DataGrid

EDG Users' Guide

Integration Team (WP6)

Document identifier:	**DataGrid-06-TED-0109-2-3**
Date:	**October 29, 2003**
Workpackage:	**Integration Team (WP6)**
Partners:	**Contributions from all partners**

Abstract: This guide explains how to start using the European DataGrid (EDG) testbed, provides basic example of use, and indicates the location of more detailed documentation.

Change Log

Version	Date	Comment
1.0	15 Jan 2003	Version For EDG 1.4.3
2.0	04 Sep 2003	Draft For EDG 2.0.0
2.1	07 Sep 2003	Changes from Massimo Sgaravatto
2.2	12 Oct 2003	Response to comments from J. Linford, D. Boutigny, J. Linford, C. Leroy, and G. Moguilny
2.3	29 Oct 2003	Fix example 6.1. Must delete LFN explicitly.

Contents

List of Tables

List of Figures

1 Overview

The Grid makes widely distributed computing resources transparently available to the end-user. As well as purely computational resources, these include data storage and networking. The European DataGrid (EDG) collaboration builds software components which enable this access; the EDG testbeds are the vehicles for testing of the EDG software.

There are two testbeds: the development testbed and the application testbed. The application testbed, is intended for semi-production use by end-users. The development testbed is a key component of the development, integration, and certification process.

A number of sites (currently around 15 sites in 8 countries) provide the computational resources for the testbed. Each site provides services which are typically organized as follows:

User Interface (UI) This machine runs the User Interface software which allows the end-user to interact with the EDG testbed. This is typically the machine the end-user logs into to submit jobs to the grid and to retrieve the output from those jobs.

Computing Element or Service (CE) A computing element consists of one gatekeeper node and one or more worker nodes. Together these provide computational resources to the user.

Gatekeeper (GK) This is the frontend of a computing element, accepting jobs, dispatching them for execution, and returning the output. It provides a uniform, grid-accessible interface to the computational resources it manages.

Worker Node (WN) These nodes sit behind a gatekeeper and are typically managed via a local batch system. The details of this are hidden from the end-user by the gatekeeper; however, these are the nodes on which user computations are actually performed. Consequently, the end-user software is installed on these nodes. These nodes do not run any EDG daemons,[1] but do have client APIs for accessing EDG services and information.

Storage Element (SE) These nodes provide uniform, high-level access to data storage. The storage element may control large disk arrays, mass storage systems and the like; however, the SE interface hides the differences between these systems allowing uniform user access.

Monitoring Node (MON) The Monitoring Node runs the R-GMA servlets for the site and ROS, the replica optimization service.

The resources within a testbed site and the total number of sites change over time as new resources are added to the testbed or are temporarily withdrawn for reasons such as maintenance.

There are also several nodes which provide shared services and are not site-specific but shared by various subgroups of the testbed users. The most visible are the following:

Resource Broker (RB) These machines accept jobs from users (via the User Interface), match the jobs' requirements to the available testbed resources, and dispatch the jobs.

Replica Location Service (RLS) This machine runs LRC, the local replica catalog (one part of RLS), and RMC, the replica metadata catalog. The RLS maintains a database of a Virtual Organization's (see Section 2.3) data files and associated metadata. These services are used by users and by grid services to locate appropriate copies of input data files.

[1] Except for the Mercury daemon for the optional application monitoring package GRM.

Information Catalog (IC) The Information Catalog runs the R-GMA schema and registry servlets. The registry is where the URLs of all producers and consumers of information within the grid can be found. Both users and grid services use this extensively in the course of normal operation.

There are numerous other services which support, for example, the EDG security model, but which are used only indirectly by end-users.

2 Getting Started

This chapter provides a very brief summary on use of the EDG Testbed. The rest of the guide and the referenced documentation contain important details which a prudent reader will browse before starting. Before using the EDG Testbed resources you must do the following:

1. You must obtain a cryptographic certificate from an EDG-approved Certificate Authority (CA). (See Section 2.1.)
2. With your certificate loaded into a web browser, you must sign the EDG Usage Guidelines and register with at least one virtual organization. These can be done via the user registration page on the Testbed website[1]. See Section 2.3 to choose an appropriate Virtual Organization.
3. You must obtain an account on a machine which has the software to access the EDG testbed (a User Interface machine). If one is not available locally at your home institute, you may request an account on either the Lyon or RAL User Interface machines. (See Section 2.4.)

The following sections provide details of these prerequisites.

2.1 Obtaining a Certificate

Cryptographic certificates are used to attest to the identity of a user or machine to the extent specified in the issuing Certification Authority's (CA) policy documents. Users accessing EDG resources must have a valid certificate; similarly, machines providing grid services within the testbed must also have one.

For users, a certificate is the grid-equivalent of a passport. As such it is *personal* and should not be shared with anyone else. You should also ensure that the private key is kept private and that you choose a secure passphrase.

The EDG-approved CAs have service areas which cover most of Europe and the United States. (Consult the current list[2] on the web.) If a user or site is not covered by an existing CA's service area, then one must either start a new CA or negotiate with the French CA to extend its service area[3].

Note that the certificate application and delivery procedures for each of the CAs varies. Refer to your CA's web site for a description of it's procedures.

2.2 Installing User Certificates

2.2.1 Installing for use with Globus

The DataGrid relies on the Globus Security Infrastructure (GSI) to implement certificate management. To use the GSI you must have your certificate in PEM format. Follow the instructions in Appendix C if you need to change a P12-formatted certificate into a PEM-formatted certificate. You should then place the two files `usercert.pem` and `userkey.pem` into a `.globus` directory in your home area. The file permissions for the `userkey.pem` file must be 0600, for `usercert.pem` 0644 is appropriate.

[1] http://marianne.in2p3.fr/
[2] http://marianne.in2p3.fr/datagrid/ca/ca-table-ca.html
[3] Extending the service area is done only if the request involves a small number of certificates and there is someone to act as a registration authority for those certificates.

You may place your certificate and key in a non-standard location, but in this case you must define the two environmental variables X509_USER_CERT and X509_USER_KEY to point to your certificate and key, respectively before creating a proxy (see Section 2.4).

2.2.2 Importing Certificates into a Browser

Signing the EDG Usage Guidelines must be done via a SSL-protected web form. To gain access to this page you must have your certificate loaded into a web-browser. The procedure for loading a certificate into a browser varies greatly between browsers. Instructions are given here for several popular browsers.

Netscape (7.1), Mozilla (1.2.1) From the Edit menu select the Preferences... item. In the navigation panel on the left site of the dialog window, click on Privacy & Security. Under this category click on the Certificates entry. On the right side of the dialog, there will be a button labeled Manage Certificates...; click it. This will bring up another window. In the new window, select the Your Certificates tab and then click on the Import button. Use the file browser to select a P12 copy of your certificate. You will then be asked to enter (or create) a master password for the "Software Security Device" and to verify the password on the certificate itself. After entering the passwords, the certificate should appear in the Your Certificates panel.

Internet Explorer (5.00) Go to the Tools menu and select Internet Options. Choose the Content tab and click on the Organizations... button in the Certificates section. In the dialog click Import to start the Import wizard, then follow the instructions. Select the a P12 copy of your certificate for import and give the password (it should go into the "Personal" certificate store).

You should also select high security otherwise Internet Explorer remembers your pass phrase for you.

Opera (7.11) From the File menu, select the Preferences... item. In the preferences dialog, choose the Security item in the navigation panel on the left; click the Manage certificates... button. In the next dialog, click the Import... button. Use the file dialog to select a copy of your certificate and key in P12 format. You will be asked for the password protecting the certificate; enter it and then confirm that you want to import the certificate. You will then be asked to enter (or create) a password for your "security device." Enter this and then you should get a dialog confirming that you certificate has been imported. Unless you have previously imported your CA's public certificate, Opera will warn you that it cannot verify the certificate. This is not necessary for using the certificate.

2.3 Virtual Organizations

Virtual Organizations (VOs) are used to organize the testbed users into various subgroups and are the basis for grid *authorization*. When a user runs a task, the user's certificate information is compared with a file which is populated by information from the various VOs. "John Doe" may have been added to the Alice VO, in which case the file referred to will have an entry for "John Doe" along with a directive to map his requests onto a local Alice environment. On the other hand, John would not be allowed to run jobs under other environments.

The current list of virtual organizations[4] can be found on the web. If you did not register with a virtual organization when you signed the EDG Usage Guidelines (or wish to change your VO membership), then you must contact the VO manager directly.

If none of the listed virtual organizations is appropriate for you, use the WP6 VO. It is intended as a catch-all for folks who are not members of any of the others.

With the Testbed 2 software, membership in more than one virtual organization is possible. However the burden of specifying which virtual organization falls to the user. The local account is still determined by

[4] http://marianne.in2p3.fr/datagrid/vo/vo-table.html

the site's configuration and mismatches may cause problems when standard unix permissions are used by services.

2.4 "Logging into the Grid"

To access the resources of the EDG Testbed, you must have a machine available to you which has the User Interface tools installed. Ask your local site administrator if there is such a machine at your site. If not, EDG users may obtain an account on a UI machine at the Computer Center in Lyon[5] (ccedgui.in2p3.fr) or at RAL[6] (gppui04.gridpp.rl.ac.uk). People with an AFS account at CERN may use testbed010.cern.ch.

Your access rights to the grid services are tied to the subject name of your certificate. In essence, this is your "grid user name". Access is controlled via a proxy which is a time-limited credential signed by your private key. Grid services may take actions on your behalf if they have a copy of this proxy.

