GAT Beginners Guide

Alexander Beck–Ratzka
Max–Planck–Institut for Gravitational Physics, Potsdam, Germany

Thomas Zangerl
Nordic DataGrid Facility/PDC, Stockholm, Sweden

February 28th 2009

1 Introduction

The Java implementation of GAT is an API which enables an unique and easy coding of Grid applications. GAT has been developed first in C within the Gridlab project[1] at the AEI[2]. Later on the Vrije Universiteit Amsterdam developed the Java implementation of GAT. Due to the "Early Adaptor Binding" in C–GAT, and also due to the fact that C–GAT does not provide all client operations of the Grid middlewares Globus, gLite, and Unicore by itself, and also because it is written in C[3] C–GAT is only scarcely used in D–Grid. Therefore C–GAT is not supported anymore, instead of that, the C++ implementation of SAGA[4] is the substitute of C–GAT.

JavaGAT (simply called GAT later on) offers all the client operations of the Grid middlewares Globus, gLite, and Unicore without the necessity of installing these middlewares. With GAT it is therefore possible to get a Laptop or Desktop ready for the Grid in only 5 minutes, what is nothing compared to the time needed for a Globus installation, which usualy takes several hours.

Beside that it is much easier to code against the GAT–API then against the API of a Grid–Middleware[5]. An user which want to run his applications on the Grid needs only to program against the GAT–API, and all the difficult stuff of coding against the middleware itself is left over to GAT. GAT contacts the middleware with its corresponding adaptor.

Another advantage of programming against GAT then against a middleware directly is that one needs not to change the code again, if another middleware should be accessed. In such a case GAT uses internally the adaptor for the other middleware, and the user even does not recognize it.

GAT provides adaptors for Globus (GRAM as WSGRAM), gLite, Unicore 6, PBS, SGE, and also so–called local adaptors. The availability of the local adaptors is quite helpful during the development phase, because they enable testing the program logic without having access to the Grid.

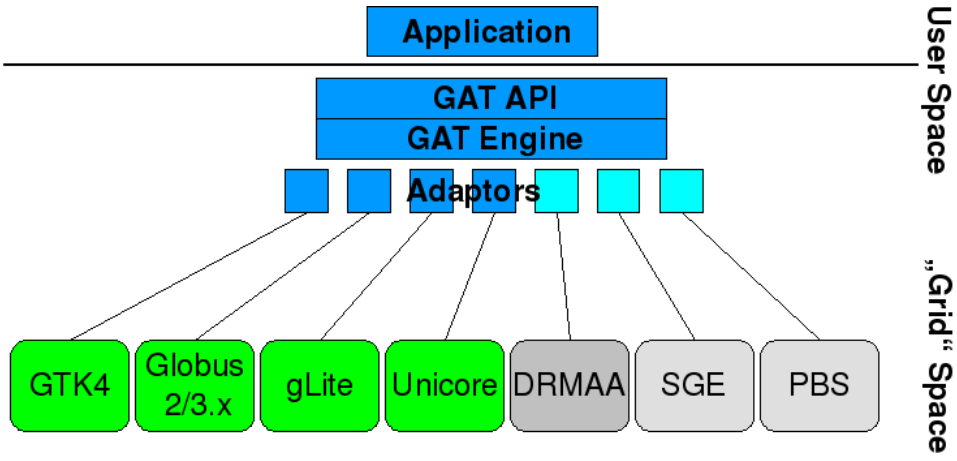

Figure 1: GAT Architecture

[1] www.gridlab.org

[2] AEI stands for Albert–Einstein–Institute, which is the second name of the Max–Planck–Institute for Gravitational Physics in Potsdam–Golm

[3] Most of the Workflow engines which could use GAT to get access to the Grid are written in Java, e.g. Triana, ProC

[4] SAGA: Simple API for Grid Application is a common OGF effort, written in C++ and Java, and thought to be the follow up of GAT. For more information see https://forge.gridforum.org/projects/saga-rg/

[5] see http://www.cs.vu.nl/ rob/JavaGAT-tutorial.pdf for example

2 How to get and install GAT

2.1 Download and installation of GAT

GAT can be checked out from:

`https://svn.aei.mpg.de/eScience/GridAPI/trunk/JavaGAT`

With the user `anonymous` and an empty password.

