDG-SE) is running that takes care of space management, interfacing to Mass Storage Systems etc.

In case of a conventional disk server that runs GridFTP (`type : disk`, this path might be exposed. In case the SE is running an SRM or EDG-SE (`type : edg-se`), the path might not be there since the SRM decides internally where files can be written to. In all cases, the VO specific directories can be obtained by combining the following two attributes **accesspoint** and **VO directories**. For example:

```
          host : lxshare0408.cern.ch
   accesspoint : /flatfiles/SE00
VO directories : lhcb:/lhcb,cms:/cms,tutor:/tutor
```

Thus, files for the VO tutor are written into the directory */flatfiles/SE00/tutor*.

4.3. Exercise DM-2: File replication with the EDG Replica Manger

In this exercise we will use the EDG Replica Manager for replicating files between various SEs and get familiar with the basic catalogue commands to list and delete replicas. The following new commands will be used:

- **copyAndRegisterFile**
- **listReplicas**, **listGUID**
- **replicateFile**
- **deleteFile**

EXERCISE

1. Create a file using e.g. the **touch** or **echo** commands.
2. Find a SE which you want to use to copy your file to.
3. Copy the created file to the SE and register it in the replica catalogue with a Logical File Name.
4. Check if the copy was successful and that the file is registered.
5. Create a replica of the file at a different SE.
6. Repeat this until you get bored with it...
7. Inspect all created replicas.
8. Copy the file stored on Grid Storage back to you local account.
9. Cleanup and unregister all created replicas.

COPYANDREGISTERFILE; UPLOADING A FILE FROM THE UI TO THE GRID

In order to upload a file to the Grid, i.e., to transfer it from the local machine to a Storage Element where it must reside permanently, the **copyAndRegisterFile** (**cr**) command can be used (in a machine with a valid proxy):

```
$ edg-rm --vo alice copyAndRegisterFile file://$(pwd)/knuth.tex \
> -l lfn:donald
guid:683f176c-ab2e-11d8-bb19-b6edebfcbb1f
```

where the only argument is the local file to be uploaded (a fully qualified URI) and the **-l** option indicates an LFN for it. The command returns the unique GUID for the file. If no LFN is provided, then the returned GUID will be the only way to access the file in the Grid.

If the **-d** <**destination**> option is included, then the specified SE (which must be known in advance) is used as the destination for the file. Without the **-d** option, a default SE is chosen automatically. A complete SURL, including the SE hostname, the path (accesspoint plus VO-specific directory) and a chosen filename, or only the SE hostname can be used as the destination. This is illustrated by the following commands:

```
$ edg-rm --vo alice copyAndRegisterFile file://$(pwd)/knuth.tex \
> -l lfn:donald -d teras.sara.nl
```

or

```
$ edg-rm --vo alice copyAndRegisterFile file://$(pwd)/knuth.tex \
> -l lfn:donald -d sfn://teras.sara.nl/home/alice/donald
```

In this and other commands the **-p** <**protocol**> and **-n** <**#streams**> options can be used to specify the protocol (**gsiftp** being the default one) and the number of parallel streams to be used in the transfer (default is 8).

LISTREPLICAS & LISTGUID; LISTING REPLICAS AND GUIDS

The Replica Manager allows users to list all the replicas of a file that have been successfully registered with the Replica Location Service. For that purpose the **listReplicas** (**lr**) command is used:

```
$ edg-rm --vo alice listReplicas lfn:donald
sfn://teras.sara.nl/home/alice/generated/2004-05-21/file62a6237b-ab2e-11d8-b
b19-b6edebfcbb1f
sfn://lcgse01.gridpp.rl.ac.uk/flatfiles/lcg-data/alice/generated/2004-05-21/
file6a6a7ba7-ab31-11d8-8293-a6af3465f6ce
```

Note: Instead of LFN the GUID or SURL can be used to specify the file for which all replicas must be listed. The SURLs of the replicas are returned.

Reciprocally, the **listGUID** (**lg**) return the GUID associated with a specified LFN or SURL:

```
$ edg-rm --vo alice listGUID sfn://teras.sara.nl/home/alice/generated/2004-
05-21/file62a6237b-ab2e-11d8-bb19-b6edebfcbb1f
guid:683f176c-ab2e-11d8-bb19-b6edebfcbb1f
```

REPLICATEFILE; REPLICATING A FILE

Once a file is stored on an SE and registered with the Replica Location Service, the file can be replicated using the **replicateFile** (**rep**) command, as in:

```
$ edg-rm --vo alice replicateFile \
> guid:683f176c-ab2e-11d8-bb19-b6edebfcbb1f -d lcgse01.gridpp.rl.ac.uk
sfn://lcgse01.gridpp.rl.ac.uk/flatfiles/lcg-data/alice/generated/2004-05-21/
file6a6a7ba7-ab31-11d8-8293-a6af3465f6ce
```

where the file to be replicated can be specified using a LFN, GUID or even a particular SURL, and the **-d** option is used to specify the SE where the new replica will be stored (and, as with **CopyAndRegisterFile**, using either the SE hostname or a complete SURL). If this option is not set, then the an SE is chosen automatically.

Note: For one GUID, there can be only one replica per SE. If the user tries to use the **replicateFile** command with a destination SE that already holds a replica, the existing SURL will be returned, and no new replica will be created.

COPYFILE; COPYING FILES OUT OF THE GRID

The **copyFile** (**cp**) command can be used to copy a Grid file to a non-grid storage resource. This is useful to have a local copy of the file. The command accepts the LFN, GUI or SURL of the LCG-2 file as its first argument and a local filename or valid TURL as the second, as is shown in the following example:

```
$ edg-rm --vo alice copyFile lfn:donald file://$(pwd)/file.new
```

Note: Although this command is designed to copy files from a SE to a non-grid resources, if the proper TURL is used, a file could be transferred from one SE to another, or from out of the Grid to a SE. *This should not be done*, since it has the same effect as using **replicateFile** but *skipping the file registration*, making in this way this replica invisible to Grid users.

DELETEFILE; DELETING REPLICAS

Once a file is stored on a Storage Element and registered with a catalogue, it can be deleted using the **deleteFile** (**del**) command. If a SURL is provided as argument, then that particular replica will be deleted. If a LFN is given instead, then the **-s** <**SE**> option must be used to indicate which one of the replicas must be erased. The same is true if a GUID is specified, unless the **--all-available** option is used, in which case all replicas of the file will be deleted and unregistered (on a best-effort basis).

The following commands:

```
$ edg-rm --vo alice deleteFile guid:683f176c-ab2e-11d8-bb19-b6edebfcbb1f \
> -s lcgse01.gridpp.rl.ac.uk
```

and

```
$ edg-rm --vo alice deleteFile guid:683f176c-ab2e-11d8-bb19-b6edebfcbb1f \
> --all-available
```

remove, from the file system and the catalog, one particular replica and all available replicas of the file, respectively.

GETTURL; OBTAINING A TURL FOR A REPLICA

For any given replica (identified by its SURL) the TURL for accessing it using a particular protocol can be obtained with the **getTurl** (**gt**) command. The arguments are the SURL and the protocol to be used. The command returns the valid TURL or an error message if the specified protocol is not supported by that SE for the given replica. For example:

```
$ edg-rm --vo dteam getTurl \
> \sfn://tbed0101.cern.ch/flatfile/SE00/dteam/generated/2004-02-26/f1 gsiftp
gsiftp://tbed0101.cern.ch/flatfile/SE00/dteam/generated/2004-02-26/f1
```

```
$ edg-rm --vo dteam getTurl \
> sfn://tbed0101.cern.ch/flatfile/SE00/dteam/generated/2004-02-26/f1 ftp
The file sfn://tbed0101.cern.ch/flatfile/SE00/dteam/generated/2004-02-26/f1
is not accessible via the protocol: ftp
```

4.4. EXERCISE DM-3: USING THE REPLICA CATALOG

After some preliminary replica catalogue interaction in the previous exercise, we now go a bit more into detail with using the replica catalogue with the EDG Replica Manager and use the following commands:

- **addAlias**, **removeAlias**
- **registerFile**, **unregisterFile**

EXERCISE

1. Create a file using e.g. the **touch** or **echo** commands.
2. Find a SE which you want to use to copy your file to.
3. Misuse the **copyFile** command to copy the created file to the SE.
4. Become aware of your "mistake" and register the file.
5. Add an alias to the registered file.
6. Cleanup and unregister all created replicas and aliases.

