EGEE

JOB DESCRIPTION LANGUAGE ATTRIBUTES SPECIFICATION

FOR THE GLITE MIDDLEWARE

(SUBMISSION THROUGH WMPROXY SERVICE)

Document identifier:	**EGEE-JRA1-TEC-590869-JDL-Attributes-v0-7**
Date:	**02/05/2006**
Activity:	**Activity: JRA1 - Middleware**
Document status:	
Document link:	**https://edms.cern.ch/document/590869/1**

Abstract: This document provides the specification of the Job Description Language attributes supported by the gLite software (version 3.0.0 or later). Attributes and features described in this document are fully supported only if the job submission to WMS is performed through the **WMProxy**, i.e. the Web services based interface to the gLite Workload Management System.

Document Log

Issue	Date	Comment	Author/Partner
0.1	28/01/2005	Draft	Fabrizio Pacini
0.2	05/05/2005	- Fix DAG attribute name retry => node_retry_count, - Add InputSandbox to partitionable job example. - Add UserTags attribute description. - Add OutputSandboxBaseDestURI attribute description. - Add JobState attribute description. - Add ListenerHost, ListenerPipeName attributes description. - Add JobState attribute description	Fabrizio Pacini
0.3	11/08/2005	- General review - Specify MyProxyServer inheritance for sub-jobs - Add description of LBAddress attribute - Add ExpiryTime attribute - Data requirements attributes clarified - Review partitionable job description - Added example on InputData - Fix composition rules.	Fabrizio Pacini
0.4	14/12/2005	- Add DataRequirements attribute - Fix DAG dependencies attribute description - Add ShallowRetryCount attribute description - Add AllowZippedISB attribute description - Add PerusalFileEnable, PerusalTimeInterval and PerusalFilesDestURI attributes descriptions - Add NodesCollocation attrbute description	Fabrizio Pacini
0.5	04/01/2006	- Fix StdOutput description - Clarify about HTTPS servers support	Fabrizio Pacini

Document Log

0.6	27/01/2006	- Clarify about wildcards support in Input and OutputSandbox - Fix DataAccessProtocols	Fabrizio Pacini
0.7	02/05/2006	- Update HLRLocation description - Naming fix: max_nodes_running ==> max_running_nodes - Clarify NodeName	Fabrizio Pacini

CONTENT

1. INTRODUCTION

The Job Description Language (JDL) is a high-level, user-oriented language based on Condor classified advertisements (classads) for describing jobs and aggregates of jobs such as Direct Acyclic Graphs (DAG) and Collections. Being the JDL an extensible language the user is allowed to use whatever attribute for the description of a request without incurring in errors from the JDL parser. However, only a certain set of attributes, that we will refer as "supported attributes" from now on, is taken into account by the Workload Management System (WMS) components in order to schedule and submit a job or the jobs of a complex request.

JDL attributes represent request-specific information and specify in some way actions that have to be performed by the WMS to schedule the job or the jobs of a complex request.

Some of the attributes in the JDL are mandatory. If the user does not specify them, the WMS cannot handle the request. For the other attributes the system may find a default value if they are necessary for processing the request.

Note that the JDL attributes described in this document represent the whole set of attributes supported by the WMS when accessed through the WMProxy (see [R15]).

Section 2 describes the requests/jobs types supported by the WMS and the way they can be composed. Sections 3 to 7 provide the complete list of the JDL attributes supported by the WMS together with their meaning and format and the rules to follow for building correct requests (i.e. Job, DAG and Collection) descriptions.

Before starting with the detailed attribute description please note that a request description is composed by entries that are strings having the format

attribute = expression;

and are terminated by the semicolon character. The whole description has to be included between square brackets, i.e. [<*job descr.*>]. The termination with the semicolon is not mandatory for the last attribute before the closing square bracket].

Attribute expressions can span several lines provided the semicolon is put only at the end of the whole expression. Comments must have a sharp character (#) or a double slash (//) at the beginning of each line. Comments spanning multiple lines can be specified enclosing the text between "/*" and "*/".

Moreover it is worth noting that the *requirements* and *rank* attribute expressions (see 3.40 and 3.41) that are evaluated by the WM during the match making process for selecting the CEs where the job will be sent, are built using the *resources attributes*, representing the characteristics and status of the resources in the grid. These resources attributes are not part of the predefined set of attributes for the JDL as their naming and meaning depends on the adopted Information Service schema (e.g. in gLite we use the Glue schema [R6]). This independence of the JDL from the resources information schema allows targeting for the submission resources that are described by different Information Services without any changes in the job description language itself.