Before installing GAT, extract it to a directory of your choice. The installation of GAT requires a Java SUN JDK version 6 or newer. There are two ways to install GAT:

1. installation of engine, adaptors, tests and javadoc in one step;
2. separate installation of the engine and the adaptors.

2.1.1 The one step installation

For the one step installation:

1. Change to the directory, where you have extracted GAT to.
2. Set the environment variable `GAT_LOCATION` to point to this directory. If `GAT_LOCATION` is not set, the build process will fail!
3. Enter `ant` to build the engine, the adaptors, some tests, and the documentation.

The documentation is of type javadoc, and the entry page to this documentation is

`$GAT_LOCATION/doc/javadoc/index.html`

The jar–files necessary for the GAT–Engine resides in `$GAT_LOCATION/engine/lib`, those of the adaptors can be found in `$GAT_LOCATION/engine/lib`

2.1.2 Separate build

If you are only interested in the engine and the adaptors, you can build them separately. Before starting the build, please ensure that the environment variable `$GAT_LOCATION` points to the parent directory of GAT. To build the engine change to the directory `$GAT_LOCATION/engine`, and enter `ant`. For building the adaptors, please change to `$GAT_LOCATION/adaptors`, and again enter ant. That's all.

2.1.3 Build the supplemental AstroGrid packages

The command line interfaces **GATJobRun** and **GATFileOps**, which have been developed within the AstroGrid–D project, must be build separately. The build of the packages require a previous build of the GAT engine and the GAT adaptors.

To build the **GATJobRun** package, change to the directory

`$GAT_LOCATION/astrogrid-packages/GATJobRun`,

and enter

`ant`

If you haven't installed a Sun Grid Engine (SGE) on your computer the build process displays the message:

```
SGE_ROOT = '/opt/SGE' found.  If empty, you can't submit jobs to SGE.
Please set SGE_ROOT and rebuild GATJobRun, if you like to use SGE
```

This indicates that is is not possible to use the SGE adaptor of GAT without an SGE on your computer.

For the `GATFileOps` package please change to

`$GAT_LOCATION/astrogrid-packages/GATFileOps`,

and enter

`ant -f build-standalone.xml`.

We recommend to have a look at the Readme files, before building the packages.

3 The Command Line Interface

The command line interface (CLI) packages GATJobRun and GATFileOps represent an extension to GAT. They have been developed within in the AstroGrid project by the AEI in Potsdam–Golm.

3.1 Submitting (Grid) jobs using the GATJobRun package

The entry point to GATJobRun is the script `gat-job-run` which is located in the directory

`$GAT_LOCATION/astrogrid-packages/GATJobRun/scripts`.

It is recommended to add this directory to the `PATH` environment variable, to enable an easy usage from everywhere directory in your system. If `gat-job-run` is called without arguments, it prompts a list of options and arguments:

```
Missing host and/or executable string

Usage:  gat-job-run [OPTIONS] hostname executable [ARGUMENTS]

OPTIONS:
-username [STRING] username for security context
-password [STRING] password for security context

-SubmitOnly [true/false] submit only (true) or poll still exit
```

```
-stdin [FILE] path to input file
-stdout [FILE] path to output file
-stderr [FILE] path to error output file

-prestage [FILE],...  path to prestage file(s) - comma separated
-poststage [FILE],...  path to poststage file(s) - comma spearated

-RB.adaptor [STRING] force the use of a specific Resource Broker adaptor
-RB.jobmanager [STRING] force the use of a specific Resource Broker jobmanager
```

Some comments to the options / arguments.

- `hostname, executable`

  : The name of the execution host, and the name of the executable. It is mandatory, to enter them.