REGISTERFILE/GUID & UNREGISTERFILE/GUID; REGISTERING AND UNREGISTERING GRID FILES

Usually, new files are introduced in LCG-2 copying them from a non-grid resource using **CopyAndRegisterFile**; they are replicated to different SEs using **replicateFile**; and can be copied out of the Grid with **copyFile**. But it is also possible that a file is copied between SEs using **copyFile** (i.e., without registering) or by physically carrying a great amount of data in tapes, or it is possible that a new storage resource that already holds files is added to the Grid (becoming a SE). These files will be in a SE (they will have a valid SURL), but will not be registered in the LCG2 catalogs (i.e., they will not have an associated GUID).

For this situation, the **registerFile** (**rf**) and **registerGUID** (**rg**) commands may be useful. The **registerFile** command creates a new GUID for a given SURL, whereas **registerGUID** associates the replica identified by a SURL with an existent GUID (also specified as an argument). In the second case, it is assumed that there exist already some other replicas of the files that are being registered.

An example of the commands usage follows:

```
$ edg-rm --vo alice registerFile sfn://teras.sara.nl/home/alice/knuth.tex
guid:aa15e024-ab3b-11d8-a233-cd68eaee9526
$
$ edg-rm --vo alice registerGUID \
> sfn://lcgse01.gridpp.rl.ac.uk/flatfiles/lcg-data/alice/knuth.tex \
> guid:d3e9071e-687b-11d8-b3fa-8c0b6b5cbb30
guid:d3e9071e-687b-11d8-b3fa-8c0b6b5cbb30
```

Likewise, instead of using the **deleteFile**, which both unregister and physically deletes a replica, a user can unregister a replica from the LRC catalogue, without actually deleting it (it can still be accessed on the SE with **copyFile**, for instance). This can be achieved with the **unregisterFile** (**uf**) command, specifying both the GUID and the SURL to be unregistered, as in:

```
$ edg-rm --vo alice unregisterFile \
> guid:aa15e024-ab3b-11d8-a233-cd68eaee9526 \
> sfn://teras.sara.nl/home/alice/knuth.tex
$
$  edg-rm --vo alice unregisterFile \
> guid:d3e9071e-687b-11d8-b3fa-8c0b6b5cbb30 \
> sfn://lcgse01.gridpp.rl.ac.uk/flatfiles/lcg-data/alice/knuth.tex
```

Note: If the last replica of a file is unregistered, then the GUI is also removed from the catalogue.

ADDALIAS & REMOVEALIAS; MANAGING ALIASES

The **addAlias** (**aa**) command allows the user to add a new LFN to an existing GUID:

```
$ edg-rm --vo dteam addAlias guid:c06a92ee-6911-11d8-a453-d9c1af867039 \
> lfn:last\_results
```

The **removeAlias** (**ra**) command allows the user to remove an LFN from an existing GUID:

```
$ edg-rm -\,-vo=dteam removeAlias guid:c06a92ee-6911-11d8-a453-d9c1af867039 \
> lfn:last\_results
```

In order to list the aliases of a file, the user has to use the **edg-replica-metadata-catalog** command, discussed later.

LIST; LISTING AN SE DIRECTORY

The **list** (**ls**) command can be used to list the contents of an SE directory (and, in the future, of an SRM directory):

```
$ edg-rm --vo alice list sfn://teras.sara.nl/home/alice
edg_query_vo_storage.dat
edg_query_vo_storage.dat.old
generated
```

The argument of the command is a URI where the schema can be **sfn**, **srm**, or **gsiftp**.

4.5. EXERCISE DM-4: ACCESSING A GRID FILE FROM A JOB

A job that is submitted to the Grid can access files stored in LCG-2. For that purpose, the JDL file of the job must include the name (GUID or LFN) of the files to be accessed, in the **InputData** attribute; and the protocol that will be used to access them in the **DataAccessProtocol** attribute. Currently, the only two supported protocols to access grid files are: GridFTP (**gsiftp**) and rfio (**rfio**).

The following exercise show access of the files from a Perl script. It could be done also from a C++ or Java program, using the respective **edg-rm** and **rfio** APIs. For information on that, please refer to [R25] and [R26].

Note: The Logical File Names used in the exercises should be treated as exemplary. Since, the LFNs that can be registered in the replica catalogue are unique, to be able to do these exercises the lfn should be altered accordingly.

EXERCISE

1. Change to the *DMExercise4* directory, and study the *.jdl* and *.pl* files.
2. Copy and register the file *values* to a SE (e.g. *lcgse01.gridpp.rl.ac.uk*.) with your uniquely chosen LFN. Note that the SE should support both **gsiftp** and **rfio**.
3. Change the LFNs in the scripts and JDL files to make then reflect your LFN.
4. Run the *gsiftp.jdl* job (Tip: first try to run the Perl scrips locally, this can save you a lot of debug time).
5. Retrieve, inspect and try to understand the output of the job.
6. Run the *rfio.jdl* job.
7. Retrieve, inspect and try to understand the output of the job.
8. Do you understand the difference between the two Grid file accessing methods...
9. (Extra: Rewrite the Perl files in your favourite "scripting" language, e.g. bash, python, . . .)

ACCESSING A FILE USING THE GRIDFTP PROTOCOL

We assume that a user has registered a data file (called *values*) within LCG-2, using **lfn:unique_name** as its LFN. The contents of the file are the following:

```
The contents of these lines,
which are not really important,
will be shown in the std.out file.
```

The JDL file of the job (*gsiftp.jdl*) includes the LFN of the file, and the protocol (**gsiftp**) to be used when accessing it. The contains of the JDL file follows:

```
Executable="gsiftp.pl";
StdOutput="std.out";
StdError="std.err";
InputSandbox={"gsiftp.pl"};
OutputSandbox={"std.out","std.err"};
InputData={"lfn:unique_name"};
DataAccessProtocol={"gsiftp"};
```

The executable (**gsiftp.pl**) is a Perl program, that calls the **edg-rm copyFile** command to copy the grid file to the local filesystem of the Worker Node where the job is running. The rest of the script is simple Perl code to show the data retrieved:

```
#!/usr/bin/perl

# Copy the input data file to the WN local filesystem
system "edg-rm --vo alice copyFile lfn:unique_name file://$(pwd)/values";

# Open it
open(file,'values');
```

```
# Read all the lines\\
@lines=<file>;

#Show the info
print "The values stored in the input data file are:\n";
print " @lines";
```

The job is submitted as usual:

```
$ edg-job-submit -o jobid gsiftp.jdl
```

And the results retrieved with:

```
$ edg-job-get-output -i jobid
```

The *std.out* file obtained is this:

```
The values stored in the input data file are:
  The contents of these lines,
  which are not really important,
  will be shown in the std.out file.
```

ACCESSING A FILE USING THE RFIO PROTOCOL

This exercise is very similar to the previous one, but here the rfio protocol is used. The same data file *values* is used, and the only changes in the new *rfio.jdl* file are the executable file, and the access protocol:

```
Executable="rfio.pl";
StdOutput="std.out";
StdError="std.err";
InputSandbox={"rfio.pl"};
OutputSandbox={"std.out","std.err"};
InputData={"lfn:unique_name"};
DataAccessProtocol={"rfio"};
```

The *rfio.pl* file is a bit more complicated this time, because the rfio protocol cannot handle LFNs, and needs the complete path to the file instead. For this reason, the TURL of the file is obtained first and then it is adapted to rfio needs. The commands to get the TURL from a known LFN have been already seen and could be also performed manually instead of inserting them in the Perl script, but are included here for completeness. The form of the TURL will be: **rfio://<hostname>/<path>**, while the rfio command expects a **<hostname>:<path>** string, and therefore the Perl code has to do a little extra work to adapt the string before invoking the **rfcp** command, which copies the file to the WN local filesystem.

```
#!/usr/bin/perl

# Obtain the SURL of the file whose LFN we know
$surl = `edg-rm --vo alice lr lfn:unique_name`;
chop($surl);

# Now obtain the TURL for the rfio protocol
$turl = `edg-rm --vo alice getTurl $surl rfio`;

# Adapt the "rfio://hostname/path" form returned to the
# "hostname:path" form that rfio uses
```

```
$turl =~ s/rfio:\/\///;  # delete the extra "//"
$turl =~ s/\//:\//;      # add the ":"
chop($turl);

# Copy the input data file to the WN local filesystem
system "rfcp $turl `pwd`/values";

# Open it
open(file,'values');

# Read all the lines\\
@lines=<file>;

#Show the info
print "The values stored in the input data file are:\n";
print " @lines";
```

Note: That the Perl script, *rfio.pl*, can not deal with more than one replica per LFN. This is due to the fact of using **lr** in "Obtaining the SURL of the file whose LFN we know". For a more robust way of doing this see 4.6. (using **getBestFile**).