Please note that since the WMProxy exposes a publicly available WSDL interface, no assumption is made in the document (unless explicitly specified) about the client tool used to submit the JDL description of the job.

1.1. PURPOSE

The purpose of this document is to provide a detailed description of the syntax and semantics of the JDL attributes supported by the WMS when accessed through the WMProxy Service in order to help users in describing their applications to be submitted to the grid.

1.2. APPLICATION AREA

This document applies to the implementation of the EGEE Middleware within the scope of the EGEE Project [R1] and the JRA1 activity mandate.

1.3. REFERENCES

[R1] EGEE: Enabling Grids for E-science in Europe

[R2] EGEE Middleware Architecture, EU Deliverable DJRA1.1, https://edms.cern.ch/document/476451/

[R3] EGEE Middleware Design, EU Deliverable DJRA1.2, https://edms.cern.ch/document/487871/

[R4] Definition of the architecture, technical plan and evaluation criteria for the resource co-allocation framework and mechanisms for parallel job partitioning http://www.infn.it/workload-grid/docs/DataGrid-01-D1.4-0127-1_0.pdf

[R5] DataGrid Accounting System - Architecture v 1.0 http://www.infn.it/workload-grid/docs/DataGrid-01-TED-0126-1_0.pdf

[R6] The Glue Schema - http://infnforge.cnaf.infn.it/glueinfomodel

[R7] Job Description Language HowTo – DataGrid-01-TEN-0102-02 http://www.infn.it/workload-grid/docs/DataGrid-01-TEN-0102-0_2.pdf

[R8] JDL Attributes - DataGrid-01-TEN-0142-0_2 http://server11.infn.it/workload-grid/docs/DataGrid-01-TEN-0142-0_2.pdf

[R9] The Resource Broker Info file – DataGrid-01-TEN-0135-0_0 http://www.infn.it/workload-grid/docs/DataGrid-01-TEN-0135-0_0.doc

[R10] LB-API Reference Document – DataGrid-01-TED-0139-0_0 http://lindir.ics.muni.cz/dg_public/lb_api.pdf

[R11] Job Partitioning and Checkpointing – DataGrid-01-TED-0119-0_3 https://edms.cern.ch/file/347730/1/DataGrid-01-TED-0119-0_3.pdf

[R12] "Gang-Matching in EDG WMS" - DataGrid-01-TEN-014X-0_0

[R13] EGEE Middleware Architecture (revision), EU Deliverable DJRA1.4, https://edms.cern.ch/document/594698/1.0

[R14] EGEE Middleware Design (revision), EU Deliverable DJRA1.5, https://edms.cern.ch/document/606574/1.0

[R15] WMProxy User's Guide – EGEE-JRA1-TEC-674643 https://edms.cern.ch/document/674643/1

[R16] WMS User's Guide - EGEE-JRA1-TEC-572489 https://edms.cern.ch/document/572489/1

[R17] WMProxy Wiki pages http://trinity.datamat.it/projects/EGEE/wiki/wiki.php

1.4. DOCUMENT EVOLUTION PROCEDURE

This document can be amended by the JRA1 IT-CZ cluster as resulting from feedback and discussion with the other teams. Proposals for amendments can also be sent to the JRA1 IT-CZ with a brief description of the proposed change and its benefits. Minor changes, such as spelling corrections, content formatting or minor text reorganization not affecting the content and meaning of the document

can be applied by the author without peer review. All other changes must be submitted to peer review and approval. When the document is modified for any reason its version number shall be increased accordingly. The document version number shall follow the standard EGEE conventions for document versioning. The document shall be maintained using the tools provided by CERN EDMS system.

1.5. TERMINOLOGY

Glossary

class-ad	Classified advertisement
CE	Computing Element
CLI	Command Line Interface
DGAS	Datagrid Grid Accounting Service
DLI	Data Location Interface
EDG	European DataGrid
EGEE	Enabling Grids for E-sciencE
FQDN	Fully Qualified Domain Name
GIS	Grid Information Service, aka MDS
GSI	Grid Security Infrastructure
GUI	Graphical User Interface
HLR	Home Location Register
IS	Information Service
ISM	Information Super Market
job-ad	Class-ad describing a job
JA	Job Adapter
JC	Job Controller
JDL	Job Description Language
JRA	Joint Research Activity
LB	Logging and Bookkeeping
LM	Log Monitor
LRMS	Local Resource Management System
MDS	Metacomputing Directory Service, aka GIS
MPI	Message Passing Interface
NS	Network Server
OS	Operating System
PA	Price Authority
PID	Process Identifier
PM	Project Month
RB	Resource Broker