- `-SubmitOnly`: The default behaviour of `gat-job-run` is to submit a job, and to poll after the status of the job until it has finished. After the end of the job, `gat-job-run` exits. This behaviour is only useful for short test jobs. In case of long term runs, it is desired that `gat-job-run` exits before the job has finished. This can be achieved, by setting `SubmitOnly` to `true`.

- `-RB.adaptor`: Here it is possible to enter the name of the GAT adaptor that shall be used for the job submission. If no adaptor is specified explicitly, GAT uses the first job submission adaptor which can be invoked successfully. The name of an adaptor is obtained by the name of its jar file without the suffix `.jar` The GAT adaptors are located in `$GAT_LOCATION/adaptors/lib`. The name of the jar file which contains the gram adaptor is `GlobusResourceBrokerAdaptor.jar`. So if you like to invoke the gram adaptor of GAT you have to enter:

  `-RB.adaptor GlobusResourceBrokerAdaptor`

- `-RB.jobmanager`: Enables to select the usage of a job manager as SGE or PBS. If no job manager is specified here, the default job manager will be invoked; usually this would be fork.

The script `gat-job-run` consists mainly of the java call for the GATJobRun package. However, you can define some variables for GAT in this call; the most important one are:

- `-Dgat.adaptor.path=<path name>` the name of the directory where the adaptors (there jar files) are located;

- `-Dgat.debug` set the logging level to debug, GAT writes debug and info messages to the console, very helpful in case of problems.

3.2 File operations with GATFileOps

Like GATJobRun GATFileOps also uses a script as entry point. It is called `gat-file-operation`, and it is located in the directory

$GAT_LOCATION/astrogrid-packages/GATJobRun/scripts.

Again we recommend to add this directory to the PATH. Calling gat-file-operation without arguments, will deliver list of possible options and arguments:

```
Usage:  gat-file-operation [OPTIONS] <operation> <source> <dest>

OPTIONS:

-Adaptor [STRING] force the use of a specific file adaptor

<operation>:  RemoteCopy, RemoteMove or DeleteFile
<source>:  source file
<dest>:  destination file; not used at <operation> DeleteFile
```

The only option that is available is -Adaptor, and it can be used to force the usage of a special adaptor. The other arguments are necessary for a call of gat-file-operation, in order to obtain the desired operation.

- <operation>: This argument can take the values RemoteCopy, RemoteMove, or DeleteFile. RemoteCopy copies a file, if desired from one grid host to another; RemoteMove moves a file through the grid, and DeleteFile deletes a file anywhere in the grid.
- <source> The source file of the operation.
- <dest> The destination file of the operation. The destination file will be ignored in case of DeleteFile.

Both <source> and <dest> can be file names or URIs, dependent on where the files are located. In order to benefit of the flexibility of GAT, Java–GAT offers its own protocol type "any". So a RemotyCopy call with gat-file-operation might look as follows:

```
gat-file-operation RemoteCopy any://localhost.de//home/ali/bin/test-geo600.sh
                              any://remotehost.de//tmp/tester.sh
```

If any is entered instead of file or gsiftp, any adaptor that can be used for the copy will be invoked. Using gsiftp will force the usage of gridftp for the transfer.

4 Programming against the GAT–API

In this chapter we don't provide a description of the whole GAT–API, this is left to the GAT documentation. However, because we have developed job submission adaptors for gLite, Unicore, SGE, and PBS within the AstroGrid project, we present two short examples describing the coding against the GAT–API for file operations, and for job submission.

4.1 File operations on entire files with GAT

If you like to copy a file from one URI (or file) to another, your class might look as follows:

```
1   import org.gridlab.gat.*;
2   import org.gridlab.gat.io.File;
3
4   public class RemoteCopy {
5      public static void main(String[] args) throws Exception {
6         GATContext context = new GATContext();
7
8         URI src = new URI("file://src_host.grid.org//home/griduser/data/file1");
9         URI dest = new URI("file://dest_host.grid.org//home/griduser1/data/file2");
10        File file = GAT.createFile(context, src); // create file object
11
12        file.copy(dest); // and copy it
13        GAT.end();
14     }
15 }
```

Some comments to the code. To enable any further GAT operation, e. g. the creation of GATFile or a GATResourceBroker (the latter is required for the submission of jobs, you need to create a new instance of the object `GATContext`, as shown in line 6.