The job is submitted and the output retrieved like in the previous example, and the retrieved *std.out* file is:

```
95 bytes in 0 seconds through eth0 (in) and local (out)
  The values stored in the input data file are:
  The contents of these lines,
  which are not really important,
  will be shown in the std.out file.
```

where the first line is produced by the **rfcp** command.

4.6. Exercise DM-5: Replica Optimisation with the EDG Replica Manager

In this exercise we will learn more about the EDG Replica Manger basic file replication an optimisation services. We will use the following new commands:

- **listBestFile**
- **getBestFile**
- **getAccessCost**

In the scenario the user knows that there is a file available at CERN, that has been put on a host accessible through GridFTP. It is not a grid-aware store, so first the user has to copy the file to a Storage Element and register it in the Grid. Say that for some reason the user cannot copy it to the local CERN Storage Element but has to copy it to the one at IN2P3 in France.

In the example the file is called *higgs0* and resides at *testbed008.cern.ch/tmp/*.

Note: That the machine names and the directories change over time: please consult your tutorial web page for the exact machine names or enquire the machine information as shown in exercise DM-1! All machine names given here are only examples but they are not necessarily the one you can use.

Copy and registration is an atomic operation. In the example we assign also a Logical File Name alias to it in the process, *lfn:higgs*, which is easier to remember than the GUID that is returned by the call:

```
$ edg-rm --vo=tutor copyAndRegisterFile \
> gsiftp://testbed008.cern.ch/tmp/higgs0 -l lfn:higgs
> -d srm://ccgridli02.in2p3.fr/edg/StorageElement/dev2/tutor/higgs
guid:7c29f32b-4964-11d7-a86c-9ee9a33b1f19
```

To verify whether the operation got successfully executed, we can issue **listReplicas**:

```
$ edg-rm --vo=tutor listReplicas lfn:higgs
srm://ccgridli02.in2p3.fr/edg/StorageElement/dev2/tutor/higgs
```

As a second step, the user might want to have a replica of this data file available at NIKHEF in the Netherlands, because he intends to share it or to submit jobs that require resources at NIKHEF. A replica can be created using the **replicateFile** command:

```
$ edg-rm --vo=tutor replicateFile lfn:higgs \
> -d srm://se01.nikhef.nl/flatfiles/tutor/higgs
srm://se01.nikhef.nl/flatfiles/tutor/higgs
```

The command confirms its execution by returning the actual SURL used. Note that if the **-d** option is omitted, an automatic SURL would have been created. To list all replicas now in the system, we can issue **listReplicas**:

```
$ edg-rm --vo=tutor listReplicas lfn:higgs
srm://ccgridli02.in2p3.fr/edg/StorageElement/dev2/tutor/higgs
srm://se01.nikhef.nl/flatfiles/tutor/higgs
```

To actually make the best file available at CERN, we can issue **getBestFile**:

```
$ edg-rm --vo=tutor getBestFile lfn:higgs -d pcrd24.cern.ch
srm://pcrd24.cern.ch/data/temp/a6289c7c-4966-11d7-bc63-d91230733e2d
```

We should now have three replicas:

```
$ edg-rm --vo=tutor listReplicas lfn:higgs
srm://pcrd24.cern.ch/data/temp/aaa64014-4967-11d7-a6cc-f7a1ff1899b0
srm://se01.nikhef.nl/flatfiles/tutor/higgs
srm://ccgridli02.in2p3.fr/edg/StorageElement/dev2/tutor/higgs
```

To delete a replica we can use the **deleteFile** command (e.g.):

```
$ edg-rm deleteFile lfn:higgs -s ccgridli02.in2p3.fr
```

EXERCISE

Create a few more replicas, see how the commands **listBestFile** and **getBestFile** work on your replicas.

4.7. Exercise DM-6: Taking a look at the .BrokerInfo file

When a Job is submitted to the Grid, we do not know *a-priori* its destination Computing Element.

There are reciprocal "closeness" relationships among Computing Elements and Storage Elements (SE) which are taken into account during the match making phase by the Resource Broker and affect the choice of the destination CE according to where required input data are stored.

In general, we need a way to inform the Job of the choice made by the Resource Broker during the matchmaking so that the Job itself knows which are the physical files actually to be opened. Therefore, according to the actual location of the chosen CE, there must be a way to inform the job how to access the data.

This is achieved using a file called the *.BrokerInfo* file, which is written at the end of the matchmaking process by the Resource Broker and it is sent to the worker node as part of the *InputSandbox*.

The *.BrokerInfo* file contains all information relevant for the Job, like the destination CEId, the required data access protocol for each of the SEs needed to access the input data ("file" - if the file can be opened locally, **rfio** or **gridftp** - if it has to be accessed remotely, etc), the corresponding port numbers to be used and the physical file names (SURLs) corresponding to the accessible input files from the CE where the job is running.

The *.BrokerInfo* file provides to the application a set of methods to resolve the LFN into a set of possible corresponding SURLs (getLFN2SFN).

Note: That by definition the SFN corresponds to an SURL but it does not contain the prefix "srm:". Neither the BrokerInfo API nor CLI distinguishes between SFN and SURL and thus they are identical here.

In addition, there exists a Command Line Interface (CLI) called **edg-brokerinfo** that is equivalent to the C++ API. It can be used to get information on how to access data, compliant with the chosen CE. The CLI methods can be invoked directly on the Worker Node (where the *.BrokerInfo* file actually is held), and - similarly to the C++ API - do not actually re-perform the matchmaking, but just read the *.BrokerInfo* file to get the result of the matchmaking process.

Note: That the CLI and the API can only be successfully used on the WN where the .BrokerInfo file exists. You will not be able to use the tool on the UI.

In this example exercise we take a look at the *.BrokerInfo* file on the Worker Node of the destination CE, and examine its various fields.

The vary basic JDL we are going to use is the following one (*brokerinfo.jdl*):

```
Executable     = "/bin/cat";
StdOutput      = "message.txt";
StdError       = "stderror.log";
OutputSandbox = {"message.txt", "stderror.log"};
Arguments      = " .BrokerInfo";
```

The corresponding set of commands we have to issue are as follows:

```
$ grid-proxy-init
$ edg-job-submit brokerinfo.jdl
$ edg-job-get-output <JobId>
```

The BrokerInfo file is a mechanism by which the user job can access, at execution time, certain information concerning the job, for example the name of the CE, the files specified in the **InputData** attribute, the SEs where they can be found, etc.

The BrokerInfo file is created in the job working directory (that is, the current directory on the WN for the executable) and is named *.BrokerInfo*. Its syntax is, as in job description files, based on Condor ClassAds and the information contained is not easy to read; however, it is possible to get it by means of a CLI, whose description follows.

Detailed information about the BrokerInfo file, the **edg-brokerinfo** CLI, and its respective API can be found in [R17].

The **edg-brokerinfo** command has the following syntax:

```
edg-brokerinfo [-v] [-f <filename>] function [parameter] [parameter] ...
```

where **function** is one of the following:

getCE	returns the name of the CE the job is running on.
getDataAccessProtocol	returns the protocol list specified in the **DataAccessProtocol** JDL attribute.
getInputData	returns the file list specified in the **InputData** JDL.
getSEs	returns the list of the storage elements with contain a copy of at least one file among those specified in **InputData**.
getCloseSEs	returns a list of the storage elements close to the CE.
getSEMountPoint <**SE**>	returns the access point for the specified <**SE**>, if it is the list of close SEs of the WN.
getSEFreeSpace <**SE**>	returns the free space on <**SE**>.
getLFN2SFN <**LFN**>	returns the storage file name of the file specified by <**LFN**>, where <**LFN**> is a logical file name of a GUID specified in the **InputData** attribute.
getSEProtocols <**SE**>	returns the list of the protocols available to transfer data in the storage element <**SE**>.
getSEPort <**SE**> <**Protocol**>	returns the port number used by <**SE**> for the data transfer protocol <**Protocol**>.
getVirtualOrganization	returns the name of the VO specified in the **VirtualOrganization** JDL attribute.
getAccessCost	not supported at present.

The **-v** option produced a more verbose output, and the **-f** <**filename**> option tells the command to parse the BrokerInfo file specified by <**filename**>. If the **-f** option is not used, the command tries to parse the file `$EDG_WL_RB_BROKERINFO`.

There are basically two ways for parsing elements from a BrokerInfo file.