SD	Service Discovery Service
SE	Storage Element
SI00	Spec Int 2000
SMP	Symmetric Multi Processor
TBC	To Be Confirmed
TBD	To Be Defined
TQ	Task Queue
UI	User Interface
UUID	Universally Unique IDentifier
VO	Virtual Organisation
WM	Workload Manager
WMProxy	Workload Manager Proxy: the web service interface to the WMS
WMS	Workload Management System
WN	Worker Node
WP	Work Package

Definitions

Condor	Condor is a High Throughput Computing (HTC) environment that can manage very large collections of distributively owned workstations
Globus	The Globus Toolkit is a set of software tools and libraries aimed at the building of computational grids and grid-based applications.
Request	An object that can be described through the JDL and has to be processed by the WMS. It can be a job (simple request) or a DAG or a Collection (complex requests).

2. REQUEST AND JOB TYPES

The JDL allows description of the following types of requests (the ones supported by the WMS):

- **Job** a simple job
- **DAG** a Direct Acyclic Graph of dependent jobs
- **Collection** a set of independent jobs

Support for advance reservation and co-allocation requests ("Reservation" and "Co-allocation" types) will be evaluated for the second release of the gLite middleware.

Although DAGs and Collections represent sets of jobs, they are described through a single JDL description and can hence be submitted in one shot to the WMS. Moreover upon submission of such kind of requests, the WMS (besides the ids of the single nodes) will provide the user with a collective identifier that will allow monitor and control of the whole set of jobs through a single handle.

As it is described in the following sections, the JDL structure for DAGs and Collections is in some way similar but they have been kept as separate request types as they represent sets of jobs with different kind of interrelation.

The WMS currently supports the following types for Jobs (this does not apply to DAGs and Collections):

- **Normal** a simple batch job
- **Interactive** a job whose standard streams are forwarded to the submitting client
- **MPICH** a parallel application using MPICH-P4 implementation of MPI
- **Partitionable** a job that can be thought as composed by a set of independent steps/iterations, i.e. a set of independent sub-jobs, each one taking care of a step or of a sub-set of steps, and which can be executed in parallel
- **Checkpointable** a job able to save its state, so that the job execution can be suspended, and resumed later, starting from the same point where it was first stopped
- **Parametric** a job whose JDL contains parametric attributes (e.g. Arguments, StdInput etc.) whose values can be made vary in order to obtain submission of several instances of similar jobs only differing for the value of the parameterized attributes

Analogously to DAGs and Collections, for job types whose submission results in the spawning of a set of jobs (i.e. Partitionable and Parametric jobs) the WMS (besides the ids of the single children) will provide the user with a collective identifier that will allow monitoring and control of the whole set of jobs through a single handle.

Important Note: Due to current limitations of the WMS the nodes of a DAG/Collection cannot be *Partitionable* or *Parametric* jobs.

As mentioned above the WMS supports different types of requests. The type of a submission request for the WMS is specified in the JDL through the "Type" attribute described in section 2.1.

2.1. TYPE

The *Type* attribute is a string representing the type of the request described by the JDL, e.g.

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

Possible values are:

- `Job`
- `DAG`
- `Collection`

The value for this attribute is case insensitive. If this attribute is not specified in the JDL description, the WMS will set it to `"Job"`.

- **Default:** `"Job"`

3. JOB ATTRIBUTES DESCRIPTION

This section reports the detailed description of the JDL attributes that can be specified for describing **Job** requests. A sub-section for each attribute is provided.

3.1. JOBTYPE

The *JobType* attribute is a string representing the type of the job described by the JDL, e.g.:

```
JobType = "Interactive";
```

Possible values are:

- `Normal`
- `Interactive`
- `MPICH`
- `Checkpointable`
- `Partitionable`
- `Parametric`

Note that a partitionable job must be also checkpointable.

This attribute only makes sense when the `Type` attribute equals to `"Job"`. The value for this attribute is case insensitive. If not specified in the JDL the WMS will set it to `"Normal"`

- **Default:** `"Normal"`

3.2. EXECUTABLE

The *Executable* attribute is a string representing the executable/command name.