The example class above is called with two arguments, namely the source, and the destination file. These files must be given in an URI description, e.g. as `ftp://10.0.0.1/output`. Of course you can use here also the protocol type `any://` for allowing GAT to use any protocol (via available GAT adaptor) which can be used for a file operation. This is the concept of **late binding** realized in GAT. Due to this concept, GAT uses the first adaptor for a file operation, which can invoked successfully. If it is desired to force the usage of a specific adaptor, you must enter the protocol you want to use, e.g. `ftp://, gsiftp://, http://, file://,`, etc...

The GATFile inherits all methods of the Java File class, and offers extensions as `copy` or `move`, and via the corresponing GAT adaptors it is possible to perform file operations in a Grid environment. For a detailed description please look at the JavaDoc of JavGAT. In our example the method `copy` (line 12) is used, to copy the file. Every GAT program should call `GAT.end` before exiting the class. This terminates all threads which might be still alive.

4.2 Job submission with the ResourceBroker

At the begin again a screenshot of an example class, which enables to submit a job.

```
1     import org.gridlab.gat.*;
2     import org.gridlab.gat.io.File;
3     import org.gridlab.gat.resources.*;
4
5
6     public class SubmitJob {
7           public static void main(String[] args) throws Exception {
8                GATContext context = new GATContext();
9
10
11               URI LocalFile=new URI("file:///~/file1");
12               URI RemoteFile=new URI("file://host.grid.org/~/file2");
13               URI PostRemoteFile=new URI("file://host.grid.org/~/result");
```

```
14              URI PostLocalFile=new URI("file:///~/result");
15
16              SoftwareDescription sd = new SoftwareDescription();
17              sd.setExecutable("/bin/hostname");
18              File stdout = GAT.createFile(context, "hostname.txt");
19              sd.setStdout(stdout);
20              sd.addPreStagedFile(GAT.createFilecontext,LocalFile),
21                                      GAT.createFile(context,RemoteFile));
22              sd.addPostStagedFile(GAT.createFilecontext,PostRemoteFile),
23                                      GAT.createFile(context,PostLocalFile));
24
25              JobDescription jd = new JobDescription(sd);
26
27              String JobManagerContact = new String("any://" + "host.grid.org");
28              ResourceBroker broker = GAT.createResourceBroker(context,
29                                              new URI(JobManagerContact));
30
31              Job job = broker.submitJob(jd);
32              while (job.getState() != Job.STOPPED &&
33                        job.getState() != Job.SUBMISSION_ERROR) Thread.sleep(1000);
34
35              GAT.end();
36              }
37          }
38      }
```

To submit a job, you need to create a `SoftwareDescription`, which is required for the later creation of a `JobDescription`, and the `JobDescription` is the only argument you need to create an instance of the `ResourceBroker` class.

Among others, the `SoftwareDescription sd`

offers the following methods:

- `sd.setExecutable("executable"`, to add the name of the executable;
- `sd.SetArguments(String[] jobargs)`, to add the executables arguments;
- `sd.setStdin(GATFile Input, sd.setStdout(GATFile Output))`, allows to define stdin and stdout,
- `sd.addPreStagedFile(GATFile File, sd.addPostStagedFile(GATFile File`, can be used, to define files for pre, and poststaging,
- `sd.addAttribute(String arg0, Object arg1)`, this method allows to add special attributes as the number od nodes or the memory required for the job. Please contact the adaptor developers if you are not sure, how this attributes must be called.

The `SoftwareDescription` contains a list of variables which must be known, to execute the program For a complete description please have a look at the Java documentation of GAT.