Two different methods can be used to generate a user proxy. A standard Globus proxy contains only information to verify your identify. The VOMS (Virtual Organization Membership Service) proxy contains additional authorization information about your memberships in virtual organizations and any groups or roles within those organizations. The VOMS method can only be used if your VO is running a VOMS server; contact your VO manager to determine if this is the case.

The proxy is a plain file stored by default in the `/tmp` area of the machine. For sites with a large number of workstations, it may be more convenient to store the proxy on a shared file system. This will allow you to generate your proxy once and use it on all of the workstations. To do so, define X509_USER_PROXY in your shell startup file to point to a file in the shared file system. For example,

```
export X509_USER_PROXY=~/.globus/proxy
```

for sh-shell variants. You can then use your standard proxy initialization command and it will generate the proxy in this location.

Once created, the proxy is copied automatically, when necessary, by the various grid commands. Having a copy of your proxy, a service can act on your behalf. Correspondingly anyone with a copy of your proxy can act as you while the proxy is valid. The proxy is initially created so that only you can read it; be careful not to loosen the permissions on the proxy or give anyone else a copy of the proxy.

You can remove the proxy with one of the proxy destruction commands or simply by deleting the proxy file. Note that this will only affect the local proxy and will not affect jobs running with their own copy of your proxy.

2.4.1 Globus Proxy

The **grid-proxy-*** commands allow you to manage a Globus proxy. All of these commands accept the *-help* option to give online usage information.

Initialization

If you have installed your certificate correctly, the command **grid-proxy-init** will create a new proxy for you. A successful attempt will look similar to the following:

```
>>  grid-proxy-init

Your identity: /C=FR/O=CNRS/OU=LAL/CN=Charles Loomis/Email=loomis@lal.in2p3.fr
```

[5] http://ccgrid.in2p3.fr/index.php?id=user_interface
[6] http://atlas.rl.ac.uk/csf/csfuserguide/usrreg.shtml

```
Enter GRID pass phrase for this identity: **********
Creating proxy ............................................... Done
Your proxy is valid until Tue Aug 13 03:15:11 2002
```

By default, the proxy will have a lifetime of 12 hours. Proxies with different lifefimes can be generated using the *-hours* option if desired. Note that you expose yourself to a greater chance of having your credentials hacked if you generate a proxy with a long lifetime.

If the certificate has not been installed correctly, then you will see a "user certificate not found" error. Check that you have followed the instructions correctly in Section 2.2.1. An incorrect passphrase will result in a "wrong pass phrase" error.

Information

To obtain information about a generated proxy, you can use the command **grid-proxy-info**:

```
>>  grid-proxy-info
subject  : /C=FR/O=CNRS/OU=LAL/CN=Charles Loomis/Email=loomis@lal.in2p3.fr/CN=proxy
issuer   : /C=FR/O=CNRS/OU=LAL/CN=Charles Loomis/Email=loomis@lal.in2p3.fr
type     : full
strength : 512 bits
timeleft : 11:36:17
```

Individual parts of this information can be selected using command options.

Destruction

To destroy explicitly the proxy before it has expired, use the command **grid-proxy-destroy**; the simpliest form takes no arguments. However this only destroys (erases) the local copy of the proxy. It does not affect copies of your proxy in use by your jobs or by grid services.

2.4.2 VOMS Proxy

The **edg-voms-proxy-*** commands allow you to manage a VOMS proxy. All of these commands accept the *-help* option to give online usage information.

Initialization

To initialize a VOMS proxy, use the command:

```
>> edg-voms-proxy-init
```

Information

To obtain information about your VOMS proxy, use the command:

```
>> edg-voms-proxy-init -print
```

Destruction

To destroy your VOMS proxy, use the command:

```
>> edg-voms-proxy-destroy
```

3 Grid Information

The aim of the Information and Monitoring Service is to deliver a flexible infrastructure that efficiently provides information about the state of grid services and about applications running on the grid. The information system forms the backbone of the grid and consequently is used heavily by other grid services to locate and select suitable resources.

For its information system, EDG uses R-GMA (Relational Grid Monitoring Architecture). R-GMA uses a relational model and HTTP Servlet technology to implement the Grid Monitoring Architecture from the Global Grid Forum. Information generated by R-GMA "Producers" are made available to to R-GMA "Consumers" as relations (tables).

Services communicate with the R-GMA servlets via an API which has been implemented in Java, C, C++, python and perl. The servlet responds with an XML document that corresponding to an XML schema definition. Related R-GMA documentation[1] can be found on the web.

1. Information and Monitoring Services Architecture: This presents the architecture, use cases, and requirements along with the design and evaluation criteria.
2. R-GMA Users Guide: This explains what you need to know as a user of the Relational Grid Monitoring Architecture (R-GMA) Information and Monitoring Services.
3. R-GMA Installation Guide: This explains what you need to know as an installer of the various parts of the R-GMA system. It covers all the components, though most sites will only need some parts configuring.
4. R-GMA Developers Guide: This explains how to get started as an R-GMA developer. It trys to cover everything from setting up a computer to do the development to generating a R-GMA release.

As a user, the main interaction with R-GMA will be through the command line or browser interfaces. A useful, annotated subset of the information system schema can be found in Appendix D.

A number of information providers have been produced by EDG, including site information, computing element, storage element and network monitoring scripts. On the various grid resources, a program called Gin invokes these scripts to obtain the state information of the resource. Gin then parses the output of the script and publishes this information by using R-GMA.

R-GMA queries use SQL. Queries can be executed with the **edg-rgma** command line tool available on user interface machines. Several interesting queries are:

```
edg-rgma -c "latest select UniqueID,RunningJobs,TotalJobs from GlueCE"
edg-rgma -c "latest select UniqueID,CurrentIOLoad from GlueSE"
edg-rgma -c "latest select URI,Type from Service"
```

which return information about the computing elements, storage elements, and services on the grid. These commands use a "latest" producer to process the given SQL query. Using the **edg-rgma** command with no arguments invokes an interactive command line interface. Within the interactive R-GMA shell, help is available.

The information in R-GMA can also be accessed via a browser. The browser interface runs on the Information Catalog (gppic06.gridpp.rl.ac.uk). Pointing a web browser to the R-GMA browser URL[2] allows one to view the information schema and perform interactive queries.

[1] http://hepunx.rl.ac.uk/edg/wp3/documentation/
[2] http://gppic06.gridpp.rl.ac.uk:8080/R-GMA/index.html

4 Job Submission

The job submission system functions as a large batch system with commands to submit a job, check its status, and to retrieve any output. The two main differences between a standard batch system (such as PBS, LSF etc.) and the EDG job submission service are that the job submission service adds a high-level layer permitting uniform access to the resources at different sites and matches the available resources to the requirements for the job automatically.

The summary below is a brief overview of the main features of the job submission system; Figure 4.1 gives a brief pictorial overview. For further information, see the Workload Management Documentation[1]. See Chapter 5 for a description of the job execution environment.

4.1 Job Submission Commands

The relevant commands are

```
edg-job-submit --vo <VO> <job.jdl>
edg-job-status <jobId>
edg-job-get-output <jobId>
edg-job-cancel <jobId>
```

for submitting a job, querying its status, retrieving the output, and cancelling a job, respectively.

For input, the submit command takes a job description file ⟨job.jdl⟩ (see below). The submission returns a job identifier, ⟨jobId⟩, of the form:

```
https://lxshare0403.cern.ch:9000/2ODSom6oFPlHOsLGvTQX4Q
```

which can then be used to determine the status of the job and eventually retrieve its output.

The other commands take job identifiers as input (⟨jobId⟩) and perform the corresponding action on the listed jobs. The **edg-job-get-output** retrieves the output from the job from the resource broker machine where it is cached; the output can only be retrieved once a job has reached the "Done" status.

4.2 Job Description File

The key to the job submission and resource matching process is the job description file. This file describes the necessary inputs, generated outputs, and resource requirements of a job using the Job Description Language (JDL).

A typical example of a job description file:

```
Executable    = "HelloScript.sh";
Arguments     = "hello 200";
StdOutput     = "std.out";
StdError      = "std.err";
InputSandbox  = {"HelloScript.sh"};
OutputSandbox = {"std.out","std.err"};
Requirements  = other.GlueCEInfoTotalCPUs > 4;
Rank          = other.GlueCEStateFreeCPUs;
```