The first one is directly from the job, and therefore from the WN where the job is running. In this case, the `$EDG_WL_RB_BROKERINFO` variable is defined as the location of the **.BrokerInfo** file, in the working directory of the job, and the command will work without problems. This can be accomplished for instance by including a line like the following in a submitted shell script:

```
/opt/edg/bin/edg-brokerinfo getCE
```

where the **edg-brokerinfo** command is called with any desired function as its argument.

If, on the contrary, **edg-brokerinfo** is invoked from the UI, the `$EDG_WL_RB_BROKERINFO` variable will be usually undefined, and an error will occur. The solution to this is to include an instruction to generate the *.BrokerInfo* file as ouput of the submitted job, and retrieve it with the rest of generated output, when the job finishes. This can be done for instance with:

```
#!/bin/sh
cat \$EDG_WL_RB_BROKERINFO
```

in a submitted shell script.

Then, the file can be accessed locally with the **-f** option commented above.

4.8. Exercise DM-7: Use Case - Read Data on the Grid

In this exercise you are going to copy a file from a non Grid site to a remote SE. On replicating the file to various sites, you will retrieve the best file, read the contents and print it on the screen.

Note: In this exercise we only list a set of steps that will guide you through the use case. We do not provide a detailed list of commands but leave it as a challenge for you to solve the exercise in the best possible way.

This exercise requires the following steps:

Pre-Use Case Steps

These steps are necessary to set up the test data for the use case.

- Create a local test file.
- Place the test file on several SEs remote to the UI (or CE for a job) where you are working. [Hint: Use a specific LFN to keep track of the replicas. You will need the LFN later on.]

Use Case Steps

We assume that replicas are created at several remote SEs.

- Replicate the best file to the close SE. [Hint: use **getBestFile**]
- Extract the SFN from the SURL that is returned from **getBestFile**. [Hint: the SFN is the SURL without the prefix “srm://”]
- Obtain the TURL for the SURL. [Hint: use **getTurl**]
- Open the file for reading using the TURL.
- Print the file contents on the screen.

4.9. Exercise DM-8: Use Case - Copy and Register Job Output Data

In this exercise you are going to write a job `job1` that produces several output files that are afterwards copied and registered to various SEs. Next you are going to write a second job that reads the best files of `job1` and prints the output on the screen. This is a typical use case of centralised data production where the result is distributed to various sites that are part of a particular Virtual Organisation. These production results are later analysed by users distributed all over the globe.

Note: In this exercise we only list a set of steps that will guide you through the use case. We do not provide a detailed list of commands but leave it as a challenge for you to solve the exercise in the best possible way.

The following steps are necessary for implementing this use case:

- Write a simple job `job1` that produces 5 output files with random numbers between 0 and 100.
- Copy and register these files to various SEs at the end of the job.
- Register metadata about file size, owner and description of the files.
- Write a second job `job2` that reads in the best files of `job1` and prints the output on the screen. [Hint: This step is similar to Exercise 9.]

CHAPTER 5

INFORMATION SYSTEM

5.1. INTRODUCTION

The resources described up to now constitute the compute and storage power of the LCG-2 Grid. Together with that infrastructure, additional services are provided to locate and report on the status of Grid resources, to find the most appropriate resources to run a job requiring certain data access and to automatically perform data operations necessary before and after a job is run. These are the *Information System* services.

The Information System (IS) provides information about the LCG-2 Grid resources and their status. In LCG-2, the *Monitoring and Discovery Service (MDS)* from Globus [R8] has been adopted as the provider of this service. Information is propagated in a hierarchy: Compute and storage resources at a site report (via the *Grid Resource Information Servers (GRISes)*) their static and dynamic status to the *Site Grid Index Information Server (GIIS)*.

Due to dynamic nature of the GRID, the GIISes might not contain information about resources that are actually available on the Grid but that, for some reasons, are unable to publish updated information to the GIISes. Because of this, the *Berkeley DB Information Index (BDII)* was introduced. The BDII queries registered GIISes and acts as a cache storing information about the Grid status in its database. Every time a resource appears in one of the GIISes, its existence is registered in one of the BDIIs. There is one BDII running at each site where a Resource Broker is installed. Users and other Grid services (such as the RB) can interrogate BDIIs to get information about the Grid status. Very up-to-date information can be found by directly interrogating the site GIISes or the local GRISes that run on the specific resources.

A lot of web pages show status information about Grid sites. The LCG *Grid Operations Centre (GOC)* will be responsible for coordinating the overall operation of the LCG. One of its responsibilities is monitoring. This information can be access through the following web page:

```
http://http://goc.grid-support.ac.uk/gridsite/gocmain/monitoring/
```

In the following sections examples are given on how to interrogate the Information System in LCG-2 Grid. In particular, the different servers from which the information can be obtained are discussed. These are the local GRISes, the site GIISes and the global BDIIs. The data in the IS of LCG-2 conforms to the GLUE Schema. For a list of GLUE Schema elements (*objectclasses*) and their attributes, check Appendix A.

Since the Information System “talks” the *Lightweight Directory Access Protocol (LDAP)* tools which also talks LDAP are needed to interrogate the IS. The command line tool **ldapsearch** is the tool which is used for the examples presented in this section. Note, however that there are also tools with a Graphical

User Interface, e.g. my favourite tool **JXplorer** (http://pegacat.com/jxplorer/) which is installed under */usr/public/bin/jxplorer.sh* on the nikhef computers.

5.2. The Local GRIS

The local Grid Resource Information Services running on Computing Elements and Storage Elements at the different sites report information on the characteristics and status of the services. They give both static and dynamic information.

In order to interrogate the GRIS on a specific Grid Element, the hostname of the Grid Element and the TCP port where the GRIS run must be specified. Such port is always **2135**. The following command can be used:

```
ldapsearch -x -h <hostname> -p 2135 -b "mds-vo-name=local, o=grid"
```

where the **-x** option indicates that simple authentication (instead of LDAP's SASL) should be used; the **-h** and **-p** options precede the hostname and port respectively; and the **-b** option is used to specify the initial search node in the LDAP tree.

The same effect can be obtained with:

```
ldapsearch -x -H <LDAP_URI> -b "mds-vo-name=local, o=grid"
```

where the hostname and port are included in the **-H** <**LDAP_URI**> option, avoiding the use of **-h** and **-p**.

Interrogating the GRIS on a Computing Element

The command used to interrogate the GRIS located on the NIKHEF SE *tbn17.nikhef.nl* is:

```
$ ldapsearch -x -h tbn17.nikhef.nl -p 2135 -b "mds-vo-name=local, o=grid"
```

or:

```
$ ldapsearch -x -H ldap://tbn17.nikhef.nl:2135 \
> -b "mds-vo-name=local, o=grid"
```

And the obtained reply will be:

```
version: 2

#
# filter: (objectclass=*)
# requesting: ALL
#

# tbn17.nikhef.nl, local, grid
dn: GlueSEUniqueID=tbn17.nikhef.nl,Mds-Vo-name=local,o=grid
objectClass: GlueSETop
objectClass: GlueSE
objectClass: GlueInformationService
objectClass: Gluekey
objectClass: GlueSchemaVersion
GlueSEUniqueID: tbn17.nikhef.nl
```

```
GlueSEName: nikhef.nl:disk
GlueSEPort: 2811
GlueInformationServiceURL: ldap://tbn17.nikhef.nl:2135/Mds-Vo-name=local,o=gri
 d
GlueForeignKey: GlueSLUniqueID=tbn17.nikhef.nl
GlueSchemaVersionMajor: 1
GlueSchemaVersionMinor: 1

# gsiftp, tbn17.nikhef.nl, local, grid
dn: GlueSEAccessProtocolType=gsiftp,GlueSEUniqueID=tbn17.nikhef.nl,Mds-Vo-name
 =local,o=grid
objectClass: GlueSETop
objectClass: GlueSEAccessProtocol
[...]
GlueSchemaVersionMajor: 1
GlueSchemaVersionMinor: 1

# local, grid
dn: Mds-Vo-name=local,o=grid
objectClass: GlobusStub

# search result
search: 2
result: 0 Success

# numResponses: 12
# numEntries: 11
```

In order to restrict the search to a specific objectclass, a filter of the form `'objectclass=<name>'` can be used. By specifying a list of attribute names, the reply is limited to the value of those attributes for the corresponding objectclass, as is shown in the next example. A description of all objectclasses and their attributes to optimize the LDAP search command can be found in Appendix A.