The user can specify an executable that lies already on the remote CE WN and in this case the absolute path, possibly including environment variables referring to this file should be specified, e.g.:

```
Executable = "/usr/local/java/j2sdk1.4.0_01/bin/java";
```

or equivalently

```
Executable = "$JAVA_HOME/bin/java";
```

The other possibility is to provide either an executable available on the local file system or an executable located on a remote gridFTP server accessible by the user (HTTPS servers are also supported but this requires to have the GridSite **htcp** client command installed on the WN that is not

always the case currently). In both cases the executable file will be staged from the original location to the Computing Element WN. In both cases only the file name has to be specified as executable. The URI of the executable should be then listed in the *InputSandbox* attribute expression to make it be transferred. E.g. respectively:

```
Executable = "cms_sim.exe";
InputSandbox = {"file:///home/edguser/sim/cms_sim.exe", ..... };
```

Or

```
Executable = "cms_sim.exe";
InputSandbox = {"gsiftp://neo.datamat.it:5678/tmp/cms_sim.exe",
..... };
```

for backward compatibility with previous versions of the JDL (e.g. the one used in LCG) also description as follows:

```
Executable = "cms_sim.exe";
InputSandbox = {"/home/edguser/sim/cms_sim.exe", ..... };
```

are accepted and interpreted as in the first case, i.e. the executable file is available on the local file system.

It is important to remark that if the job needs for the execution some command line arguments, they have to be specified through the *Arguments* attribute. This attribute is mandatory.

- Mandatory: Yes
- **Default:** No

3.3. ARGUMENTS

The *Arguments* attribute is a string containing all the job command line arguments.

E.g. an executable *sum* that has to be started as:

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

is described by:

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

If you want to specify a quoted string inside the Arguments then you have to escape quotes with the \ character. E.g. when describing a job like:

```
$ grep -i "my name" *.txt
```

you will have to specify:

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

Analogously, if the job takes as argument a string containing a special character (e.g. the job is the *tail* command issued on a file whose name contains the ampersand character, say *file1&file2*), since on the shell line you would have to write:

```
$ tail -f file1\&file2
```

in the JDL you'll have to write:

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

i.e. a \ for each special character.

In general, special characters such as &, |, >, < are only allowed if specified inside a quoted string or preceded by triple \. The character ` cannot be specified in the JDL.

- Mandatory: No
- **Default:** No

3.4. STDINPUT

The *StdInput* attribute is a string representing the standard input of the job. This means that the job is run as follows:

```
$ job < <standard input file>
```

It can be an absolute path possibly including environment variables (wild cards are instead not allowed), i.e. it is already available on the CE, e.g.

```
StdInput = "/var/tpm/jobInput";
```

or just a file name, e.g.

```
StdInput = "myjobInput";
```

and this means that file needs to be made available on the WN where the job is run. Therefore the standard input file has to be added to the *InputSandbox* file list so that it will be copied to the WMS node and then downloaded on the WN by the WMS JobWrapper script. The same rules described for the *Executable* attribute apply to *StdInput*.

This attribute cannot be included in the JDL for Interactive jobs as the standard input for such jobs is either provided interactively by the user or read from a named pipe on the UI machine.

- Mandatory: No
- **Default:** No

3.5. STDOUTPUT

The *StdOutput* attribute is a string representing the file name where the standard output of the job is saved on the WN.

The user can specify either a file name or a relative path (with respect to the job working directory on the WN), e.g.:

```
StdOutput = "myjobOutput";
StdOutput = "event1/myjobOutput";
```

Wild cards are not allowed. The value specified for *StdError* can be the same as the one for *StdOutput* attribute: this means that the two standard streams of the job are saved in the same file.

To have this file staged back on the submitting machine the user has to list the file name also in the *OutputSandbox* attribute expression and use e.g the **glite-wms-job-output** command [R15].

Alternatively the user can choose to have this file staged automatically on a GridFTP server specifying a URI for that file in the *OutputSandboxDestURI* attribute expression. E.g.:

```
StdOutput = "myjobOutput";
OutputSandbox = { "myjobOutput", … };
OutputSandboxDestURI = {
            "gsiftp://fox.infn.it:5678/home/gftp/myjobOutput",
            …
            };
```

indicates that `myjobOutput` when the job has completed its execution has to be transferred on `gsiftp://fox.infn.it:5678` in the `/home/gftp` directory.

- Mandatory: No
- **Default:** No

3.6. STDERROR

The *StdError* attribute is a string representing the file name where the standard error of the job is saved on the WN.