[1] http://www.infn.it/workload-grid/documents.html

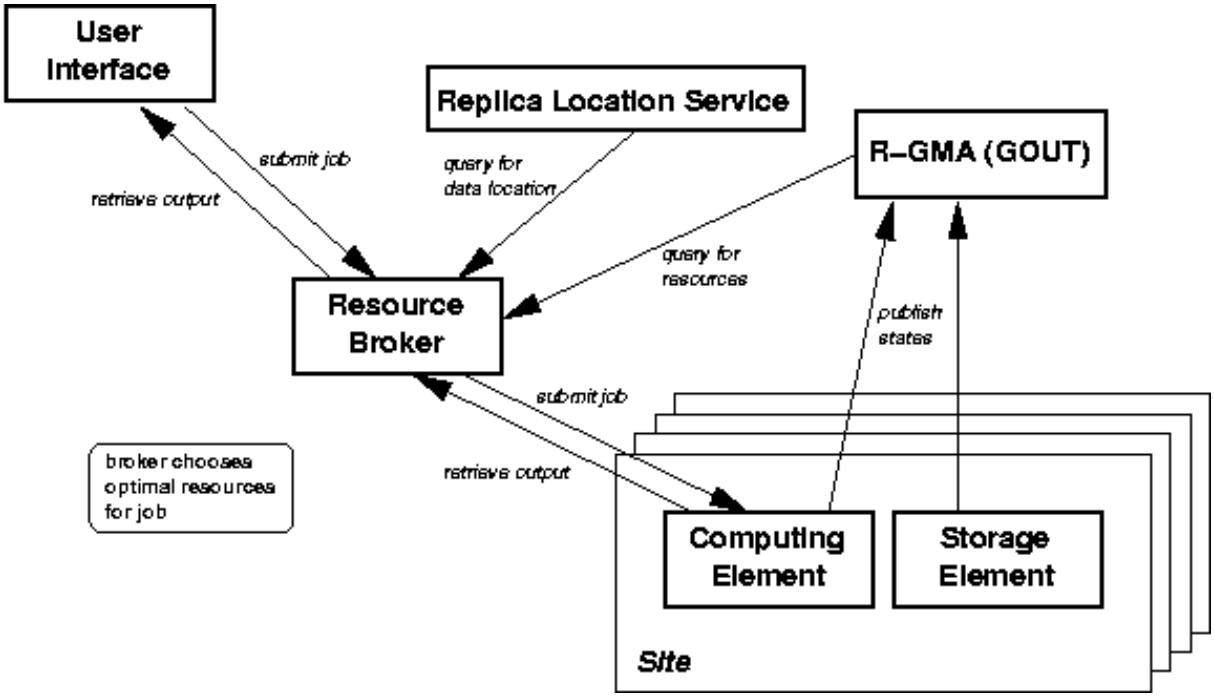

Figure 4.1: Job submission and execution. The user interacts with the grid resources via the User Interface by contacting a Resource Broker. When a user submits a job, the broker collects information for matchmaking, submits the job to the selected resource, and collects the results on completion. The broker caches the job output for later retrieval by the user.

shows that a script is passed as input to a job, which then produces standard output and error files which will be transported back to the user (eventually with a **edg-job-get-output**).

The input and output sandboxes are intended for relatively small files (on the order of a few megabytes) like scripts, standard input, and standard output streams. If you are using large input files or generating large output files, you should instead directly read from or write to a storage element. *Abuse of the input and output sandboxes can fill the storage on the ResourceBroker and make the broker unusable.*

The two parameters—Requirements and Rank—control the resource matching for the job. The expression given for the requirements specifies the constraints necessary for a job to run. In this case, a site with more than four CPUs is required. The job will only be submitted to resources which satisfy this condition. If more than one resource matches, then the rank is used to determine which is the most desirable resource and hence the one to which the job is submitted. (Higher values are more desirable.) In this case, the resource with the largest number of free CPUs is chosen. Both of these can be arbitrary expressions which use the fields published by the resources in the information system. The JDL uses the LDAP names of these attributes (which are slightly different than the SQL names used by R-GMA); see Appendix D for a list of useful attributes as well as the JDL Attributes documents[2].

JDL is based on the Condor ClassAds library. More information about the supported functions and the syntax of these expressions can be found in the ClassAds documentation[3].

ClassAds are extensible and place no restrictions on the parameter names. A side-effect of this is that misspelled parameter names are still syntatically valid, but will not act as expected. *Be extremely careful spelling the JDL parameter names.*

Using parameters in the JDL file one can steer jobs to sites which have a copy of a particular input file:

[2]http://server11.infn.it/workload-grid/documents.html
[3]http://www.cs.wisc.edu/condor/classad/

```
InputData = {"lfn:file1.txt","guid:135b7b23-4a6a-11d7-87e7-9d101f8c8b70"};
DataAccessProtocol = {"file", "gridftp"};
```

Both the parameters are required to use this feature. Either logical file names (LFN) or file identifiers (GUID) can be specified; see Chapter 6 for details. The file catalog to use is determined from the information system and the VO specified on the command line. NOTE: If you specify the wrong VO, the wrong catalog will be used leading (usually) to a job matching failure. Similarly, one can also select a site with a particular storage element:

```
OutputSE = "gppse05.gridpp.rl.ac.uk";
```

For more details, see the job submission examples (Section 4.5), further examples in the Data Management Section (Section 6), and the Workload Management Documentation[4].

It is often useful to check the results of the resource matching without submitting a job. For this, one can use the **edg-job-list-match** command. Given the JDL file it will return a ranked list of matching resources. The highest-ranked resource will appear first.

4.3 Long-lived Jobs

Long jobs may outlive the validity of the initial proxy; if so and the proxy is not renewed, the job will die prematurely. To avoid this the workload management software allows the proxy to be renewed automatically if your credentials are managed by a proxy server.

To use the automatic proxy renewal mechanism, first register a proxy with the MyProxy server using the command

```
myproxy-init -s <server> -t <hours>  -d -n
```

where "server" is the host name of the MyProxy server and "hours" is the number of hours the proxy should be valid on the server (default is 7 days). Specifying the *-d* and *-n* options are vital to proper proxy renewal! MyProxy servers can be found in the service table with R-GMA (see Chapter 3).

As this proxy is only copied to the server, you will need to create a local short-lived proxy using **grid-proxy-init** to do the job submissions. The resource broker will retrieve renewed proxies from the MyProxy server for jobs which need them.

Information about your stored proxy can be obtained via the command

```
myproxy-info -s <server> -d
```

and the proxy can be removed with

```
myproxy-destroy -s <server> -d.
```

Once the proxy is removed from the server, running jobs will no longer receive renewed credentials.

To allow the broker to find the correct MyProxy server, you must specify it in a job's JDL file:

```
MyProxyServer = "myproxy.example.org";
```

and the MyProxy server must be configured to allow renewals from the broker you use. If this isn't specified in the JDL file, the proxy will not be renewed. A default can be specified in the UI configuration file, if desired.

[4] http://www.infn.it/workload-grid/documents.html

4.4 Interactive, MPI, and Checkpointed Jobs

With the latest versions of the workload management software, it is now possible to submit interactive, MPI (Message Passing Interface), and checkpointed jobs. Interactive jobs open real time connections to the standard input, output, and error streams of the job and allow direct interaction with it. See Example 4.5 for a typical interactive job example. MPI and checkpointed jobs require specially-instrumented user executables linked against the MPI or checkpointing libraries, respectively.

Parallelized computations are typically instrumented with the MPI and are designed to run on many CPUs simulataneously. To submit an MPI job with the EDG software, one must specify two parameters in the job's JDL file:

```
JobType = "MpiCh";
NodeNumber = 4;
```

The first parameter identifies this job as an MPI executable; the second parameter specifies the number of nodes needed for the executable to run.

The workload management system implements logical checkpointing for jobs. The checkpointing system handles the details for saving and retrieving a checkpoint state. However, the job itself is responsible for the content of the checkpoint state and it must be written to allow restarting from a specified checkpoint state.

Jobs linked against the checkpointing library periodically save the state of the computation; these jobs can be restarted from any of the intermediate saved states. To specify a job which uses an executable instrumented with the checkpointing library, simply specify the type of the job in the JDL file:

```
JobType = "checkpointable";
```

The checkpointing framework allows you to retrieve the intermediate state of a job and to specify the state when submitting a job. For example, the commands

```
edg-job-get-chkpt --cs 1 -o <state.file> <jobid>
edg-job-submit -chkpt <state.file> <jdl file>
```

will retrieve the penultimate state of the job and then submit a new job which starts from that state.

4.5 Examples

Example 4.1 (Hello World) The simplest job is one which echos "Hello World" to the standard output. To run this example prepare a file `hello.jdl` which contains the following:

```
Executable    = "/bin/echo";
Arguments     = "Hello World";
StdOutput     = "std.out";
StdError      = "std.err";
OutputSandbox = {"std.out","std.err"};
```

Note that the complete path of the command is given and that the standard output and standard error are specified in the output sandbox.

Submitting the job returns the generated job identifier:

```
>> edg-job-submit -nomsg Hello.jdl

https://boszwijn.nikhef.nl:9000/F0qoyQxCejiW5GLeaYc_Ag
```

where the *-nomsg* option suppresses superfluous information.

Using **edg-job-status** with the job identifier above yields the following, edited output:

```
>> edg-job-status https://boszwijn.nikhef.nl:9000/F0qoyQxCejiW5GLeaYc_Ag

Printing status info for the Job : https://boszwijn.nikhef.nl:9000/F0qoyQxCejiW5GLeaYc_Ag
Current Status:    Running
Status Reason:     Job successfully submitted to Globus
Destination:       grid-w2.ifae.es:2119/jobmanager-pbs-workq
reached on:        Wed Oct  1 13:00:40 2003
```

When the status was requested, the job was running. States seen in the normal processing of jobs are: Accepted, Waiting, Running, and Done. Abnormal execution usually ends with an "Aborted" status.

To retrieve the output (once the job has reached the "Done (Success)" state), use the **edg-job-get-output** command with the job identifier as the argument. The following is returned:

```
>> edg-job-get-output https://boszwijn.nikhef.nl:9000/F0qoyQxCejiW5GLeaYc_Ag

 Output sandbox files for the job:
 - https://boszwijn.nikhef.nl:9000/F0qoyQxCejiW5GLeaYc_Ag
 have been successfully retrieved and stored in the directory:
 /tmp/jobOutput/F0qoyQxCejiW5GLeaYc_Ag
```

where the output has again been edited. The important information is the location of the output files. Within the given directory will be the `std.out` and `std.err` files specified in the output sandbox. The `std.out` file should contain the string "Hello World". The `std.err` file is empty in this case but would contain anything written to the standard error.