GETTING INFORMATION ABOUT THE SITE NAME FROM THE GRIS ON A COMPUTING ELEMENT

```
$ ldapsearch -x -h tbn18.nikhef.nl -p 2135 \
> -b "mds-vo-name=local, o=grid" 'objectclass=SiteInfo' siteName
version: 2

#
# filter: objectclass=SiteInfo
# requesting: siteName
#

# tbn18.nikhef.nl/siteinfo, local, grid
dn: in=tbn18.nikhef.nl/siteinfo,Mds-Vo-name=local,o=grid
siteName: nikhef.nl

# search result
search: 2
result: 0 Success
```

```
# numResponses: 2
# numEntries: 1
```

By adding the **-LLL** option we can avoid the comments and the version information in the reply:

```
$ ldapsearch -LLL -x -h tbn18.nikhef.nl -p 2135 \
> -b "mds-vo-name=local, o=grid" 'objectclass=SiteInfo' siteName
dn: in=tbn18.nikhef.nl/siteinfo,Mds-Vo-name=local,o=grid
siteName: nikhef.nl
```

5.3. THE SITE GIIS

At each site, a site Grid Information Index Service collects information about all resources present at a site (i.e. data from all GRISes of the site).

For a list of a number of LCG-2 sites and their resources present, please refer to:
http://grid-deployment.web.cern.ch/grid-deployment/cgi-bin/index.cgi?var=gis/lcg2Status

Usually a site GIIS runs on a Computing Element. In order to interrogate the site GIIS for NIKHEF, one needs to find out the name of that CE. This can be found in the web page reporting the site status, e.g.: http://http://www.dutchgrid.nl/Org/Nikhef/lcg-2.html

The port used to interrogate a site GIIS is usually the same as that of GRISes: **2135**. In order to interrogate the GIIS (and not the local GRIS) a different base name must be used (instead of `mds-vo-name=local, o=grid`). This base name is just the site name, which is published by all sites, where all "-" characters have been removed. So, for instance, for nikhef, the site name is *nikheflcgprod* and the mds base name is `mds-vo-name=nikheflcgprod, o=grid`.

As we can see in Figure 5.1, the CE name is *tbn18.nikhef.nl*.

So, in order to interrogate the site GIIS, we can use the command shown in the following example:

INTERROGATING THE SITE GIIS

```
$ ldapsearch -x -H ldap://tbn18.nikhef.nl:2135 \
> -b "mds-vo-name=nikheflcgprod,o=grid"
version: 2

#
# filter: (objectclass=*)
# requesting: ALL
#

# tbn18.nikhef.nl/siteinfo, nikheflcgprod, grid
dn: in=tbn18.nikhef.nl/siteinfo,Mds-Vo-name=nikheflcgprod,o=grid
objectClass: SiteInfo
objectClass: DataGridTop
objectClass: DynamicObject
siteName: nikhef.nl
sysAdminContact: grid-support-admin@nikhef.nl
userSupportContact: grid-support-admin@nikhef.nl
siteSecurityContact: grid-support-admin@nikhef.nl
dataGridVersion: LCG-2_0_0
installationDate: 20040413110OZ
```

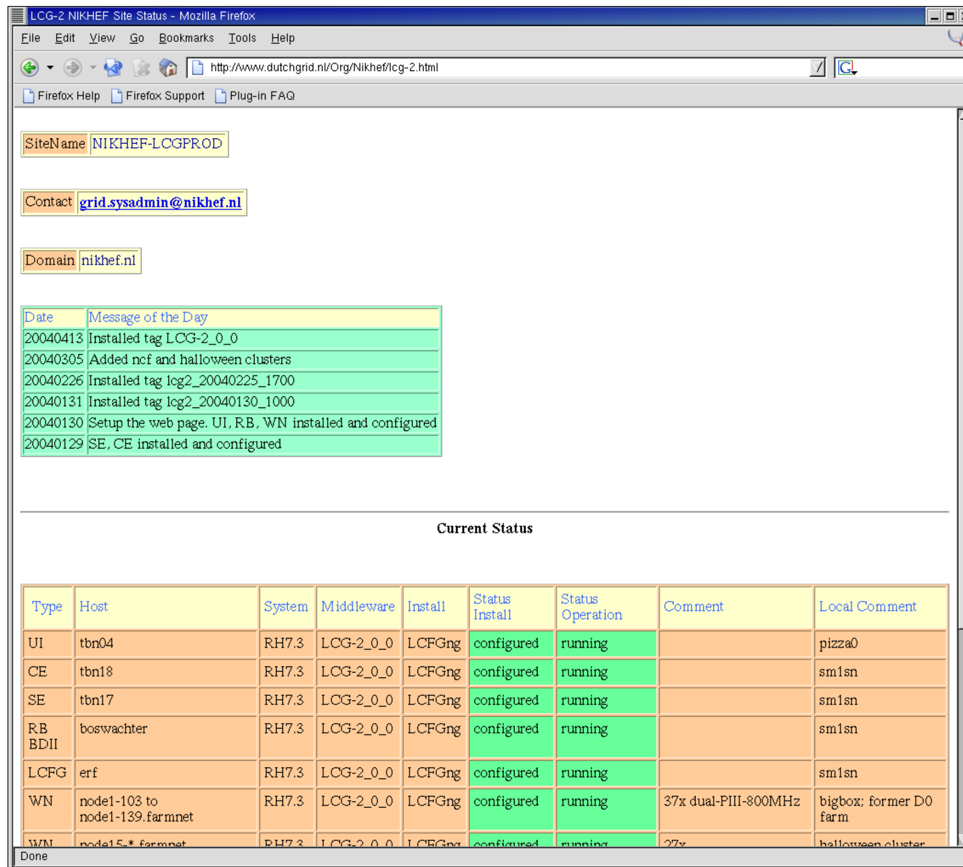

Figure 5.1: Webpage showing NIKHEF LCG-2 site status.

```
# tbn18.nikhef.nl:2119/jobmanager-pbs-qshort, nikheflcgprod, grid
dn: GlueCEUniqueID=tbn18.nikhef.nl:2119/jobmanager-pbs-qshort, Mds-Vo-name=nik
 heflcgprod,o=grid
objectClass: GlueCETop
objectClass: GlueCE
objectClass: GlueSchemaVersion
objectClass: GlueCEAccessControlBase
[...]
GlueSchemaVersionMajor: 1
GlueSchemaVersionMinor: 1

# nikheflcgprod, grid
dn: Mds-Vo-name=nikheflcgprod, o=grid
objectClass: GlobusStub

# search result
search: 2
result: 0 Success

# numResponses: 21
# numEntries: 20
```

5.4. THE BDII

Each site running a Resource Broker runs as well a Berkley Database Information Index that collects all information coming from the Regional GIISes and stores them in a permanent database. In order to find out the location of the BDII you can consult the web page of the LCG-2 site status as done for the site GIISes.

The BDII can be interrogated using the standard mds base: `mds-vo-name=local, o=grid`, and the BDII port: **2170**.

INTERROGATING A BDII

In this example, two attributes from the **GlueCESEBind** object class are retrieved for all sites.

```
$ ldapsearch -x -LLL -H ldap://boswachter.nikhef.nl:2170 \
> -b "mds-vo-name=local,o=grid" 'objectclass=GlueCESEBind' \
> GlueCESEBindCEUniqueID GlueCESEBindSEUniqueID
dn: GlueCESEBindSEUniqueID=lcg-se.usc.cesga.es, GlueCESEBindGroupCEUniqueID=lc
 g-ce.usc.cesga.es:2119/jobmanager-lcgpbs-short, Mds-Vo-name=usclcg2,mds-vo-na
 me=local,o=grid
GlueCESEBindCEUniqueID: lcg-ce.usc.cesga.es:2119/jobmanager-lcgpbs-short
GlueCESEBindSEUniqueID: lcg-se.usc.cesga.es

dn: GlueCESEBindSEUniqueID=gtbcg13.ifca.unican.es, GlueCESEBindGroupCEUniqueID
 =gtbcg12.ifca.unican.es:2119/jobmanager-lcgpbs-long, Mds-Vo-name=ifcalcg2,mds
 -vo-name=local,o=grid
GlueCESEBindCEUniqueID: gtbcg12.ifca.unican.es:2119/jobmanager-lcgpbs-long
GlueCESEBindSEUniqueID: gtbcg13.ifca.unican.es

dn: GlueCESEBindSEUniqueID=lcg-se.usc.cesga.es, GlueCESEBindGroupCEUniqueID=lc
[...]
GlueCESEBindSEUniqueID: wacdr002d.cern.ch

dn: GlueCESEBindSEUniqueID=lxn1183.cern.ch, GlueCESEBindGroupCEUniqueID=lxn118
 4.cern.ch:2119/jobmanager-lcglsf-grid, Mds-Vo-name=cernlcg2,mds-vo-name=local
 ,o=grid
GlueCESEBindCEUniqueID: lxn1184.cern.ch:2119/jobmanager-lcglsf-grid
GlueCESEBindSEUniqueID: lxn1183.cern.ch
```