The user can specify either a file name or a relative path (with respect to the job working directory on the WN), e.g.:

```
StdError = "myjobError";
StdError = "event1/myjobError";
```

Wild cards are not allowed. The value specified for *StdError* can be the same as the one for *StdOutput* attribute: this means that the two standard streams of the job are saved in the same file.

To have this file staged back on the submitting machine the user has to list the file name also in the *OutputSandbox* attribute expression and use e.g the **glite-wms-job-output** command [R15].

Alternatively the user can choose to have this file staged automatically on a GridFTP server specifying a URI for that file in the *OutputSandboxDestURI* attribute expression (see 3.5 for details).

The same rules as for the *StdOutput* apply to *StdError*.

- Mandatory: No
- **Default:** No

3.7. INPUTSANDBOX

The *InputSandbox* attribute is a string or a list of strings identifying the list of files on the UI local file system or on an accessible gridFTP server (HTTPS servers are also supported but this requires to have the GridSite **htcp** client command installed on the WN; this is not true in current CE standard configuration) needed by the job for running. These files are transferred from the UI to the WMS node and then downloaded on the CE WN where the job is run by the WMS JobWrapper script. If located on a gridFTP server, they are directly downloaded on the CE WN where the job is run by the WMS JobWrapper script.

Wildcards and environment variables are admitted in the specification of this attribute only if the submission takes place through a client able to resolve them locally before passing the JDL to the WMProxy service. Admitted wildcard patterns are the ones supported by the Linux glob function. One can remove the special meaning of the characters '? ', '*' and ' [' by preceding them by a backslash.

File names can be provided as URI on a gridFTP/HTTPS server, simple file names, absolute paths and relative paths with respect to the current UI working directory. The *InputSandbox* file list cannot contain two or more files having the same name (even if in different paths) as when transferred in the job's working directory on the WN they would overwrite each other.

LFNs cannot be given as *InputSandbox* file names; the *InputData* attribute (see 3.22.3) has to be used for that purpose.

This attribute is also used to accomplish executable and standard input staging to the WN of the CE where job execution takes place as explained above. The *InputSandbox* attribute meaning is strictly coupled with the value of the *InputSandboxBaseURI* defined in section 3.8 that specifies a common location on a gridFTP/HTTPS server where files in the *InputSandbox* not being already specified as URI are located.

Here below follows an example of *InputSandbox* setting:

```
InputSandbox = {
           "/tmp/ns.log",
           "mytest.exe",
           "myscript.sh",
           "data/event1.txt",
```

```
            "gsiftp://neo.datamat.it:5678/home/fpacini/cms_sim.exe ",
            "file:///tmp/myconf"
            };
InputSandboxBaseURI = "gsiftp://matrix.datamat.it:5432/tmp";
```

It means that:

- `/tmp/log` is located on the user local file system (accepted for backward compatibility)
- `mytest.exe,myscript.sh` and data/`event1.txt` are available on `gsiftp://matrix.datamat.it:5432` in the `/tmp` directory
- `/tmp/myconf` is located on the user local file system (explicitly specified using the `file://` prefix)

If the *InputSandboxBaseURI* is not specified than also `mytest.exe`, `myscript.sh` and `data/event1.txt` would be interpreted as located on the user local file system

- Mandatory: No
- **Default:** No

3.8. INPUTSANDBOXBASEURI

The *InputSandboxBaseURI* attribute is a string representing the URI on a gridFTP server (HTTPS servers are also supported but this requires to have the GridSite **htcp** client command installed on the WN; this is not true in current WN standard configuration) where the *InputSandbox* files that have been specified as simple file names and absolute/relative paths are available for being transferred on the WN before the job is started. E.g.

```
InputSandbox = {
                  …
                  "data/event1.txt",
                  …
                  };
InputSandboxBaseURI = "gsiftp://matrix.datamat.it:5432/tmp";
```

makes the WMS consider

```
"gsiftp://matrix.datamat.it:5432/tmp/data/event1.txt"
```

for the transfer on the WN.

- Mandatory: No
- **Default:** No

3.9. OUTPUTSANDBOX

The *OutputSandbox* attribute is a string or a list of strings identifying the list of files generated by the job on the WN at runtime, which the user wants to retrieve. The listed files are transferred at job completion on the WMS node and can be downloaded on the UI local file system by mean of e.g. the **glite-wms-job-output** command [R15]. Wildcards are not admitted in the specification of this attribute.