The contents of the JDL file are transformed by many different programs during the submission process. A side effect of this is that special characters in the "Arguments" attribute must be preceeded by triple '\', e.g.:

```
Executable = "/usr/bin/tail";
Arguments = "-f file1\\\&file2";
```

and quotes must be escaped with the '\' character, e.g.:

```
Executable = "/bin/grep";
Arguments = "-i \"my name\" *.txt";
```

or better, create a script as in the next example.

Directly handling the job identifiers quickly becomes tedious. To avoid this, the **edg-job-submit** will append the job identifier to a named file when using the *-o* option. Job management commands which take job identifiers as an argument accept the *-i* which allows the job identifier to be read from a file.

Note that the job management system does not limit the rate of jobs submitted either by a single user or in total. Extremely high rates have not been tested with this release, but users are asked to show some restraint in this regard.

Example 4.2 (Hello from Script) The next example simply sends a small script with the job, executes it and returns the results. Create an executable file called `HelloScript.sh` which contains the following:

```
#!/bin/sh
/bin/echo "Hello From Script"
/bin/echo "Error From Script" 1>&2
```

This will echo "Hello From Script" to the standard output and "Error From Script" to the standard error.

The appropriate JDL file for this job is the following.

```
Executable    = "HelloScript.sh";
StdOutput     = "std.out";
StdError      = "std.err";
InputSandbox  = {"HelloScript.sh"};
OutputSandbox = {"std.out","std.err"};
```

If the script is not in the current directory, then you must give the full path in the input sandbox line.

After submitting the job as in the previous example and retrieving the output, one finds that the `std.out` and `std.err` files contain the strings "Hello From Script" and "Error From Script", respectively.

The executable named in the JDL will automatically be set with executable permissions. Other executable files shipped in the input sandbox may need to have the permissions set explicitly.

Important note: The input and output sandboxes are intended for relatively small files (few megabytes) like scripts, standard input, and standard output streams. If you are using large input files or generating large output files, you should instead directly read from or write to a storage element. *Abuse of the input and output sandboxes can fill the storage on the ResourceBroker and make the broker unusable.*

Example 4.3 (Specifying Job Requirements) By specifying job requirements, the user can steer the job to sites which have the resources necessary to run the job correctly. Incompletely specifying the requirements may cause the job to be scheduled on a resource which cannot handle the job. This will cause the job to fail, wasting computing resources and the user's time.

The requirements are specified with a "Requirements" attribute in the JDL description of the job. This value of this attribute is a boolean expression which specifies the necessary constraints. Nearly the full set of C operators and syntax are supported.

The values (or variables) which can be used in the requirements expression can be found by looking at the GlueCE attributes in the information system (see Appendix D). To see the values of attributes for all CEs, try the following command:

```
> edg-rgma -c "latest select * from GlueCE"
```

NOTE: the names vary between the LDAP and SQL versions of the information schema. The ones specified in JDL expressions must be LDAP names (see Appendix D).

To express that a job requires at least 25 minutes of CPU time and 100 minutes of real time, the expression:

```
Requirements = other.GlueCEPolicyMaxCPUTime>=1500 &&
               other.GlueCEPolicyMaxWallClockTime>=6000;
```

would limit the matching to viable sites. The times are given in seconds. Note that the attribute names are prefixed with "other."; this is a remnant of the ClassAds syntax on which JDL is based. Note also that the values are *not* quoted. Using quotes around a numeric value will result in a string comparision which will produce an erroneous match (or none at all).

The "GlueHostApplicationSoftwareRunTimeEnvironment" is usually used to describe application software packages which are installed on a site. For example,

```
Requirements = Member("ALICE-3.07.01",
                      other.GlueHostApplicationSoftwareRunTimeEnvironment);
```

will choose a site with the "ALICE-3.07.01" tag defined. The run time environment is a multi-valued attribute and evaluates to a list. The "Member" function returns true if the given value is in the list. (NOTE: The order of these parameters have changed from earlier releases.)

Occasionally, one may wish to exclude or include a site manually. Forcing a job to a site can be accomplished with the *-resource* option of the **edg-job-submit** command. However, this entirely bypasses the matchmaking process and will not produce a `.BrokerInfo` file (see the Workload Management Documentation[5] for information on the BrokerInfo file) with the matchmaking results. Instead one can use the matchmaking with a clause like[6]

```
Requirements = other.GlueCEUniqueID ==
                "adc0006.cern.ch:2119/jobmanager-pbs-short";
```

to do the same thing. More interestingly one can select or exclude a site:

```
Requirements = RegExp(".*cern.*",other.GlueCEUniqueID);
Requirements = (!(RegExp(".*cern.*",other.GlueCEUniqueID)));
```

which cannot be accomplished with the *-resource* option. Note that the JDL is very picky about the logical "not" syntax. Many sites also define a run time environment variable which identifies the site.

In the UI configuration file (`/opt/edg/etc/edg_wl_ui_cmd_var.conf`) there is a requirements clause which is added to all JDL files by default. By default this is

```
Requirements = other.GlueCEStateStatus == "Production";
```

which chooses sites marked as production. Users may create a UI configuration file of their own to specify habitual requirements (or to choose an alternate resource broker, etc.). To use a custom UI configuration file set the EDG_WL_UI_CONFIG_PATH variable to the full path name, or specify the *-c* option when submitting a job with the **edg-job-submit** command. This configuration also contains a default Virtual Organization to use if one is not specified on the command line.

Example 4.4 (Ranking Resources) If more than one resource matches the specified requirements, then the highest-ranked resource will be used. If the "Rank" attribute is not specified in the user's JDL description, then

```
rank = - other.GlueCEStateEstimatedResponseTime;
```

will be used by default (also specified in the UI configuration file). The estimated response time is the expected time in seconds that a job will take to begin executing at the site.

This ranking is not always ideal, and the user may wish to choose some other criteria for the ranking. The rule to remember is that the larger the rank, the more desirable the resource is. If there are multiple resources with the same, highest rank, the broker will choose randomly between those resources. For example,

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

will choose the site with the largest number of free CPUs. If there are no suitable sites with free CPUs, the broker will choose randomly. An expression like:

[5] http://www.infn.it/workload-grid/documents.html

[6] A "CE" in this context is a batch queue. A "CEId" is a concatenation of the host, port, type of batch system, and queue name.

```
Rank = 1;
```

will ensure a random selection between all suitable resources.

Example 4.5 (Interactive Jobs) Save the following JDL to a file called "Interactive.jdl",

```
JobType       = "Interactive";
Executable    = "ScriptInt.sh";
StdOutput     = "std.out";
StdError      = "std.err";
InputSandbox  = {"ScriptInt.sh"};
OutputSandbox = {"std.out","std.err"};
ListenerPort  = 50101;
Requirements  = other.GlueCEStateFreeCPUs > 0;
Rank          = other.GlueCEStateFreeCPUs;
```

and the following to a script called "ScriptInt.sh"

```
#!/bin/sh
echo "Welcome!"
sleep 1;
echo "What is your name?"
read A
echo "Bye Bye $A"
exit 0
```

When the job starts, an X-window will be opened allowing you to see the standard output and error streams; moreover, you'll be able to provide the standard input.

Note that you must connect to the User Interface machine in such a way that X-Window input is possible. Usually this means using **ssh** with the *-X* option.

Also the RB must be able to make an incoming connection to a TCP port on the UI. If your UI is behind a firewall, then specify the listener port in the JDL with the following:

```
ListenerPort  = 50100;
```

where you choose a free port which is accessible from outside the firewall. The environmental variable GLOBUS_TCP_PORT_RANGE, if set, contains a range of ports accessible from the exterior.

5 Job Environment

Currently the environment seen by jobs on grid resources contains very little "grid" customization. The standard PATH and LD_LIBRARY_PATH variables will be set allowing access to typical grid client commands. In addition, the variables EDG_LOCATION and GLOBUS_LOCATION will be set to the top of the EDG and Globus software trees.[1]

Notably there will be no environment specific to your virtual organization setup automatically. To bootstrap the environment setup, your job should first execute the command:

```
eval `\$EDG_LOCATION/bin/edg-vo-env --shell=sh atlas`
```

where "atlas" is replaced by the name of your virtual organization and "sh" is either "sh" or "csh" for sh-like or csh-like shells, respectively. The command uses "sh" if the shell is not specified explicitly. After execution the variable ATLAS_ROOT_DIR will be defined. Your virtual organization may also require additional setup in which case you will be required to source a script under the ATLAS_ROOT_DIR area.

Information about the matchmaking process is provided in the `.BrokerInfo` file which is accessible from the environment variable EDG_WL_RB_BROKERINFO. A precise example on using this file can be found in Chapter 6.

[1] The environment described here is only initialized for login shells. A login shell is started automatically when submitting jobs via EDG commands but not for the raw Globus commands. If you use the raw Globus commands, you must specify a login shell to get this environment.

6 Data Management

A large part of user tasks on the grid consist of access to data and management of the files containing data. Most users will use the Replica Manager command line interface and API to perform data management tasks on the grid. The Replica Manager interacts with the Replica Location Service (RLS), the Replica Metadata Catalog (RMC), the Replica Optimization Service (ROS) and the Storage Elements (SE) to provide high-level functionality and concurrently to shield users from tedious details of direct RLS and SE interaction. Nonetheless some details concerning the RLS and SE help users understand how the Replica Manager performs its job.