LISTING ALL THE CES WHICH PUBLISH A GIVEN TAG QUERYING THE BDII

The attribute **GlueHostApplicationSoftwareRunTimeEnvironment** can be used to publish experiment-specific information (*tag*) on a CE, for example that a given experiment software is installed. To list all the CEs which publish a given tag, a query to the BDII can be performed. In this example, that information is retrieved for all the subclusters:

```
$ ldapsearch -h boswachter.nikhef.nl -p 2170 \
> -b "mds-vo-name=local,o=grid" -x 'objectclass=GlueSubCluster' \
> GlueChunkKey GlueHostApplicationSoftwareRunTimeEnvironment
```

LISTING ALL THE SES WHICH SUPPORT A GIVEN VO

A storage element supports a VO if users of that VO are allowed to store files on that SE. It is possible to find out which SEs support a VO with a query to the BDII. For example, to have the list af all SEs supporting ATLAS, the **GlueSAAccessControlBaseRule**, which specifies a supported VO, is used:

```
$ ldapsearch -h boswachter.nikhef.nl -p 2170 \
> -b "mds-vo-name=local,o=grid" -x 'objectclass=GlueSATop' \
> GlueChunkKey GlueSAAccessControlBaseRule | \
> grep -B 4 'GlueSAAccessControlBaseRule: atlas'
```

References

[R1] LCG-1 User Guide
http://grid-deployment.web.cern.ch/grid-deployment/eis/docs/LCG-1-UserGuide.htm

[R2] Regional Centres for LHC computing
The MONARC Architecture Group
http://barone.home.cern.ch/barone/monarc/RCArchitecture.html
http://monarc.web.cern.ch/MONARC/

[R3] The Anatomy of the Grid.
Enabling Scalable Virtual Organizations
Ian Foster, Carl Kesselman, Steven Tuecke
http://www.globus.org/research/papers/anatomy.pdf

[R4] Overview of the Grid Security Infrastructure
http://www-unix.globus.org/security/overview.html

[R5] Resource Management
http://www-unix.globus.org/developer/resource-management.html

[R6] WP1 Workload Management Software – Administrator and User Guide. Nov 24th, 2003
http://server11.infn.it/workload-grid/docs/DataGrid-01-TEN-0118-1_2.pdf

[R7] The GridFTP Protocol and Software
http://www.globus.org/datagrid/gridftp.html

[R8] MDS 2.2 Features in the Globus Toolkit 2.2 Release
http://www.globus.org/mds/

[R9] European DataGrid Project
http://eu-datagrid.web.cern.ch/eu-datagrid/

[R10] The GLUE schema
http://www.cnaf.infn.it/~sergio/datatag/glue/

[R11] LCG-2 Manual Installation Guide
https://edms.cern.ch/file/434070//LCG2Install.pdf

[R12] Classified Advertisements. Condor.
http://www.cs.wisc.edu/condor/classad

[R13] The Condor Project.
http://www.cs.wisc.edu/condor/

[R14] Job Description language HowTo. December 17th, 2001
http://server11.infn.it/workload-grid/docs/DataGrid-01-TEN-0102-0_2-Document.pdf

[R15] JDL Attributes – Release 2.x. Oct 28th, 2003
http://server11.infn.it/workload-grid/docs/DataGrid-01-TEN-0142-0_2.pdf

[R16] WP1 Workload Management System – Job Partitioning and Checkpointing. June 3, 2002
https://edms.cern.ch/file/347730/1/DataGrid-01-TED-0119-0_3.pdf

[R17] The EDG-Brokerinfo User Guide - Release 2.x. 6th August 2003
http://server11.infn.it/workload-grid/docs/edg-brokerinfo-user-guide-v2_2.pdf

[R18] Workload Management Software – GUI User Guide. Nov 24th, 2003
http://server11.infn.it/workload-grid/docs/DataGrid-01-TEN-0143-0_0.pdf

[R19] User Guide for the EDG Replica Manager 1.5.x
http://edg-wp2.web.cern.ch/edg-wp2/replication/docu/r2.1/edg-replica-manager-userguide.pdf

[R20] User Guide for the EDG Local Replica Catalog 2.1.x
http://edg-wp2.web.cern.ch/edg-wp2/replication/docu/r2.1/edg-lrc-userguide.pdf

[R21] User Guide for the EDG Replica Metadata Catalog 2.1.x
http://edg-wp2.web.cern.ch/edg-wp2/replication/docu/r2.1/edg-rmc-userguide.pdf

[R22] EDG Tutorial – Handout for Participants for EDG Release 2.x
http://edms.cern.ch/document/393671

[R23] User Guide for the EDG Replica Optimization Service 2.1.x
http://edg-wp2.web.cern.ch/edg-wp2/replication/docu/r2.1/edg-ros-userguide.pdf

[R24] User Guide for the Replica Location Index 2.1.x
http://edg-wp2.web.cern.ch/edg-wp2/replication/docu/r2.1/edg-rli-userguide.pdf

[R25] Developer Guide for the EDG Replica Manager 1.5.x
http://edg-wp2.web.cern.ch/edg-wp2/replication/docu/r2.1/edg-replica-manager-devguide.pdf

[R26] Remote File Stream. Extensions to the Standard C++ I/O Library for Accessing Remote Files
http://doc.in2p3.fr/doc/public/products/rfstream/rfstream.html

[R27] POOL - Persistency Framework. Pool Of persistent Objects for LHC.
http://lcgapp.cern.ch/project/persist
Learning POOL by examples, a mini tutorial.
http://lcgapp.cern.ch/project/persist/tutorial/learningPoolByExamples.html

Acknowledgements

APPENDIX A

THE GLUE SCHEMA

As explained earlier, the GLUE Schema describes what data about the elements in the Grid is stored for its use by the Information System.

In this section, all the objectclasses of the LDAP hierarchy tree for the GLUE schema are described. The attributes for each one of the objectclasses (where the dynamic data is actually stored) are presented. The objectclasses are grouped in CE attributes, SE attributes and CE-SE binding attributes. Some of the attributes may actually be empty, even if they are defined in the schema.

A.1. ATTRIBUTES FOR THE COMPUTING ELEMENT

- *CE (objectclass **GlueCE**)*
  - **GlueCEUniqueID**: unique identifier for the CE
  - **GlueCEName**: human-readable name of the service
- *Info (objectclass **GlueCEInfo**)*
  - **GlueCEInfoLRMSType**: name of the local batch system
  - **GlueCEInfoLRMSVersion**: version of the local batch system
  - **GlueCEInfoGRAMVersion**: version of GRAM
  - **GlueCEInfoHostName**: fully qualified name of the host where the gatekeeper runs
  - **GlueCEInfoGateKeeperPort**: port number for the gatekeeper
  - **GlueCEInfoTotalCPUs**: number of CPUs in the cluster associated to the CE
- *State (objectclass **GlueCEState**)*
  - **GlueCEStateRunningJobs**: number of running jobs
  - **GlueCEStateWaitingJobs**: number of jobs not running
  - **GlueCEStateTotalJobs**: total number of jobs (running + waiting)
  - **GlueCEStateStatus**: queue status: queueing (jobs are accepted but not run), production (jobs are accepted and run), closed (jobs are neither accepted nor run), draining (jobs are not accepted but those in the queue are run)

- **GlueCEStateWorstResponseTime**: worst possible time between the submission of a job and the start of its execution
- **GlueCEStateEstimatedResponseTime**: estimated time between the submission of a job and the start of its execution
- **GlueCEStateFreeCPUs**: number of CPUs available to the scheduler

