



Enabling Grids for E-scienceE

Job Description Language (JDL)

Giuseppe La Rocca

INFN

First gLite tutorial on GILDA

Catania, 13-15.06.2005



www.eu-egee.org



- **Job Description Language – Overview**
- **Relevant attributes**
- **JDL Examples**
 - **csound.jdl**
 - **raster.jdl**
 - **scilab.jdl**

- ✦ In gLite **Job Description Language (JDL)** are used to describe jobs for execution on Grid.
- ✦ The JDL adopted within the gLite middleware is based upon Condor's **CLASSified Advertisement language (ClassAd)**.
 - A ClassAd is a record-like structure composed of a finite number of attribute separated by semi-colon (;)
 - A ClassAd is highly flexible and can be used to represent arbitrary services

*The JDL is used in gLite to specify the desired job characteristics and constrains, which are used in by **match-making process** to select the resources that the job will use.*

- ✚ The **JDL syntax** consists on statements like:

Attribute = value;

- ✚ Comments must be preceded by a sharp character (#) or have to follow the C++ syntax

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

- ✚ In a JDL, some attributes are mandatory while others are optional.
- ✚ A essential JDL is the following:

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

- ✚ If needed, arguments to the executable can be passed:

```
Arguments = "Hello World!";
```

- ✚ If the arguments contains quoted strings, the quotes must be escaped with a backslash

e.g. Arguments = “\”Hello World!\“ 10”;

- ✚ Special characters such as &, |, >, < are only allowed if specified inside a quoted string or preceded by triple \ (e.g. Arguments = "-f file1\\\&file2";)

- ✚ The supported attributes are grouped in two categories:

- Job Attributes

- Define the job itself

- Resources

- Taken into account by the RB for carrying out the matchmaking algorithm (to choose the “best” resource where to submit the job)

- *Computing Resource*

- *Used to build expressions of Requirements and/or Rank attributes by the user*

**Requirements=other.GlueCEUniqueID ==
“adc006.cern.ch:2119/jobmanager-pbs-infinite”**

**Requirements=Member(“ALICE-3.07.01”,
other.GlueHostApplicationSoftwareRunTimeEnvironment);**

Data and Storage resources

- *Input data to process, SE where to store output data, protocols spoken by application when accessing Ses*

InputData = {"lfn:cmstestfile",

"guid:135b7b23-4a6a-11d7-87e7-9d101f8c8b70"};

⚡ JobType (optional)

- Normal (simple, sequential job), Interactive, MPICH, Checkpointable, Partitionable

- Or combination of them

- Checkpointable, Interactive
- Checkpointable, MPI

E.g. JobType = “Interactive”;

JobType = {“Interactive”, “Checkpointable”};

“Interactive” + “MPI” not yet permitted

✚ **Type** (mandatory, default “Job”)

🌐 This is a representing the type of the request described by the JDL.

🌐 Possible values are:

🌐 **Job**





🌐 **DAG**

🌐 **Reservation**

🌐 **Co-allocation**

E.g.: **Type = “Job”;**

Executable (mandatory)

-  This is a string representing the executable/command name.
-  The user can specify an executable which is already on the remote CE
-  `Executable = {"/opt/EGEODE/GCT/egeode.sh"};`
-  The user can provide a local executable name, which will be staged from the UI to the WN.
`Executable = {"egeode.sh"};`
`InputSandbox = {"/home/larocca/egeode/egeode.sh"};`

Arguments (optional)

This is a string containing all the job command line arguments.

E.g.: If your executable sum has to be started as:

```
$ sum N1 N2 -out result.out
```

```
Executable = "sum";
```

```
Arguments = "N1 N2 -out result.out";
```

+ Environment (optional)

- List of environment settings needed by the job to run properly

E.g. `Environment = {"JAVABIN=/usr/local/java"};`

+ InputSandbox (optional)

- List of files on the UI local disk needed by the job for running
- The listed files will automatically staged to the remote resource

E.g. `InputSandbox = {"myscript.sh", "/tmp/cc,sh"};`

✚ **OutputSandbox** (optional)

- 🌐 List of files, generated by the job, which have to be retrieved

E.g. `OutputSandbox = { "std.out", "std.err", "image.png" };`

✚ **VirtualOrganisation** (optional)

- 🌐 This is a string representing the name of the VO the submitting user is currently working for.