6.1 Terminology

Jargon unfortunately permeates the descriptions of the data management middleware. The following definitions will help to understand the typical terminology:

GUID Grid Unique Identifier. This is a unique, immutable label for a file registered in the RLS. All replicas of this file share the same GUID. GUIDs take the form:

```
guid:135b7b23-4a6a-11d7-87e7-9d101f8c8b70
```

LFN Logical File Name. This is a user-specified, unique label for a file–usually a more intuitive tag which gives some indication of the file's content. In contrast to a GUID, a LFN is mutable. LFNs take the form:

```
lfn:HiggsMonteCarlo.dat.
```

SURL Storage URL. A URL which uniquely identifies a file contained in a Storage Element. SURLs typically take the form:

```
srm://grid02.lal.in2p3.fr/iteam/higgsCandidate.dat
```

TURL Transport URL. A temporary URL which can used used to access a particular data file contained in a Storage Element via a certain protocol. For example, a TURL for access to a file via rfio takes the form:

```
rfio://grid02.lal.in2p3.fr/iteam/higgsCandidate.dat
```

Much of the terminology has changed from the previous release and has been replaced by the above, more precise terms.

6.2 Replica Location Service

The Replica Location Service (RLS) consists of two services: the Local Replica Catalog (LRC) and the Replica Location Index (RLI). The RLI allows the RLS to be geographically distributed; however, for EDG 2.0, this is not deployed. Therefore, a single LRC instance per Virtual Organization acts as a global registry for that VO's files. (Technically, the LRC contains the GUID→SURL mapping as well as some metadata concerning the physical file. See below for an explanation of the terminology.)

A service closely related to the RLS is the Replica Metadata Catalog (RMC). The RMC contains metadata about a VO's files. (Technically, the LFN→GUID mapping and metadata tied to the GUID.)

The Replica Optimization Service (ROS) allows the Replica Manager to choose the "closest" file in terms of total transfer time.

Finally, the Storage Element (SE) provides a uniform interface to data storage. It provides a web service interface for management functions and typically allows for several types of direct access to data stored on the SE. The GridFTP protocol is supported by all SEs and can be used for wide-area access to the data. Typically "file" (i.e. standard POSIX access) and "rfio" access to the data are also provided to a "close" Storage Element. A SE and CE are in fact defined to be "close" if file access to the SE's data is possible from the CE.

6.3 Replica Manager

The Replica Manager allows one to copy files into grid storage, register files, replicate files between SEs, delete individual replicas, delete all replicas of a particular file, among other things. All of these are available via the **edg-replica-manager** command or its abbreviated version **edg-rm**. Two general options to this command that are absolutely vital to correct operation are the *--vo* and *--insecure*[1] options. The *--vo* option takes the name of your VO as an argument.

A good first test is to execute the following on an User Interface machine:

```
edg-replica-manager --vo iteam --insecure printInfo
```

substituting "iteam" for the name of your VO. This prints a lot of information about exactly what services the replica manager command will use; the information is pulled from R-GMA. If there are problems with the Replica Manager commands, this command is often useful for debugging.

The subcommands for the Replica Manager also have shortened forms; for example the "printInfo" in the above command could have been replaced with "pi". A full list of the abbreviations is available from the command's usage obtained with the *--help* option. The examples in this chapter will use the long forms for clarity.

Frequently used subcommands are:

- **copyAndRegisterFile**: useful for bringing a file on to the grid
- **listReplicas**: list all of the replicas of a given file
- **deleteFile**: delete a replica and remove from catalog
- **replicateFile**: create a new replica of a file which already exists on the grid on a particular SE
- **listGUID**: list the GUID of a LFN or SURL
- **getBestFile**: return the SURL of the "closest" replica of a file making a local replica if possible

The use of all of these commands will be seen in the following examples.

[1]The current testbed deployment does not use the security features of RLS. If this option is not specified, then the replica manager will attempt to use the secure port and the command will fail.

6.4 Examples

The examples show typical data management cases and highlight the commands described above.

Example 6.1 (Bringing a File onto the Grid) Often data files are first created in temporary scratch space or on computers outside of the grid. To make these data grid-accessible, they must be moved to a Storage Element; usually one wants to register these files in a VO's catalog as well. This example demonstrates how to do this.

First create a fresh proxy with **grid-proxy-init**. Although the registration is not currently secured, the file transfer is; therefore, valid credentials will be needed.

Create an empty local file to work with:

```
touch empty-local-file
```

and now perform a **copyAndRegisterFile** with the Replica Manager to copy this to a Storage Element and register the file.

```
>> edg-replica-manager --vo iteam --insecure \
     copyAndRegisterFile file:‘pwd‘/empty-local-file \
     --destination-file gppse05.gridpp.rl.ac.uk \
     --logical-file-name lfn:my-demo-2003-10-01-1600

guid:b793f080-f417-11d7-b584-857330072702
```

The GUID of the created file is returned on successful completion. If the *--destination-file* option is not given, then the copy is made on the "local" SE. You can use the **printInfo** subcommand to see what the "local" SE is. If the *--logical-file-name* is not given, then the only way to access this file is through the returned GUID.

To check that this file exists, use the **listReplicas** command:

```
>> edg-replica-manager --vo iteam --insecure \
     listReplicas guid:b793f080-f417-11d7-b584-857330072702

srm://gppse05.gridpp.rl.ac.uk/iteam/generated/2003/10/01/fileb16684bf...
```

Either the logical file name or GUID can be used. One sees that both commands return the same SURL (truncated here) for the replica and that this replica is indeed on the specified SE.

To delete this file, one can use the subcommand **deleteFile**, specifying the SURL to be deleted.

```
>> edg-replica-manager --vo iteam --insecure \
     deleteFile \
     srm://gppse05.gridpp.rl.ac.uk/iteam/generated/2003/10/01/fileb16684bf...
```

Using the *--all* option currently will delete the GUID and LFN from the catalog as well. The behavior will disappear in the future releases of the replica manager, so do not rely on it. Use the command

```
edg-rm --vo=iteam --insecure \
  removeAlias \
  guid:b793f080-f417-11d7-b584-857330072702 \
  lfn:my-demo-2003-10-01-1600
```

to delete a logical file name. You need both the GUID and LFN to do this.

Example 6.2 (Replicating Existing File) As the brokering system does not yet perform automatic replication of data files for jobs, it is often necessary to make several replicas of a file on different Storage Elements. To demonstrate this, repeat the previous example to bring a file "lfn:my-second-demo-2003-10-01-1600" onto the grid but fill the file with the string "Hello There". To verify this exists:

```
>> edg-replica-manager --vo iteam --insecure \
     listGUID lfn:my-second-demo-2003-10-01-1600

guid:a3ac7647-f418-11d7-a57b-e4d5c9608efc
```

which lists the GUID associated with this file. One could have also used **listReplicas** again

```
>> edg-replica-manager --vo iteam --insecure \
     listReplicas lfn:my-second-demo-2003-10-01-1600

srm://gppse05.gridpp.rl.ac.uk/iteam/generated/2003/10/01/file9de8efe6...
```

which shows that the file is on the gppse05.gridpp.rl.ac.uk SE.

You can use the **edg-rgma** to find another SE. Now to replicate this to another storage element:

```
>> edg-replica-manager --vo iteam --insecure \
     replicateFile --destination se001.fzk.de \
     lfn:my-second-demo-2003-10-01-1600

srm://se001.fzk.de/iteam/generated/2003/10/01/file42a1d2b2...
```

which returns the SURL of the copy. Using **listReplicas** again shows the two distinct replicas:

```
>> edg-replica-manager --vo iteam --insecure \
     listReplicas lfn:my-second-demo-2003-10-01-1600

srm://gppse05.gridpp.rl.ac.uk/iteam/generated/2003/10/01/file9de8efe6...
srm://se001.fzk.de/iteam/generated/2003/10/01/file42a1d2b2...
```

Leave these files on the grid for the next example.

Example 6.3 (Accessing a File from a Job) The previous example showed how to bring data onto the grid and move it around. This one now demonstrates how to read the data from a job using the "file" protocol. It uses **getBestFile** to get the SURL of the local copy (making a copy if necessary) and then transforms that SURL into a filename which can be opened. The script calculates the checksum of the file.

Put the following JDL into a file called "ReadData.jdl":

```
Executable         = "script.sh";
InputData          = {"lfn:my-second-demo-2003-10-01-1600"};
DataAccessProtocol = {"file","gridftp","rfio"};
StdOutput          = "std.out";
StdError           = "std.err";
InputSandbox       = {"script.sh"};
OutputSandbox      = {"std.out","std.err"};
```

and put the following into a file `script.sh`:

```
#!/bin/sh

# Get SURL of local replica (making one if necessary).
surl=‘edg-replica-manager --vo iteam --insecure \
        getBestFile lfn:my-second-demo-2003-10-01-1600‘
echo SURL: $surl

# Get TURL of the local replica.
turl=‘edg-replica-manager --vo iteam --insecure \
        getTurl $surl file‘
echo TURL: $turl

# Strip off URL’s scheme and fix multiple slashes.
fname=‘echo $turl | sed -r ’s%/+%/%g’ | sed s%file:%%‘
echo FILE: $fname

# Get the check sum of this file.
cksum $fname
```

Checking the matching with an **edg-job-list-match** should return Computing Elements at the two sites which have replicas of this file. Actually sending the job should return the correct checksum of the file in the `std.out` file.

More information on the Replica Manager and the underlying services discussed in this chapter can be found in the Users' Guides for the Replica Manager[2], LRC[3], RMC[4], and ROS[5].