- *Policy (objectclass* ***GlueCEPolicy****)*
  - **GlueCEPolicyMaxWallClockTime**: maximum wall clock time available to jobs submitted to the CE
  - **GlueCEPolicyMaxCPUTime**: maximum CPU time available to jobs submitted to the CE
  - **GlueCEPolicyMaxTotalJobs**: maximum allowed total number of jobs in the queue
  - **GlueCEPolicyMaxRunningJobs**: maximum allowed number of running jobs in the queue
  - **GlueCEPolicyPriority**: information about the service priority
- *Access control (objectclass* ***GlueCEAccessControlBase****)*
  - **GlueCEAccessControlBaseRule**: a rule defining any access restrictions to the CE. Current semantic: VO = a VO name, DENY = an X.509 user subject
- *Job (currently not filled, the Logging and Bookkeeping service can provide this information) (objectclass* ***GlueCEJob****)*
  - **GlueCEJobLocalOwner**: local user name of the job's owner
  - **GlueCEJobGlobalOwner**: GSI subject of the real job's owner
  - **GlueCEJobLocalID**: local job identifier
  - **GlueCEJobGlobalId**: global job identifier
  - **GlueCEJobGlueCEJobStatus**: job status: **SUBMITTED**, **WAITING**, **READY**, **SCHEDULED**, **RUNNING**, **ABORTED**, **DONE**, **CLEARED**, **CHECKPOINTED**
  - **GlueCEJobSchedulerSpecific**: any scheduler specific information
- *Cluster (objectclass* ***GlueCluster****)*
  - **GlueClusterUniqueID**: unique identifier for the cluster
  - **GlueClusterName**: human-readable name of the cluster
- *Subcluster (objectclass* ***GlueSubCluster****)*
  - **GlueSubClusterUniqueID**: unique identifier for the subcluster
  - **GlueSubClusterName**: human-readable name of the subcluster
- *Host (objectclass* ***GlueHost****)*
  - **GlueHostUniqueId**: unique identifier for the host
  - **GlueHostName**: human-readable name of the host
- *Architecture (objectclass* ***GlueHostArchitecture****)*
  - **GlueHostArchitecturePlatformType**: platform description

  - **GlueHostArchitectureSMPSize**: number of CPUs
- *Processor (objectclass **GlueHostProcessor**)*
  - **GlueHostProcessorVendor**: name of the CPU vendor
  - **GlueHostProcessorModel**: name of the CPU model
  - **GlueHostProcessorVersion**: version of the CPU
  - **GlueHostProcessorOtherProcessorDescription**: other description for the CPU
  - **GlueHostProcessorClockSpeed**: clock speed of the CPU
  - **GlueHostProcessorInstructionSet**: name of the instruction set architecture of the CPU
  - **GlueHostProcessorGlueHostProcessorFeatures**: list of optional features of the CPU
  - **GlueHostProcessorCacheL1**: size of the unified L1 cache
  - **GlueHostProcessorCacheL1I**: size of the instruction L1 cache
  - **GlueHostProcessorCacheL1D**: size of the data L1 cache
  - **GlueHostProcessorCacheL2**: size of the unified L2 cache
- *Application software (objectclass **GlueHostApplicationSoftware**)*
  - **GlueHostApplicationSoftwareRunTimeEnvironment**: list of software installed on this host
- *Main memory (objectclass **GlueHostMainMemory**)*
  - **GlueHostMainMemoryRAMSize**: physical RAM
  - **GlueHostMainMemoryRAMAvailable**: unallocated RAM
  - **GlueHostMainMemoryVirtualSize**: size of the configured virtual memory
  - **GlueHostMainMemoryVirtualAvailable**: available virtual memory
- *Benchmark (objectclass **GlueHostBenchmark**)*
  - **GlueHostBenchmarkSI00**: SpecInt2000 benchmark
  - **GlueHostBenchmarkSF00**: SpecFloat2000 benchmark
- *Network adapter (objectclass **GlueHostNetworkAdapter**)*
  - **GlueHostNetworkAdapterName**: name of the network card
  - **GlueHostNetworkAdapterIPAddress**: IP address of the network card
  - **GlueHostNetworkAdapterMTU**: the MTU size for the LAN to which the network card is attached
  - **GlueHostNetworkAdapterOutboundIP**: permission for outbound connectivity
  - **GlueHostNetworkAdapterInboundIP**: permission for inbound connectivity
- *Processor load (objectclass **GlueHostProcessorLoad**)*
  - **GlueHostProcessorLoadLast1Min**: one-minute average processor availability for a single node

  - **GlueHostProcessorLoadLast5Min**: 5-minute average processor availability for a single node
  - **GlueHostProcessorLoadLast15Min**: 15-minute average processor availability for a single node

- *SMP load (objectclass **GlueHostSMPLoad**)*
  - **GlueHostSMPLoadLast1Min**: one-minute average processor availability for a single node
  - **GlueHostSMPLoadLast5Min**: 5-minute average processor availability for a single node
  - **GlueHostSMPLoadLast15Min**: 15-minute average processor availability for a single node
- *Operating system (objectclass **GlueHostOperatingSystem**)*
  - **GlueHostOperatingSystemOSName**: OS name
  - **GlueHostOperatingSystemOSRelease**: OS release
  - **GlueHostOperatingSystemOSVersion**: OS or kernel version
- *Local file system (objectclass **GlueHostLocalFileSystem**)*
  - **GlueHostLocalFileSystemRoot**: path name or other information defining the root of the file system
  - **GlueHostLocalFileSystemSize**: size of the file system in bytes
  - **GlueHostLocalFileSystemAvailableSpace**: amount of free space in bytes
  - **GlueHostLocalFileSystemReadOnly**: true if the file system is read-only
  - **GlueHostLocalFileSystemType**: file system type
  - **GlueHostLocalFileSystemName**: the name for the file system
  - **GlueHostLocalFileSystemClient**: host unique id of clients allowed to remotely access this file system
- *Remote file system (objectclass **GlueHostRemoteFileSystem**)*
  - **GlueHostLRemoteFileSystemRoot**: path name or other information defining the root of the file system
  - **GlueHostRemoteFileSystemSize**: size of the file system in bytes
  - **GlueHostRemoteFileSystemAvailableSpace**: amount of free space in bytes
  - **GlueHostRemoteFileSystemReadOnly**: true if the file system is read-only
  - **GlueHostRemoteFileSystemType**: file system type
  - **GlueHostRemoteFileSystemName**: the name for the file system
  - **GlueHostRemoteFileSystemServer**: host unique id of the server which provides access to the file system
- *Storage device (objectclass **GlueHostStorageDevice**)*
  - **GlueHostStorageDeviceName**: name of the storage device
  - **GlueHostStorageDeviceType**: storage device type
  - **GlueHostStorageDeviceTransferRate**: maximum transfer rate for the device

  - **GlueHostStorageDeviceSize**: size of the device
  - **GlueHostStorageDeviceAvailableSpace**: amount of free space

- *File (objectclass **GlueHostFile**)*
  - **GlueHostFileName**: name for the file
  - **GlueHostFileSize**: file size in bytes
  - **GlueHostFileCreationDate**: file creation date and time
  - **GlueHostFileLastModified**: date and time of the last modification of the file
  - **GlueHostFileLastAccessed**: date and time of the last access to the file
  - **GlueHostFileLatency**: time taken to access the file in seconds
  - **GlueHostFileLifeTime**: time for which the file will stay on the storage device
  - **GlueHostFileOwner**: name of the owner of the file

A.2. ATTRIBUTES FOR THE STORAGE ELEMENT

- *Storage Service (objectclass **GlueSE**)*
  - **GlueSEUniqueId**: unique identifier of the storage service (URI)
  - **GlueSEName**: human-readable name for the service
  - **GlueSEPort**: port number that the service listens
  - **GlueSEHostingSL**: unique identifier of the storage library hosting the service
- *Storage Service State (objectclass **GlueSEState**)*
  - **GlueSEStateCurrentIOLoad**: system load (for example, number of files in the queue)
- *Storage Service Access Protocol (objectclass **GlueSEAccessProtocol**)*
  - **GlueSEAccessProtocolType**: protocol type to access or transfer files
  - **GlueSEAccessProtocolPort**: port number for the protocol
  - **GlueSEAccessProtocolVersion**: protocol version
  - **GlueSEAccessProtocolAccessTime**: time to access a file using this protocol
  - **GlueSEAccessProtocolSupportedSecurity**: security features supported by the protocol
- *Storage Library (objectclass **GlueSL**)*
  - **GlueSLName**: human-readable name of the storage library
  - **GlueSLUniqueId**: unique identifier of the machine providing the storage service
  - **GlueSLService**: unique identifier for the provided storage service
- *Local File system (objectclass **GlueSLLocalFileSystem**)*
  - **GlueSLLocalFileSystemRoot**: path name (or other information) defining the root of the file system