The `JobDescription` is necessary for submitting a job, checking its status, killing it, etc... You can create this `JobDescription` with the command:

```
JobDescription jd = new JobDescription(SoftWareDescription sd);
```

The last step before submitting a job, is to get a `ResourceBroker`. The `ResourceBroker` requires the information about the exeution host, given in an URI representation. In the example above this information is given by `JobManagerContact` (line 27). Having defined the execution like this the ResourceBroker is created by:

```
ResourceBroker broker = GAT.createResourceBroker(context, new URI(JobManagerContact);
```

Having the the `ResourceBroker broker` available, it is possible to submit a job simply by:

```
Job job = broker.submitJob(jd);
```

The `Job` object `job` given back in case of a successful job submission allows operations on the job, e.g. as:

- `job.getState`, to check the status of the job;
- `job.stop`, to stop a job;
- `job.hold`, to hold a job;
- `job.resume`, to resume a job;
- `job.getExitStatus`, to get exit status of a job.

However, the final job scheduler must support these operations.

5 Using Grid adaptors with GAT

The usage of the Grid adaptors in GAT requires some configuration on the computer, where these adaptors should be used. These configurations only concerns providing the necessary certificates.

5.1 The usage of the Globus adaptors of GAT

To use the Globus adaptors it is necessary to provide the Globus Grid certificates. There are two types of certificates which must be available:

1. the personnel certificates `userkey.pem`, and `usercert.pem`, which must be located in the directory `$HOME/.globus`;
2. the host certificates of the Grid hosts you like to access, which must be located in the directory `$HOME/.globus/certificates`.

To create a proxy certificate you can use the `grid-proxy-init` (or `grid-proxy-init.bat` on Windows) of GAT, which is located in `$GAT_LOCATION/bin`. The script invokes the class `org.globus.tools.ProxyInit` which is a class of the Java Cog Kit, and delivered with GAT. This class reads in some configuration information from the dataset `$HOME/.globus/cog.properties`, but only if this file is available. `cog.properties` might look as follows

```
#Java CoG Kit Configuration File
#Tue Aug 28 10:41:58 CEST 2007
usercert=/home/alibeck/.globus/usercert.pem
userkey=/home/alibeck/.globus/userkey.pem
proxy=/tmp/x509up_u1000
#cacert=/etc/grid-security/certificates
cacert=/home/alibeck/.globus/cog-certificates
```

The single parameters have the following meaning:

- **usercert**: The path to the file containing your grid certificate.
- **userkey**: The path to the file containing your Grid key.
- **proxy**: The name under which your proxy certificate which you create with `grid-proxy-init` is stored.
- **cacert**: The path of the directory, which contains the host certificates.

Without any configurations in `cog.properties`, the CoG Toolkit will try to find the required files in default locations according to the following rules:

- If the `usercert` property is not set, first the `X509_USER_CERT` system property is checked. If the system property is not set, the value defaults to `$HOME/.globus/usercert.pem`.
- If the `userkey` property is not set, first the `X509_USER_KEY` system property is checked. If the system property is not set, the value defaults to `$HOME/.globus/userkey.pem`.
- If the `cacert` property is not set, first the `X509_CERT_DIR` system property is checked. If the system property is not set, then the `$HOME/.globus/certificates` irectory is checked. If the directory does not exist, and on a Unix/Linux machine, the directory with the name `/etc/grid-security/certificates` is checked next. If one of these directories with certificates is found, all the certificates in that directory will be loaded and used. If no directory is found, the Java CoG will not work.
- If the `proxy` property is not set, first the `X509_USER_PROXY` system property is checked. If the system property is not set, then it defaults to a value based on the following rules: If a `UID` system property is set, and running on a Unix/Linux machine it returns `/tmp/x509up_u$UID`. If any other machine then Unix/Linux, it returns `${tempdir}/x509up_u${UID}`, where tempdir is a platform–specific temporary directory as indicated by the java.io.tmpdir system property. If a `UID` system property is not set, the username will be used instead of the `UID`. That is, it returns `${tempdir}/x509up_u_${username}`.