[2] http://cern.ch/edg-wp2/replication/docu/edg-replica-manager-userguide.pdf
[3] http://cern.ch/edg-wp2/replication/docu/edg-lrc-userguide.pdf
[4] http://cern.ch/edg-wp2/replication/docu/edg-rmc-userguide.pdf
[5] http://cern.ch/edg-wp2/replication/docu/edg-ros-userguide.pdf

7 Storage Element

The Storage Element (SE) acts as a Grid interface to mass storage systems (MSS), or to disk. The SE can be used to make existing files in an MSS available to the Grid, or to write files from the Grid and store these in MSS. The SE itself does not store the files, it only keeps cached copies of the files on its own disk (except for disk only SEs of course), plus some metadata for the files such as access control lists etc.

So when you need to access a file from the Grid, you ask the SE to make a copy of the file available on its disk cache. Conversely, when writing a file into the SE, you need to ask the SE for a location in the disk cache to which you can upload the file. In either case you also need to tell the SE when you have finished with the file so the SE can reclaim the space in its disk cache. This step is necessary because a client may make several independent accesses to any given file.

Normally Grid clients don't access an SE directly, but leave it to the Replica management system (see Chapter 6) to manage the files for them. Nevertheless there are cases where a client will wish to access an SE directly, for example when writing output from a job to a specific SE shared for a cluster, or creating temporary files, or other files that need not be replicated.

7.1 TURLs

7.1.1 What is a TURL?

A TURL is a Transfer URL. In the Storage Resource Manager (SRM) model, you tell the SE/SRM that you wish to access an existing file and the SRM provides a URL where you can fetch it. You must then tell the SRM when you have finished accessing the file, so the disk cache space can be reclaimed.

Creating a file is entirely analogous: first you tell the SE/SRM that you wish to create a file with a given size and a given name, and then the SE/SRM sends you a TURL to which you can upload the file. Finally you must tell the SE/SRM that you have finished uploading the file.

7.1.2 An example using the SE command line interface

In this example we create and upload a file to the SE using the WP2 command line interface to the SE web service.

```
>> edg-se-webservice \
  -i create gppse01.gridpp.rl.ac.uk/nikuufop \
  --endpoint http://gppse01.gridpp.rl.ac.uk:8080/edg-se-webservice/services/edg-se-webservice

gsiftp://gppse01.gridpp.rl.ac.uk//flatfiles/01/data/d0/d0738b9c936f8790a3cd58ea60625642
```

This returns a request id marked as "<request id>" below. (For SEs prior to the 2.1 release, also happens to be a gsiftp turl.) Then we use the **getTURL** subcommand, specifying the desired protocol:

```
>> edg-se-webservice -i getTURL \
                     --endpoint <...> \
                     <request id>

gsiftp://gppse01.gridpp.rl.ac.uk//flatfiles/01/data/d0/d0738b9c936f8790a3cd58ea60625642
```

This returns the turl for the protocol ("<TURL>" below). Then we upload the file to the turl:

```
>> globus-url-copy file:///‘pwd‘/testfile <TURL>
```

Finally we call commit to tell the SE that we have finished uploading the file, passing again the request id to the SE.

```
>> edg-se-webservice -i commit \
                     --endpoint <...> \
                     <request id>
```

The SE interface to the SE (as opposed to the SRM interface to the SE) can be confusing because prior to release TB2.1 it will return a gsiftp TURL as a request id. This is confusing because people then forget to call **getTURL** subcommand (which is OK if you only wanted a gsiftp turl anyway, because in that case **getTURL** does nothing).

7.1.3 Simplified SRM Example

We give an example of using the SRM version 1 API to download the file we just uploaded above. The example is simplified in the following ways: we consider only requests on a single file and we simplify the response by only considering the most essential parts of the response.

The example is to fetch and download a file from the SRM using the SRM v1 API.

The client calls

```
get( "srm://gppse01.gridpp.rl.ac.uk/nikuufop", "gsiftp" )
```

This command returns a structure which contains

Request ID e.g. 16726, for SRM v1 request ids are always integers

File Index An index for the file in the request. This is so each file in the request can be addressed individually. Say, 0 in this case.

State Typically "Pending", "Ready", or "Failed" at this stage.

TURL A transfer URL which is valid only when the state is **Ready**.

If the state above is not "Ready", the client poll for a "Ready" status with

```
getRequestStatus(16726)
```

The number 16726 is the request id returned by the **get** method. This command returns the same structures as the **get** command, but perhaps with an updated "state" entry.

Once the state becomes "Ready", the TURL entry is valid:

```
gsiftp://gppse01.gridpp.rl.ac.uk//flatfiles/01/data/d0/d0738b9c936f8790a3cd58ea60625642
```

Then you fetch the file using, e.g., **globus-url-copy**. Finally, when you're done you must inform the SRM that you have finished with the file.

```
setFileStatus( 16726, 0, "Done" )
```

The 0 here is the index of the file in the request, as returned by the intial **get** method and by **getRequestStatus**.

7.2 Special TURLs

The TURL is *always* a URL. This is because the TURL is parsed by replica manager software and this software expects well-formed URLs.

However, sometimes files are not accessed using URLs, and the two canonical examples of this are RFIO and POSIX access (e.g. an NFS mounted SE).

7.2.1 RFIO

For RFIO, the returned TURL may look like this:

```
rfio://adc0027.cern.ch///flatfiles/SE02/data/5a/5a155b26b642157b7cef1f407206f825
```

However, the RFIO API expects an RFIO name of the form

```
adc0027.cern.ch:/flatfiles/SE02/data/5a/5a155b26b642157b7cef1f407206f825
```

This means that the client has to strip away the "rfio://" part, and insert a colon at the end of the hostname. The extra slashes don't matter.

NOTE: The RFIO port number used for to access CASTOR at CERN is 5001, but the IANA defined standard is 3147. The SEs at CERN generally use 3147. You may need to set the RFIO_PORT environment variable to the value 3147 before using RFIO clients to access the SE.

7.2.2 POSIX

If you request a file TURL with **getTURL**, it will return something like the following:

```
file:////flatfiles/SE02/data/5a/5a155b26b642157b7cef1f407206f825
```

In this case the hostname is already stripped out by the SE. Java can open a TURL of this format. The standard C library POSIX cannot cope with it, and it is necessary for the client to strip away the initial "file://":

```
FILE *
turl_open( char const *turl, char const *access )
{
        char const proto[] = "file://";
        if( strncmp( proto, turl, strlen(proto) ) ) {
                errno = EFAULT;
                return (FILE *)NULL;
        }
        turl += strlen( proto );
        return fopen( turl, access );
}
```

This example assumes that the SE is mounted on a directory that makes the file path identical on the SE and on the client–which is generally the case. Otherwise the client will have to translate the path as well (or the SE will have to be configured to do this, but this will only work if the translation is the same for all clients).

8 Metadata Management

Some EDG services must maintain persistent metadata in remote relational databases. However, existing relational database systems are not grid-enabled, affecting adversely cross-organizational interoperability and reuse. Spitfire[1] addresses these issues. Spitfire is designed to be used for metadata storage and retrieval.

The Spitfire middleware is inserted into the control and data path between the client and the RDBMS and so grid-enables the RDBMS. It introduces a uniform interface, data model, network protocol and security model. These are based on widely accepted standards and neutral with respect to programming language, platform and database product.

There are three main components to the Spitfire service: the primary server component and the client library component(s). Applications that have been linked to the Spitfire client library communicate to a remote instance of the server. This server is put in front of a RDBMS (e.g. MySQL or Oracle), and securely mediates all Grid access to that database. The browser is a standalone web portal that can also be placed in front of a RDBMS.

The server is a fully compliant Web Service implemented in Java. It runs on Apache Axis inside a Java servlet engine (currently Apache Tomcat). The service securely mediates access to the RDBMS. The service is reasonably non-intrusive, and can be installed in front of a pre-existing RDBMS. The local database administrator retains full control of the database back-end, with only limited administration rights being exposed to properly authorized grid users.

The web services client library, at its most basic, consists of a WSDL service description that describes fully the API. This API allows SQL operations to be performed from the client application upon the remote Spitfire service, with full Grid security. Using this standard WSDL description, client stubs can be generated automatically in the programming language of choice. We provide pre-built client libraries for the Java, C, and C++ programming languages.

In addition to accessing the service with the clients, the user can access the database using a web browser, too. The server package contains a web interface for querying and manipulating the database, as well as an administrative interface for defining access rules for users and groups.

More information can be found on the Spitfire website[2].

[1] http://edg-wp2.web.cern.ch/edg-wp2/spitfire/index.html
[2] http://edg-wp2.web.cern.ch/edg-wp2/spitfire/documentation.html

9 Application Monitoring with GRM/PROVE

GRM and PROVE are application monitoring and visualization tools for analysing the performance of message-passing parallel applications in the grid. The instrumentation library of GRM provides a flexible trace event specification.

What you need to do is

- to instrument your parallel application with GRM calls
- to submit your job
- to start PROVE and (from PROVE) GRM giving the job ID as parameter
- to watch PROVE drawing the behaviour of your program in a graphical time-line display.

For more information on how to use GRM see GRM–Grid Application Monitor Users Manual[1].

The components of GRM are connected to R-GMA (see chapter 3) to find the application in the grid. Besides R-GMA, GRM uses the Mercury monitor to transfer large amount of trace data through the network to the PROVE visualisation tool. Mercury monitor is an external package and it is installed only for this purpose.

[1] http://hepunx.rl.ac.uk/edg/wp3/documentation/grm/guide/index.html

10 Support

10.1 Website

The main Testbed website[1] contains documentation, contact information, the bug-reporting system, links to the source and packages repositories as well as links to other sites. This serves as the single point-of-access to information about the testbed activities.