This attribute can be combined with the *OutputSandboxDestURI* (described in section 3.10) or the *OutputSandboxBaseDestURI* (described in section 3.11) to have, upon job completion, the output directly copied to specified locations running a gridFTP server (HTTPS servers are also supported but this requires to have the GridSite **htcp** client command installed on the WN; this is not true in current WN standard configuration). Note that output files managed in this way are not retrieved by the **glite-wms-job-output** command.

File names can be provided as simple file names or relative paths with respect to the current working directory on the executing WN. The *OutputSandbox* file list should not contain two or more files having the same name (even if in different paths), as when transitorily transferred on the WMS machine they would overwrite each other unless different destination URI are explicitly specified in the *OutputSandboxDestURI* attribute.

Here below is provided an example of the *OutputSandbox* attribute:

```
OutputSandbox = {
            "myjobOutput",
            "myjobError",
            "run1/event1",
            "run1/event2",
            };
```

This indicates that all the listed files (if actually generated by the job) will be available, after job completion, on the WMS node and will be available for retrieval using e.g. the **glite-wms-job-output** command.

- Mandatory: No
- **Default:** No

3.10. OUTPUTSANDBOXDESTURI

The *OutputSandboxDestURI* attribute is a string or a list of strings representing the URI(s) on a gridFTP/HTTPS server where the files listed in the *OutputSandbox* attribute have to be transferred at job completion.

The *OutputSandboxDestURI* list contains for each of the files specified in the *OutputSandbox* list the URI (including the file name) where it has to be transferred at job completion. E.g.

```
OutputSandbox = {
            "myjobOutput",
            "run1/event1",
            "myjobError"
            };
OutputSandboxDestURI = {
            "gsiftp://matrix.datamat.it:5432/tmp/myjobOutput ",
            "gsiftp://grid003.ct.infn.it:6789/home/cms/event1",
            "myjobError"
            };
```

makes the WMS transfer respectively

- *myjobOutput* on *matrix.datamat.it* in the directory */tmp*
- *event1* on *grid003.ct.infn.it* in the directory */home/cms*
- *myjobError* on the WMS node

The *OutputSandboxDestURI* list must have the same cardinality as the *OutputSandbox* list, otherwise the JDL will be considered as invalid. Note that the file name specified in the *OutputSandbox* can be different from the corresponding destination file name specified in the *OutputSandboxBaseDestURI.*

The *OutputSandboxDestURI* attribute and the *OutputSandboxBaseDestURI* described in section 3.11 cannot be specified together in the same JDL.

If neither *OutputSandboxDestURI* nor *OutputSandboxBaseDestURI* are specified in the JDL then all the files listed in the *OutputSandbox* (if actually generated by the job) will be available, after job completion, on the WMS node and will be available for retrieval using e.g. the **glite-wms-job-output** command [R15].

- Mandatory: No
- **Default:** the job directory on the WMS node

3.11. OUTPUTSANDBOXBASEDESTURI

The *OutputSandboxBaseDestURI* attribute is a string representing the base URI on a gridFTP/HTTPS server, i.e., a directory on the server, where the files listed in the *OutputSandbox* attribute have to be transferred at job completion (HTTPS servers are also supported but this requires to have the GridSite

htcp client command installed on the WN; this is not true in current WN standard configuration). All the *OutputSandbox* files are transferred to the location specified by the URI with the same names (only names in a flat directory) as the ones specified in the *OutputSandbox*. E.g.:

```
OutputSandbox = {
            "myjobOutput",
            "run1/event1",
            };
OutputSandboxBaseDestURI = "gsiftp://matrix.datamat.it:5432/tmp";
```

makes the WMS transfer both files in the */tmp* directory of the gridFTP server *matrix.datamat.it* (note that *event1* will go in */tmp* and not in */tmp/run1*).

The *OutputSandboxBaseDestURI* attribute and the *OutputSandboxDestURI* described in section 3.10 cannot be specified together in the same JDL.

If neither *OutputSandboxDestURI* nor *OutputSandboxBaseDestURI* are specified in the JDL then all the files listed in the *OutputSandbox* (if actually generated by the job) will be available, after job completion, on the WMS node and will be available for retrieval using e.g. the **glite-wms-job-output** command [R15].

- Mandatory: No
- **Default:** the job directory on the WMS node

3.12. ALLOWZIPPEDISB

The *AllowZippedISB* attribute is a boolean attribute. When set to true, i.e.

```
AllowZippedISB = true;
```

makes the WMProxy client commands [R15] archive and compress all job input sandbox files into a single tar, gzipped file that is then transferred to the WMS.