5.2 How to use the GAT-gLite Adaptor

5.2.1 Introduction

Regarding security, the gLite adaptor behaves mostly like Globus. That means, that a valid proxy impersonating the user on the Grid is required. The difference with gLite, however, is that instead of

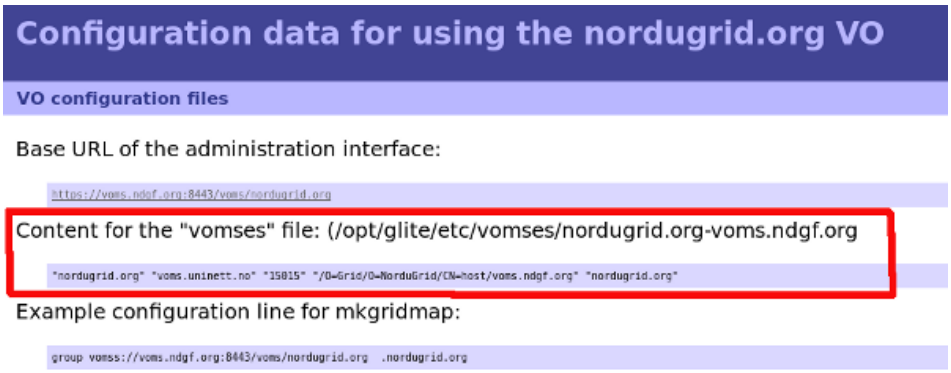

Figure 2: The membership details page in the VOMS Admin gives you the DN of the VOMS host

an entirely self-signed proxy, gLite uses so-called VOMS proxies for authentication and authorization. VOMS proxies are enhanced Globus proxies, that contain extensions signed by a VOMS server about the user's roles, group memberships and capabilities to simplify authorization at the Grid site level and spare the site administrators the task of updating huge user databases. In order to receive a VOMS proxy, users must contact a VOMS server in a connection secured with a Globus proxy and request certain roles, groups and capabilities. If the user is a member in the VO (virtual organisation), that is administered by the VOMS server, this request will either be granted or denied based on the suitability of the user for the requested memberships. If the request is granted, the server will generate a signed certificate containing the granted request and store that in the user's proxy. The user can now impersonate herself at Grid sites belonging to the same VO as she does and will receive authorization compliant to the granted capabilities.

5.2.2 Configuration of the gLite adaptor

To be able to make the VOMS-proxy request on behalf of the user, the gLite adaptor needs to know a few additional pieces of data:

- The name of the VO for which the user wants to obtain a credential (e.g. nordugrid, knowarc, VOCE, ...)
- The endpoint of the VOMS server webservice (this address is usually different to the URL at which the VOMS admin can be accessed with a browser)
- The port at which the VOMS server is listening to requests
- The distinguished name (DN) of the VOMS Host. If you are unsure about this, you can usually find the information on the "Configuration" page in the VOMS admin server application. (Fig. 2)

These values have to be specified as preferences of the current GAT context. Additionally, a CertificateSecurityContext with the path to the user certificate and key must be given.

An example configuration of all the necessary parameters for the gLite adaptor could look as follows:

```
GATContext context = new GATContext();
CertificateSecurityContext secContext =
    new CertificateSecurityContext(
    new URI("/home/mustermann/.globus/userkey.pem"),
    new URI("/home/mustermann/.globus/usercert.pem"),
    "mysupersecretpwd");

Preferences globalPrefs = new Preferences();
globalPrefs.put("vomsServerURL", "skurut19.cesnet.cz");
globalPrefs.put("vomsServerPort", "7001");
globalPrefs.put("vomsHostDN",
                "/DC=cz/DC=cesnet-ca/O=CESNET/CN=skurut19.cesnet.cz");
globalPrefs.put("VirtualOrganisation", "voce");
context.addPreferences(globalPrefs);
context.addSecurityContext(secContext);
```

Please also note that a correct `cog.properties` file as described in 5.1 should be present.