10.2 Bugzilla

The bug-tracking system, Bugzilla[2], is intended to facilitate the reporting and fixing of bugs in the European DataGrid software. This includes the DataGrid's distribution of the Globus2 system; confirmed bugs in Globus will be forwarded to the Globus team.

This system is not intended to track bugs in application software, that is, user and experimental software running on the grid. For these types of problems, please refer to the list of application support contacts found on the contacts page[3].

The Bugzilla database is publicly available and can be searched by anyone. However reporting bugs requires a valid Bugzilla account. Creating an account requires only a valid e-mail address. You will be prompted to open an account when you report your first bug. Note that Bugzilla uses cookies to keep track of your account data, so your browser must have cookie support enabled.

Concise bug reports speed the resolution of the problem. Stripped-down test cases and detailed explanations are greatly appreciated. Please do search the existing bugs to see if your problem has already been reported.

10.3 Contacts

The user's first point of contact for operational problems is the local site administrator. A list of email addresses for the site administrators can be obtained with the following command:

```
edg-rgma -c "latest select siteName,userSupportContact from SiteInfo"
```

For application-specific problems the appropriate application representative should be contacted. Users are welcome and encouraged to use the bug-reporting facility. The user support group[4] can be contacted for help. As a last resort, users may contact the Integration Team[5].

[1] http://marianne.in2p3.fr/
[2] http://marianne.in2p3.fr/datagrid/bugzilla/
[3] http://marianne.in2p3.fr/datagrid/mailing-lists.html
[4] http://marianne.in2p3.fr/datagrid/support/
[5] mailto:hep-proj-grid-integration-team@cern.ch

A Glossary

AFS Andrew File System (http://www.openafs.org/frameless/main.html)

API Application Programming Interface

Bugzilla Open source bug-tracking software (http://bugzilla.mozilla.org/)

CA Certificate Authority

CASTOR CERN Advanced STORage Manager (http://castor.web.cern.ch/castor/)

CE Computing Element

CERN European Laboratory for Particle Physics (http://www.cern.ch/)

CPU Central Processing Unit

EDG European DataGrid (http://www.edg.org/)

GGF Global Grid Forum (http://www.gridforum.org/)

Gin EDG program to publish resource state information to R-GMA

GK Gatekeeper

GLUE Grid Laboratory Universal Environment (http://www.hicb.org/glue/glue.htm)

GridFTP FTP protocol with GSI security (http://www.globus.org/datagrid/gridftp.html)

GRM EDG application monitoring package

GSI Globus Security Infrastructure (http://www.globus.org/security/)

GUID Grid Unique IDentifier

HTTP Hyper Text Transfer Protocol

IANA Internet Assigned Numbers Authority (http://www.iana.org/)

IC Information Catalog

IO Input/Output

JDL Job Description Language

LDAP Lightweight Directory Access Protocol (http://www.openldap.org/)

LFN Logical File Name

LRC Local Replica Catalog

LSF Load Sharing Facility (http://www.platform.com/products/LSF/)

MON Monitoring node type

MPI Message Passing Interface (http://www-unix.mcs.anl.gov/mpi/)

MSS Mass Storage System

P12 Format for certificates combining public and private keys

PBS Portable Batch System (http://www.openpbs.org/)

PEM Format for certificates allows separated public and private keys

PKCS12 Public-Key Cryptography Standards (version 12)

POSIX Portable Operating System Interface for Unix (http://www.opengroup.org/products/publications/catalog/un.htm version 3)

PROVE Visualization program for GRM

RAL Rutherford Appleton Laboratory (http://www.rl.ac.uk/)

RB Resource Broker

RDBMS Relational Database Management System

RFIO Remote File IO

R-GMA Relational Grid Monitoring Architecture

RLI Replica Location Index

RLS Replica Location Service

RMC Replica Metadata Catalog

RM Replica Manager

ROS Replica Optimization Service

SE Storage Element

SQL Structured Query Language

SRM Storage Resource Model

SSL Secure Sockets Layer

SURL Storage URL

TCP Transmission Control Protocol

TURL Transport URL

UDP User Datagram Protocol

UI User Interface

URI Uniform Resource Identifier

URL Uniform Resource Locator

VOMS Virtual Organization Membership Service

VO Virtual Organization

WN Worker Node

WSDL Web Services Description Language

XML eXtensible Markup Language

B EU DataGrid Software License

EU DataGrid Software License (v2.0, 09/09/2002)

This software includes voluntary contributions made to the EU DataGrid. For more information on the EU DataGrid, please see http://www.eu-datagrid.org/.

Installation, use, reproduction, display, modification and redistribution of this software, with or without modification, in source and binary forms, are permitted. Any exercise of rights under this license by you or your sub-licensees is subject to the following conditions:

1. Redistributions of this software, with or without modification, must reproduce the above copyright notice and the above license statement as well as this list of conditions, in the software, the user documentation and any other materials provided with the software.

2. The user documentation, if any, included with a redistribution, must include the following notice: "This product includes software developed by the EU DataGrid (http://www.eu-datagrid.org/)."

   Alternatively, if that is where third-party acknowledgments normally appear, this acknowledgment must be reproduced in the software itself.

3. The names "EDG", "EDG Toolkit", "EU DataGrid" and "EU DataGrid Project" may not be used to endorse or promote software, or products derived therefrom, except with prior written permission by hep-project-grid-edg-license@cern.ch.

4. You are under no obligation to provide anyone with any bug fixes, patches, upgrades or other modifications, enhancements or derivatives of the features,functionality or performance of this software that you may develop. However, if you publish or distribute your modifications, enhancements or derivative works without contemporaneously requiring users to enter into a separate written license agreement, then you are deemed to have granted participants in the EU DataGrid a worldwide, non-exclusive, royalty-free, perpetual license to install, use, reproduce, display, modify, redistribute and sub-license your modifications, enhancements or derivative works, whether in binary or source code form, under the license conditions stated in this list of conditions.

DISCLAIMER THIS SOFTWARE IS PROVIDED BY THE EU DATAGRID AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, OF SATISFACTORY QUALITY, AND FITNESS FOR A PARTICULAR PURPOSE OR USE ARE DISCLAIMED. THE EU DATAGRID AND CONTRIBUTORS MAKE NO REPRESENTATION THAT THE SOFTWARE, MODIFICATIONS, ENHANCEMENTS OR DERIVATIVE WORKS THEREOF, WILL NOT INFRINGE ANY PATENT, COPYRIGHT, TRADE SECRET OR OTHER PROPRIETARY RIGHT.

LIMITATION OF LIABILITY THE EU DATAGRID AND CONTRIBUTORS SHALL HAVE NO LIABILITY TO LICENSEE OR OTHER PERSONS FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, CONSEQUENTIAL, EXEMPLARY, OR PUNITIVE DAMAGES OF ANY CHARACTER INCLUDING, WITHOUT LIMITATION, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES, LOSS OF USE, DATA OR PROFITS, OR BUSINESS INTERRUPTION, HOWEVER CAUSED AND ON ANY THEORY OF CONTRACT, WARRANTY, TORT (INCLUDING NEGLIGENCE), PRODUCT LIABILITY OR OTHERWISE, ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

C Changing Certificate Formats

C.1 P12 Format to PEM Format

Many of the certificate authorities deliver certificates through a web browser. To use these certificates with Globus, they must be exported from the browser and then reformatted for Globus. Exporting is browser-specific so you will need to follow the help provided with your browser. Once you have extracted the certificate you should have a file with a p12 extension. This file is in the PKCS12 format; you will need to change this to PEM format. If the edg-utils package is installed on your machine, simply executing **/opt/edg/bin/pkcs12-extract** will create appropriate certificate and key files and place them in the standard location. This is a convenience method for the following:

```
openssl pkcs12 -nocerts \
        -in cert.p12 \
        -out ~user/.globus/userkey.pem

openssl pkcs12 -clcerts -nokeys
        -in cert.p12
        -out ~user/.globus/usercert.pem
```

The first command gives you your private key; this file must be readable only by you (e.g. unix permission 0600). The second command gives your public certificate (e.g. unix permission 0644). The `~ user` should be replaced by the path to your home area. The `.globus` subdirectory is standard place to put your certificates.

C.2 PEM Format to P12 Format

Popular browsers typically use certificates in PKCS12 format. Consequently you will need to modify the format of the PEM certificates used for Globus to use them within a browser. To change a certificate from PEM format into PKCS12 format (on a machine with edg-utils installed), just issue the command **/opt/edg/bin/grid-mk-pkcs12**. Again, this is a convenience method for the following:

```
openssl pkcs12 -export \
               -out file_name.p12 \
               -name "My certificate" \
               -inkey ~user/.globus/userkey.pem \
               -in ~user/.globus/usercert.pem
```

where `file_name.p12` is the name of the PKCS12 certificate, and the `~ user` in the last two lines should be replaced by the path to your home area. You must then import the certificate into your browser. (See Section 2.2.2.)

D Information Schema

This appendix describes two major parts of the EDG schema–the GLUE schema and the Service and ServiceStatus tables. Note that in addition to the fields shown here there is a pair of fields appended to each of the R-GMA tables: the "MeasurementDate" and "MeasurementTime".

D.1 GLUE

Most of the published information conforms to the GLUE schema[1] version 1.1. EDG has defined a mapping from the LDAP version of this GLUE schema to a relational schema for R-GMA. This is in three parts for the CE[2], the SE[3] and the CESEBind[4].

The GLUE schema is represented by a UML diagram to describe the relationships between the different objects.