Although the file compression rate could result in some cases negligible, this approach can be particularly useful when the job sandbox is composed by a large number of files and the cost of the authentication steps required by each single transfer call can have a big impact on the total amount of time for uploading the files.

This attribute is not mandatory. If not specified in the JDL it is assumed to be set to *false*.

A default value for this attribute can be specified in the WMProxy client [R15] configuration file (*AllowZippedISB* parameter in glite_wms.conf)

Note that the *AllowZippedISB* is only used on the client side by the WMProxy **glite-wms-job-submit** command [R15] in order to know if it has to generate the tarball containing the job's input sandbox

files and fill the *ZippedISB* attribute (see 3.13) accordingly before sending the request to the WMProxy service. It is not used at all by the WMProxy service.

- Mandatory: No
- **Default:** false (unless otherwise specified in the WMProxy client configuration)

3.13. ZIPPEDISB

The *ZippedISB* attribute is a string or a list of strings containing the file name of the compressed (gzip-ed) tarball containing the input sandbox files for the job, e.g.:

```
ZippedISB = "BossArchive_1_2_1.tgz";
```

If the *ZippedISB* attribute is set in the incoming JDL, the WMProxy service takes the specified archive (assumed to be located in the job's input sandbox area on the WMS node) and explodes it in the right locations.

Note that this attribute MUST NOT be set when the submission is done through the WMProxy client commands [R15] and the *AllowZippedISB* attribute is set to *true* as upon that setting the **glite-wms-job-submit** command takes care of the creation of the compressed archive, of the naming of the resulting tarball and of the appropriate setting of the *ZippedISB* attribute in the JDL.

Please refer to the WMProxy Wiki pages [R17] for details about the creation of the tarball and the JDL setting when submitting through the WMProxy API (see [R15] for references) or through clients generated from the service WSDL.

- Mandatory: No
- **Default:** No

3.14. EXPIRYTIME

A job for which no compatible CEs have been found during the matchmaking phase is hold in the WMS Task Queue (see [R13]) for a certain time (currently it is 1 day from job submission) so that it can be subjected again to matchmaking from time to time until a compatible CE is found. If no match is found after 1 day the job is aborted.

The *ExpiryTime* attribute is an integer representing the date and time (in seconds since epoch) until the job has to be considered valid by the WMS. E.g.

```
ExpiryTime = 1112339655;
```

tells that the job has to be considered valid for matchmaking until "2005-04-01 07:14:15".

The **glite-wms-job-submit** command [R15] provides options (--valid, --to) to specify the value for the expiry time in a user-friendly format.

If the specified *ExpiryTime* exceeds one day from job submission then it is not taken into account by the WMS that will eventually hold the job in the TQ only for 1 day.

Note that this attribute is not taken into account for sub-jobs of a DAG/Collection/Parametric-job that (in gLite 3.0.0) are not yet hold in the TQ.

- Mandatory: No
- **Default:** one day after submission time.

3.15. ENVIRONMENT

The *Environment* attribute is a list of string representing environment settings that have to be performed on the execution machine and are needed by the job to run properly. The JobWrapper on the Worker Node performs these settings just before the job is started. Each item of the list is an equality "VAR_NAME=VAR_VALUE". E.g.:

```
Environment  = {"JOB_LOG_FILE=/tmp/myjob.log",
                "ORACLE_SID=edg_rdbms_1",
                "JAVABIN=/usr/local/java"};
```

- Mandatory: No
- **Default:** No

3.16. PERUSALFILEENABLE

The *PerusalFileEnable* attribute is a Boolean attribute that allows enabling the job file perusal support in the WMS. When this attribute is set to true, i.e.

```
PerusalFileEnable = true;
```

the user can inspect, while the job is running, the files generated by the job on the WN. This is achieved by uploading on a location on the WMS node (the WMS JobWrapper does it), at regular time intervals, chunks of the files generated by the job on the WN.

This can be done through the WMProxy job file perusal specific operations or through the **glite-wms-job-perusal** command [R15].

The *PerusalFileEnable* attribute is not mandatory. If not specified in the JDL it is assumed to be set to false.