5.2.3 Optional configuration parameters

Security

By default, the gLite-adaptor creates a VOMS proxy valid for 12 hours. If another validity period is desired, this behaviour can be influenced by setting the `vomsLifetime` preference key. For example, if you want to set the lifetime to 1 hour, do:

```
Preference globalPrefs = new Preferences();
globalPrefs.put("vomsLifetime", "3600");
context.addPreferences(globalPrefs);
```

Please note, that the gLite adaptor will reuse any proxy that is valid longer than `vomsLifetime` or 10 minutes, if `vomsLifetime` is not present.[6] If you want the adaptor to explicitly construct a new proxy every time a job is submitted, you can instruct it to do so by setting the preference key glite.newProxy. For example

```
Preference globalPrefs = new Preferences();
globalPrefs.put("glite.newProxy", "true");
context.addPreferences(globalPrefs);
```

forces the gLite-adaptor to construct a new proxy for the submitted job, even if an existing one is valid for a longer period than `vomsLifetime`. This behaviour is required, if you are submitting to sites belonging to different VOs.

Matchmaking

[6] The gLite adaptor will not perform automatic renewal, if the proxy times out during the execution of a job.

SD attribute	Mapping to JDL
time.max	other.GlueCEPolicyMaxWallClockTime
walltime.max	other.GlueCEPolicyMaxWallClockTime
cputime.max	other.GlueCEPolicyMaxCPUTime
project	HLRLocation
memory.min	other.GlueHostMainMemoryRAMSize >=
memory.max	other.GlueHostMainMemoryRAMSize <=
glite.retrycount	RetryCount
glite.DataRequirements.InputData	DataRequirements = { [DataCatalogType="RLS"; InputData={...}] } DataAccessProtocol = {https, gsiftp};}

Table 1: List of SoftwareDescription attributes and their mapping to JDL

ResourceRequirement	Mapping to JDL
os.name	other.GlueHostOperatingSystemName
os.release	other.GlueHostOperatingSystemRelease
os.version	other.GlueHostOperatingSystemVersion
os.type	other.GlueHostProcessorModel
cpu.type	other.GlueHostProcessorModel
machine.type	other.GlueHostProcessorModel
machine.node = (String|List <String>)	other.GlueCEUniqueID
cpu.speed	other.GlueHostProcessorClockSpeed
memory.size	other.GlueHostMainMemoryRAMSize
disk.size	other.GlueSESizeFree
glite.other	any other.Glue* attributes here (that you specify yourself)

Table 2: List of SoftwareResourceRequirements and HardwareResourceRequirements and their mapping to JDL

Preferences influencing matchmaking decisions are set either as SoftwareDescription attributes or as SoftwareResourceDescription and HardwareResourceDescription properties, as previsioned in JavaGAT. Most attributes are supported, but since the set of previsioned attributes is somewhat tailored to Globus' RSL format, this can only be done where the attributes can be translated to JDL in a feasible way, or can be included as GLUE requirements. Table 1 gives an overview of the supported software description attributes and how they are handled in the JDL document, while table 2 shows how SoftwareResourceDescription and HardwareResourceDescription properties are translated to the JDL. If not noted otherwise, input parameters in form of attributes or resource description properties are always strings and the comparison operator in the JDL is equality.

In gLite, matchmaking is supposed to be performed at the level of the workload management server (WMS), which acts as a meta-scheduling server. The WMS uses information stored in the information supermarket, which is usually also accessible from outside the WMS and can be used to perform manual preselections by the adaptor. For example, if you don't know the address of an WMS, but instead the address of your VOs LDAP information service, you can do:

```
ResourceBroker rb =
       GAT.createResourceBroker(context, "ldap://is.example.com");
Job job = rb.submitJob(jobDesc);
```

The gLite adaptor will then try to retrieve a WMS accepting jobs for your VO from the LDAP server automatically and submit to this one, or, if more than one WMS accept jobs, submit to a random one of these.