E.g. `VirtualOrganisation = "gilda";`

+ Requirements (optional)

- Job requirements on computing resources
- Specified using attributes of resources published in the Information Service
- If not specified, default value defined in UI configuration file is considered

Default. **Requirements =**

other.GlueCEStateStatus == "Production";

✚ Rank (optional)

- Floating-point expression used to rank CEs that have already met the *Requirements* expression.
- The Rank expression can contain attributes that describe the CE in the **Information System (IS)**.
- The evaluation of the rank expression is performed by the **Resource Broker (RB)** during the match-making phase.
- A higher numeric value equals a better rank.
- If not specified, default value defined in the UI configuration file is considered

Default: **Rank = - *other.GlueCEStateFreeCPUs*;**

✚ **InputData** (optional)

- This is a string or a list of strings representing the *Logical File Name (LFN)* or *Grid Unique Identifier (GUID)* needed by the job as input.
- The list is used by the RB to find the CE from which the specified files can be better accessed and schedules the job to run there.

```
InputData = {"lfn:cmstestfile",  
"guid:135b7b23-4a6a-11d7-87e7-9d101f8c8b70"};
```

- ✚ **DataAccessProtocol** (mandatory if `InputData` has been specified)
 - 🌐 The protocol or the list of protocols which the application is able to “speak” with for accessing files listed in *InputData* on a given SE.

- ✚ Supported protocols in gLite are currently **gridftp**, **file** and **rfio**.




`DataAccessProtocol = {"file", "gridftp", "rfio"};`

✚ OutputSE (optional)

- This string representing the URI of the **Storage Element (SE)** where the user wants to store the output data.
- This attribute is used by the Resource Broker to find a CE being “close” to this SE and schedule the job there.

OutputSE = “grid009.ct.infn.it”;

- ✚ **OutputData** (optional) Not yet implemented in gLite
 - 🌐 This attribute allows the user to ask for the automatic upload and registration of datasets produced by the job on the **Worker Node (WN)**.
 - 🌐 This attribute contains the following three attributes:
 - 🌐 *OutputFile*
 - 🌐 *StorageElement*
 - 🌐 *LogicalFileName*

- + **OutputFile** (mandatory if OutputData has been specified)
 -  This is a string attribute representing the name of the output file, generated by the job on the WN, which has to be automatically uploaded and registered by the WMS.
- + **StorageElement** (optional)
 -  This is a string representing the URI of the Storage Element where the output file specified in the OutputFile has to be uploaded by the WMS.
- + **LogicalFileName** (optional)
 -  This is a string representing the LFN user wants to associate to the output file when registering it to the Catalogue.

```
OutputData = {  
  [  
    OutputFile = "dataset1.out";  
    LogicalFileName = "lfn:test-result1";  
  ],  
  [  
    OutputFile = "dataset2.out";  
    LogicalFileName = "lfn:test-result2";  
    StorageElement = "grid009.ct.infn.it";  
  ],  
};
```

```
Type = "Job";  
JobType = "Normal";  
Executable = "/bin/sh";  
MyProxyServer="lxshare0207.cern.ch";  
StdOutput = "csound.out";  
StdError = "csound.err";  
InputSandbox = {"start_csound.sh", "csound.orc", "csound.sco"};  
OutputSandbox = {"csound.aiff", "csound.err", "csound.out"};  
RetryCount = 7;  
Arguments = "start_csound.sh";  
Requirements = Member("CSOUND-4.13",  
    other.GlueHostApplicationSoftwareRunTimeEnvironment);
```



```
#!/bin/bash
date
ls -la
if [ ! -s start_csound.sh ]; then
echo " -----> missing file: start_csound.sh";
  exit 3;
fi
if [ ! -s csound.orc ]; then
  echo " -----> missing file: csound.orc";
  exit 3;
fi
if [ ! -s csound.sco ]; then
  echo " -----> missing file: csound.sco";
  exit 3;
fi
```




```

# create environment for Csound
# Execute Csound
csound -o csound.aiff -A csound.orc csound.sco
date
echo "- check if all output files are there and not empty....."
#-----
-
ls -la
if [ ! -s csound.aiff ]; then
    echo " ==> Missing output file : csound.aiff ";
    exit 2;
fi
rm -f *.sco
rm -f *.orc