If we consider the example of the GlueCE, we see in the LDAP schema:

```
objectclass ( 1.3.6.1.4.1.8005.100.2.1.1
    NAME        'GlueCE'
    DESC        'Info for Computing Element service'
    SUP         'GlueCETop'
    STRUCTURAL
    MUST        (GlueCEUniqueID)
    MAY         (GlueCEName $ GlueCEHostingCluster))
```

but there are also other classes with a 1:1 relationship with the "GlueCE", such as "GlueCEInfo":

```
objectclass ( 1.3.6.1.4.1.8005.100.2.1.2
    NAME        'GlueCEInfo'
    DESC        'General info for the Queue associated to the CE'
    SUP         'GlueCETop'
    AUXILIARY
    MAY         (GlueCEInfoTotalCPUs $ GlueCEInfoLRMSType $ GlueCEInfoLRMSVersion
               $ GlueCEInfoGRAMVersion $ GlueCEInfoHostName $ GlueCEInfoGatekeeperPort) )
```

and "GlueCEState":

```
objectclass ( 1.3.6.1.4.1.8005.100.2.1.3
    NAME        'GlueCEState'
    DESC        'CE State info'
    SUP         'GlueCETop'
    AUXILIARY
    MAY         (GlueCEStateRunningJobs $ GlueCEStateWaitingJobs $ GlueCEStateTotalJobs
               $ GlueCEStateStatus $ GlueCEStateWorstResponseTime $ GlueCEStateEstimatedResponseTime
               $ GlueCEStateFreeCpus) )
```

[1] http://www.cnaf.infn.it/ sergio/datatag/glue/
[2] http://hepunx.rl.ac.uk/edg/wp3/documentation/doc/schemas/Glue-CE.html
[3] http://hepunx.rl.ac.uk/edg/wp3/documentation/doc/schemas/Glue-SE.html
[4] http://hepunx.rl.ac.uk/edg/wp3/documentation/doc/schemas/Glue-CESEBind.html

The only attribute which is compulsory (with the MUST keyword) is the "GlueCEUniqueID" which is defined as:

```
attributetype ( 1.3.6.1.4.1.8005.100.2.2.1.1
    NAME        'GlueCEUniqueID'
    DESC        'A CE Unique ID'
    EQUALITY    caseIgnoreIA5Match
    SUBSTR      caseIgnoreIA5SubstringsMatch
    SYNTAX      1.3.6.1.4.1.1466.115.121.1.26
    SINGLE-VALUE)
```

You will note the presence of the keyword SINGLE-VALUE indicating that this attribute may only appear once in an object.

Now take a look at the GlueCE table:

UniqueID	VARCHAR(128)	A CE Unique ID
Name	VARCHAR(255)	name of this CE could be the name of the local queue associated with it
GlueClusterUniqueID	VARCHAR(100)	Relates to GlueCluster
TotalCPUs	INT	Number of CPUs available to the queue
LRMSType	VARCHAR(255)	Name of local resource management system
LRMSVersion	VARCHAR(255)	Version of local resource management system
GRAMVersion	VARCHAR(255)	The GRAM version
HostName	VARCHAR(128)	Fully qualified hostname for host where gatekeeper runs
GatekeeperPort	VARCHAR(128)	Port number for the gatekeeper
RunningJobs	INT	Number of jobs in a running state
WaitingJobs	INT	Number of jobs that are in a state different than running
TotalJobs	INT	Number of jobs in the CE
Status	VARCHAR(255)	States a queue can be in
WorstResponseTime	INT	Worst time between job submission till when job starts its execution
EstimatedResponseTime	INT	Estimated time between job submission till when job starts its execution
FreeCpus	INT	Number of free CPUs available to a scheduler
Priority	INT	Info about the Queue Priority
MaxRunningJobs	INT	The maximum number of jobs allowed to be running
MaxTotalJobs	INT	The maximum allowed number of jobs in the queue
MaxCPUTime	INT	The maximum CPU time allowed for jobs submitted to the CE in mins
MaxWallClockTime	INT	The maximum wall clock time allowed for jobs submitted to the CE in mins
InformationServiceURL	VARCHAR(128)	URL of the GRIS

The table name is identical to the main LDAP objectclass (in this case "GlueCE") and the attributes which must be globally unique in LDAP are taken from the LDAP names but without the prefix of the LDAP objectclass. So "GlueCEUniqueID" becomes simply "UniqueID". It is shown bold as it is the primary key of the table. The next field is "Name" (derived from "GlueCEName").

The next field would have been derived from the LDAP attribute "GlueCEHostingCluster", however this attribute is deprecated so we skip over it.

The next field is "GlueClusterUniqueID". This is the standard way we have to chosen to express relationships by means of a foreign key. This is constructed by taking the name of the related table ("GlueCluster") and appending the name of the primary key of that table - which in this case is also "UniqueID" as attributes are local to a table and not global.

The next bunch of attributes is derived from those of the object classes "GlueCEInfo" and "GlueCEState" and "GlueCEPolicy" (not shown above).

Finally we have the LDAP object class which has one compulsory attribute - but it is not single valued:

```
attributetype ( 1.3.6.1.4.1.8005.100.2.2.5.1
    NAME        'GlueCEAccessControlBaseRule'
    DESC        'The rule that grant/deny access of this service'
    EQUALITY    caseIgnoreIA5Match
    SUBSTR      caseIgnoreIA5SubstringsMatch
    SYNTAX      1.3.6.1.4.1.1466.115.121.1.26)

objectclass ( 1.3.6.1.4.1.8005.100.2.1.5
    NAME        'GlueCEAccessControlBase'
    DESC        'Info of a VO which users are allowed to access the CE'
    SUP         'GlueCETop'
    AUXILIARY
    MUST        (GlueCEAccessControlBaseRule) )
```

So a GlueCE may have many values of "GlueCEAccessControlBaseRule". This problem of repeated attributes is handled in the normal way in the relational model by defining an extra table "GlueCEAccessControlBaseRule"

GlueCEUniqueID	VARCHAR(128)	Relates users to CE
Value	VARCHAR(128)	The rule that grant/deny access of this service

This table name is identical to that of the LDAP attribute. The value of the quantity is called "Value" and in each case there is a foreign key back to the object to which it is related. In this case this is the "UniqueID" within the table "GlueCE" so it is "GlueCEUniqeID". Finally note that to identify a row uniquely within this table requires both attributes – so the primary key is a pair of fields.

Looking again at the relational schema you will see the type of each attribute and a comment.

D.2 Service and ServiceStatus

The Service information is meant to represent what services should exists and the ServiceStatus their current status. The tables are defined in misc[5] and look like:

Service

URI	VARCHAR(255)	URI to contact the service
VO	VARCHAR(50)	Where info should be published - or an empty string to indicate all
type	VARCHAR(50)	Type of service (e.g. EDGResourceBroker)
emailContact	VARCHAR(50)	The e-mail of a human being to complain to
site	VARCHAR(50)	Domain name of site hosting the service
secure	VARCHAR(1)	"y" or "n" - indicates whether or not this is a secure service
majorVersion	INT	Version of protocol not implementation
minorVersion	INT	Version of protocol not implementation
patchVersion	INT	Version of protocol not implementation

[5] http://hepunx.rl.ac.uk/edg/wp3/documentation/doc/schemas/misc.html

ServiceStatus

URI	VARCHAR(255)	URI to contact the service
status	INT	status code. 0 means the service is up.
message	VARCHAR(255)	Message corresponding to status code

Rows are published to the Service table around once an hour by a cron job. This shows which services ought to exist. Information system code checks the published services periodically and publishes their status in the ServiceStatus table.

E GridFTP

Management and transfer of files is now better handled by the replica manager commands; however direct use of the GridFTP commands is sometimes useful for debugging.

E.1 File Transfers

There is often a need to manually transfer files from one node to another. The tool for doing so is **globus-url-copy**; the older **gsincftp** commands are deprecated by Globus and no longer part of the EDG release. The command syntax

```
globus-url-copy [options] sourceURL destURL
```

is rather simple and the *-help* option sufficiently explains the command's options.

Unfortunately, the inline help does not list the protocols supported or give examples of the syntax. The supported protocols are: file, gsiftp, and http. Examples are:

```
file:///home/loomis/stuff.txt
gsiftp://testbed011.cern.ch/~/stuff.txt
gsiftp://testbed011.cern.ch//tmp/stuff.txt
http://marianne.in2p3.fr/datagrid/documentation/daemon-guidelines.html
```

For the file protocol only absolute file names are accepted. All of the URLs must be fully specified, e.g. you cannot omit the file name on the output URL. For the gsiftp protocol, the tilde syntax can be used to specify a home area.

One great advantage of **globus-url-copy** is its ability to make third-party transfers. This allows transfers between two remote machines without having to funnel the data through your own machine. This avoids copying the data twice and is especially important if you are executing the command from a machine with a slow network connection or with insufficient disk space.

E.2 Client Commands

There are a set of GridFTP client commands available to do simple management of directories and files in a GridFTP server. These commands are:

```
edg-gridftp-exists
edg-gridftp-ls
edg-gridftp-mkdir
edg-gridftp-rename
edg-gridftp-rm
edg-gridftp-rmdir
```

and perform the equivalent of their unix namesakes. The only unusual one, **edg-gridftp-exists**, checks for the existence of a file or directory. Man pages for each of these commands list the valid options and arguments. All of the commands also recognize the *--help* option.

Note: The **edg-gridftp-rename**, **-rm**, and **-rmdir** commands should only be used on files which are *not* managed by a higher-level service such as the Replica Manager. Using them in this situation may destroy the coherency of the replica database.