  - **GlueSLLocalFileSystemName**: name of the file system
  - **GlueSLLocalFileSystemType**: file system type (e.g. NFS, AFS, etc.)
  - **GlueSLLocalFileSystemReadOnly**: true is the file system is read-only
  - **GlueSLLocalFileSystemSize**: total space assigned to this file system
  - **GlueSLLocalFileSystemAvailableSpace**: total free space in this file system
  - **GlueSLLocalFileSystemClient**: unique identifiers of clients allowed to access the file system remotely
  - **GlueSLLocalFileSystemServer**: unique identifier of the server exporting this file system (only for remote file systems)

- *Remote File system (objectclass **GlueSLRemoteFileSystem**)*
  - **GlueSLRemoteFileSystemRoot**: path name (or other information) defining the root of the file system
  - **GlueSLRemoteFileSystemName**: name of the file system
  - **GlueSLRemoteFileSystemType**: file system type (e.g. NFS, AFS, etc.)
  - **GlueSLRemoteFileSystemReadOnly**: true is the file system is read-only
  - **GlueSLRemoteFileSystemSize**: total space assigned to this file system
  - **GlueSLRemoteFileSystemAvailableSpace**: total free space in this file system
  - **GlueSLRemoteFileSystemServer**: unique identifier of the server exporting this file system

- *File Information (objectclass **GlueSLFile**)*
  - **GlueSLFileName**: file name
  - **GlueSLFileSize**: file size
  - **GlueSLFileCreationDate**: file creation date and time
  - **GlueSLFileLastModified**: date and time of the last modification of the file
  - **GlueSLFileLastAccessed**: date and time of the last access to the file
  - **GlueSLFileLatency**: time needed to access the file
  - **GlueSLFileLifeTime**: file lifetime
  - **GlueSLFilePath**: file path

- *Directory Information (objectclass **GlueSLDirectory**)*
  - **GlueSLDirectoryName**: directory name
  - **GlueSLDirectorySize**: directory size
  - **GlueSLDirectoryCreationDate**: directory creation date and time
  - **GlueSLDirectoryLastModified**: date and time of the last modification of the directory
  - **GlueSLDirectoryLastAccessed**: date and time of the last access to the directory
  - **GlueSLDirectoryLatency**: time needed to access the directory
  - **GlueSLDirectoryLifeTime**: directory lifetime
  - **GlueSLDirectoryPath**: directory path

- *Architecture (objectclass **GlueSLArchitecture**)*
  - **GlueSLArchitectureType**: type of storage hardware (i.e. disk, RAID array, tape library, etc.)
- *Performance (objectclass **GlueSLPerformance**)*
  - **GlueSLPerformanceMaxIOCapacity**: maximum bandwidth between the service and the network
- *Storage Space (objectclass **GlueSA**)*
  - **GlueSARoot**: pathname of the directory containing the files of the storage space
- *Policy (objectclass **GlueSAPolicy**)*
  - **GlueSAPolicyMaxFileSize**: maximum file size
  - **GlueSAPolicyMinFileSize**: minimum file size
  - **GlueSAPolicyMaxData**: maximum allowed amount of data that a single job can store
  - **GlueSAPolicyMaxNumFiles**: maximum allowed number of files that a single job can store
  - **GlueSAPolicyMaxPinDuration**: maximum allowed lifetime for non-permanent files
  - **GlueSAPolicyQuota**: total available space
  - **GlueSAPolicyFileLifeTime**: lifetime policy for the contained files
- *State (objectclass **GlueSAState**)*
  - **GlueSAStateAvailableSpace**: total space available in the storage space (in kilobytes)
  - **GlueSAStateUsedSpace**: used space in the storage space (in kilobytes)
- *Access Control Base (objectclass **GlueSAAccessControlBase**)*
  - **GlueSAAccessControlBase Rule**: list of the access control rules

A.3. ATTRIBUTES FOR THE CE-SE BINDING

The CE-SE binding schema represents a mean for advertising relationships between a CE and a SE (or several SEs). This is defined by site administrators and is used when scheduling jobs that must access input files or create output files from or to SEs.

- *Associations between an CE and one or more SEs (objectclass **GlueCESEBindGroup**)*
  - **GlueCESEBindGroupCEUniqueID**: unique ID for the CE
  - **GlueCESEBindGroupSEUniqueID**: unique ID for the SE
- *Associations between an SE and a CE (objectclass **GlueCESEBind**)*
  - **GlueCESEBindCEUniqueID**: unique ID for the CE
  - **GlueCESEBindCEAccesspoint**: access point in the cluster from which CE can access a local SE
  - **GlueCESEBindSEUniqueID**: unique ID for the SE

APPENDIX B

JOB STATUS DEFINITION

As already mentioned in chapter 5, a job can find itself in one of several possible states, the definition of which is given in this table.

Status	*Definition*
SUBMITTED	The job has been submitted by the user but not yet processed by the Network Server
WAITING	The job has been accepted by the Network Server but not yet processed by the Workload Manager
READY	The job has been assigned to a Computing Element but not yet transferred to it
SCHEDULED	The job is waiting in the Computing Element's queue
RUNNING	The job is running
DONE	The job has finished
ABORTED	The job has been aborted by the WMS (e.g. because it was too long, or the proxy certificated expired, etc.)
CANCELLED	The job has been cancelled by the user
CLEARED	The Output Sandbox has been transferred to the User Interface

Only some transitions between states are allowed. These transitions are depicted in Figure B.1.

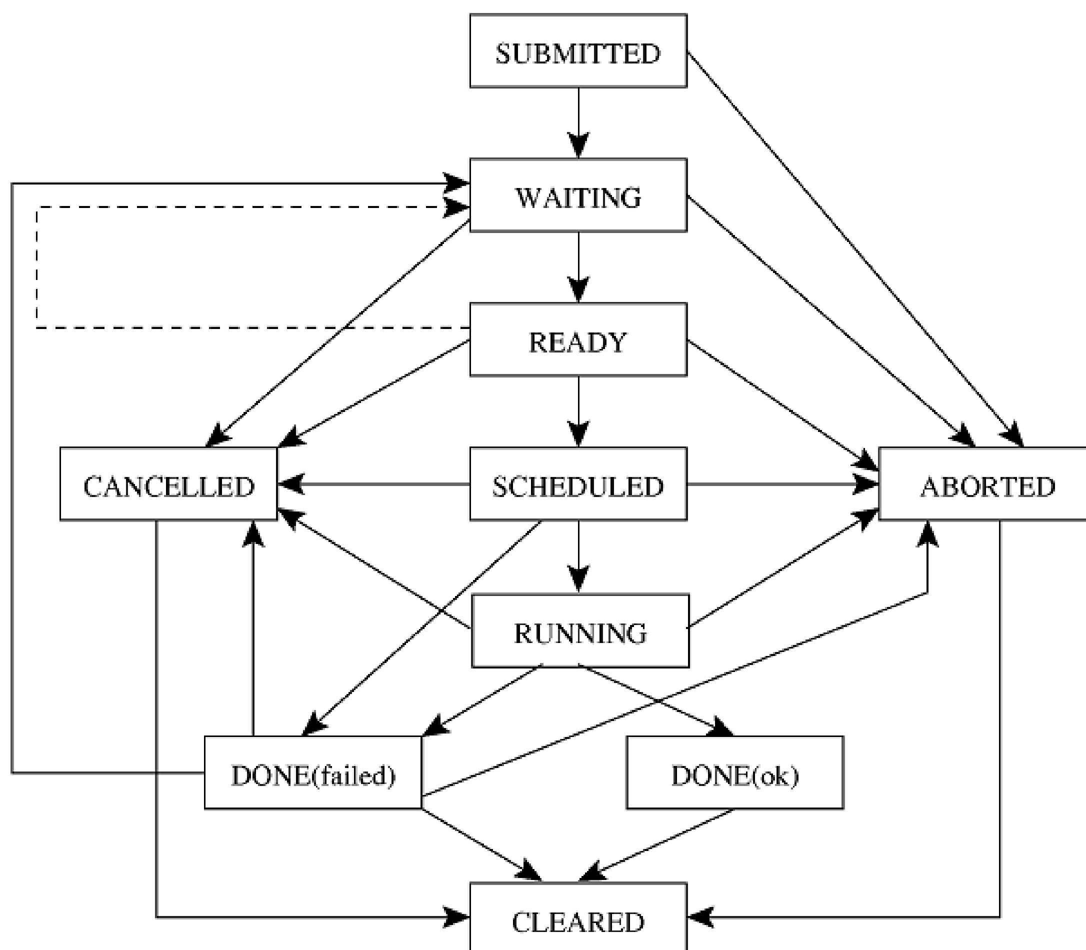

Figure B.1: Posible job states in the LCG-2