```



```
Type = "Job";
JobType = "Normal";

Executable = "/bin/sh";
Arguments = "start_raster.sh";

StdOutput = "raster.out";
StdError = "raster.err";
InputSandbox =
{"start_raster.sh", "Caffeine.pdb", "HMPv66.pdb", "Viagra.pdb", "aspiri
ne.pdb", "insuline.pdb", "penicillin.pdb", "phetrna.pdb"};
OutputSandbox = {"raster.err", "raster.out", "image.png"};

RetryCount = 7;
Requirements =
  Member("RASTER3D", other.GlueHostApplicationSoftwareRunTim
eEnvironment);
```



```

#!/bin/bash
PROTEIN2RENDERING=phetrna.pdb
RASTERTOOLS=balls

echo "Start Rendering $PROTEIN2RENDERING.."
echo "..using the $RASTERTOOLS package."

# Draw smooth $RASTERTOOLS with default color scheme 2,
# save description (with header records) in ribbon.r3d.
if [ "$RASTERTOOLS" == "balls" ] || [ "$RASTERTOOLS" == "rods" ]; then
    cat "$PROTEIN2RENDERING" | "$RASTERTOOLS" > ribbon.r3d
    echo "Creation of ribbon.r3d file..(1st case)"

else
    cat "$PROTEIN2RENDERING" | rods > ribbon.r3d
    echo "Creation of ribbon.r3d file..(2nd case)"

fi

```



```
grep `FE` "$PROTEIN2RENDERING" | balls -h > irons.r3d
if [ "$RASTERTOOLS" == "rings3d" ]; then
    rings3d -bases < "$PROTEIN2RENDERING" > temp.2
    # Combine the three descriptions and render as PNG image file.
    cat ribbon.r3d irons.r3d temp.2 | render -png > image.png
    echo "Creation of IMAGE.PNG!"
    rm temp.2
else
    echo "Creation of IMAGE.PNG!!"
    # Combine the two descriptions and render as PNG image file.
    cat ribbon.r3d irons.r3d | render -png > image.png
fi
# Remove the temporary files.
rm *.pdb
rm ribbon.r3d
rm irons.r3d
```



```
Type = "Job";
```

```
JobType = "Normal";
```

```
Executable = "/bin/sh";
```

```
Arguments = "start_scilab.sh";
```

```
StdOutput = "scilab.out";
```

```
StdError = "scilab.err";
```

```
InputSandbox={"start_scilab.sh","Lorents.dem","Misc.dem","shel..  
dem"};
```

```
OutputSandbox = {"scilab.err","scilab.out","image.gif"};
```

```
Requirements = Member("SCILAB-  
2.6",other.GlueHostApplicationSoftwareRunTimeEnvironment);
```



```
#!/bin/bash
```

```
MACROS2EXECUTE=Lorents.dem
```

```
echo "Start Executing $MACROS2EXECUTE.."
```

```
LOCALDIR=`pwd`
```

```
hostname -f
```

```
#Setting the right permission for the execution of the file.
```

```
chmod 777 start_scilab.sh
```

```
echo Opening a Virtual Frame Buffer for launching the the  
SciLab application.
```

```
/usr/X11R6/bin/Xvfb :1&
```



```
echo Running SciLab..
```

```
echo `/usr/bin/scilab -display :1 -f "$MACROS2EXECUTE"
```

```
echo Listing the content of the workdir.
```

```
ls -l
```

```
echo Converting file into gif file using ImageMagick.
```

```
convert sci1.eps image.gif
```

```
echo Kill the Xvfb.
```

```
process_id=`ps -ax | grep Xvfb | awk '{print $1}'`
```

```
for i in $process_id
```

```
do
```

```
    kill -15 $i
```

```
done
```

```
rm -f *.sh; rm -f *.eps; rm -f *.dem; rm -f *.sce
```