Miscellenaeous configuration parameters

By default, the gLite resource broker adaptor will delete the JDL file created upon job submission. If you want to keep the JDL, e.g. for debugging purposes, set the preference key glite.deleteJDL to false.

The glite.pollIntervalSecs preference key determines how often the gLite resource broker adaptor polls the logging & bookkeeping service for the current job's status. By default, this is 3 seconds.

For example, if you want to invoke the gLite adaptor with the setting that the JDL files should be kept and that the L&B server should be polled all 10 seconds, do the following:

```
Preference globalPrefs = new Preferences();
globalPrefs.put("glite.deleteJDL", "false");
globalPrefs.put("glite.pollIntervalSecs", "10");
context.addPreferences(globalPrefs);
```

5.3 How to use the GAT-Unicore Adaptor

5.3.1 Introduction

The JavaGAT Unicore adaptor is based on HiLA[7] Therefore some HiLA specific configuration is necessary. This configuration stuff is also described in this document.

5.3.2 Installation

The Unicore adaptor is now part of the regularly distributed adaptors, so it is build during the default build process (see section 2).

5.3.3 Configuration of the HiLA

The entry point for the configuration is the dataset `unicore6.xml`. It might look as follows:

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- This is the default unicore6.xml. HiLAFactory will
 look for it on the classpath, if all else fails. -->
```

[7] The HiLA package is described in http://www.unicore.eu/community/development/hila-reference.pdf.

```
<!-- Use this file as an example unicore6.xml. -->

<beans xmlns:hila="http://www.unicore.eu/hila-unicore6">
 <hila:unicore6grid id="grid" outcomeDirectory="file:${user.home}/.hila/data" config="#config" />

 <hila-common:compositeconfig id="config" xmlns:hila-common="http://www.unicore.eu/hila-common">
 <constructor-arg>
  <list>
  <hila:registryconfig
       registryURL="https://zam461.zam.kfa-juelich.de:9117/AWARE-GROW/services/Registry?res=default_registry" grid="#grid"
       securityProperties="#d-grid.security" />
  </list>
 </constructor-arg>
 </hila-common:compositeconfig>

 <bean name="d-grid.security" class="de.fzj.hila.implementation.unicore6.Unicore6SecurityProperties">
  <constructor-arg value="/home/alibeck/.hila/d-grid.security" />
 </bean>

</beans>
```

The path to this configuration file must be added as a definition while calling the Java VM with the `-D` flag, e.g.:

```
java -D/home/user/.hila/unicore.xml.
```

Some notes to `unicore.xml`:

- The outcomeDirectory defines the directory where all the results are stored. The default is $HOME/.hila
- The hila:registryconfig flag defines the security to be used (here d-grid.security), and the default registryURL:
  `https://zam461.zam.kfa-juelich.de:9117/AWARE-GROW/services/Registry?res=default_registry`
- Under the bean name `d-grid.security` the security issues are defined. The `constructor-arg` value tag describes where security configuration can be found. This configuration file might look as follows:

```
unicore.wsrflite.ssl.keystore = /home/alibeck/certdir/alicert.jks
unicore.wsrflite.ssl.keypass = ******
unicore.wsrflite.ssl.keyalias = alip12cert
```

`keystore` ...the name of the keystore file `keypass` ...the password of the keystore `keyalias` ...the alias name of the key in the keystore which shall be used.

5.3.4 How to get a keystore?

We highly recommend to install portecle from

```
http://portecle.sourceforge.net/
```

and use it with `java -jar portecle.jar`. You are then in a graphical menu, which allows to

- create a new keystore
- add your personal and your host certificates to it.

You need to add a p12-Key. However, this p12-key can be created from a pem-key using openssl.

IMPORTANT: Right now, the keystore and the p12 certificate requires the same password. However, this will be changed in the next HiLA release.