A default value for this attribute can be specified in the WMProxy client [R15] configuration file (*PerusalFileEnable* parameter in glite_wms.conf)

- Mandatory: No
- **Default:** false (unless otherwise specified in the WMProxy client configuration)

3.17. PERUSALTIMEINTERVAL

The *PerusalTimeInterval* attribute is a positive integer representing the difference in seconds between two subsequent saving (and upload on the WMS node) of the job files generated by the job on the WN. Specifying e.g.

```
PerusalTimeInterval = 10;
```

makes the WMS JobWrapper save the job files specified through the **--set** option of the **glite-wms-job-perusal** command [R15] each 10 seconds and upload them on the WMS node so that they can be inspected by the user.

The actual number of seconds taken by the WMS is represented by the maximum value between *PerusalTimeInterval* itself and the value of the *MinPerusalTimeInterval* parameter in the WM configuration file (default for *MinPerusalTimeInterval* is 5).

- Mandatory: No
- **Default:** *MinPerusalTimeInterval* parameter in the WMS configuration file (= 5)

3.18. PERUSALFILESDESTURI

The *PerusalFilesDestURI* attribute is a string representing the URI of the location on a gridFTP or HTTPS server (HTTPS servers are also supported but this requires to have the GridSite **htcp** command installed on the WN; this is not true in current WN standard configuration) where the chunks of files generated by the running job on the WN and specified through the **--set** option of the **glite-wms-job-perusal** command [R15] have to be copied. E.g.

```
PerusalFilesDestURI = "gsiftp://ghemon.cnaf.infn.it/home/glite/peek";
```

The *PerusalFilesDestURI* attribute is not mandatory. If not specified in the JDL it is set by the WMS to a job specific location on the WMS node and the uploaded files can be accessed by users through the **--get** option of the **glite-wms-job-perusal** command.

- Mandatory: No
- **Default:** job file perusal directory on the WMS node

3.19. INPUTDATA

Note that this attribute has been only kept for backward compatibility and will be soon deprecated. Use *DataRequirements* attribute 3.22 instead.

The *InputData* attribute is a string or a list of strings representing Logical File Names (LFN), Grid Unique Identifiers (GUID), Logical Dataset (LDS) and/or generic queries. All of them are used by the WMS to query the related Data Catalog for getting back a list of Physical File names (PFN) that are needed by the job as input for processing. LFNs and LDSs have the form of a URI and a GUID is a 40

characters UUID (http://www.ietf.org/internet-drafts/draft-mealling-uuid-urn-05.txt). The format of generic queries is experiment/catalog specific.

For more information please refer to [R3]. All these data are stored in SEs and published in data catalogs.

As mentioned above the listed items are used by the WMS at matchmaking time to get the list of corresponding PFNs and then to find the CE having the greatest number of physical files on a close SE in order to schedule the job to run there.

Listed names have to be prefixed with *"lfn:"*, *"guid:"*, *"lds"* and *"query:"* to indicate that they are respectively LFNs, GUIDs, LDSs and generic queries. E.g.:

```
InputData = {
          "lfn:/EO/test.file" ,
          "si-lfn:/dm/test.file",
          "lds:cms.test.file",
          "guid:135b7b23-4a6a-11d7-87e7-9d101f8c8b70",
          "query:select_my_DC1_files"
          "si-guid:456sde21-wer56-11d7-87e7-9d1--g8c8f33"
          };
```

As the data could be published by different types of catalogs and the WMS currently supports three of them (i.e. the RLS, the StorageIndex and the DLI), the rule for choosing the catalog type to contact for resolving the file names according to the prefix is as follows:

- *lfn:*, *guid:* StorageIndex if the *StorageIndex* attribute (see 3.20) is also specified in the JDL. RLS otherwise and DLI upon failure on the RLS.
- *si-lfn:*, *si-guid:* StorageIndex
- *lds:*, *query:* DLI

Wildcards are not admitted when specifying this attribute.

- Mandatory: No
- **Default:** No

3.20. STORAGEINDEX

Note that this attribute has been only kept for backward compatibility and will be soon deprecated. Use *DataRequirements* attribute 3.22 instead.

The *StorageIndex* attribute is a string representing the endpoint URI of the StorageIndex service to contact for resolving the file names specified in the InputData attribute list, e.g.

```
StorageIndex = "https://data.glite.org:9443/StorageIndex";
```

In case it is not specified and the InputData prefixes are *si-lfn:* or *si-guid:* then the VO default StorageIndex is considered and the endpoint URI is taken either from the configuration or through service discovery.

- Mandatory: No
- **Default:** No

3.21. DATACATALOG

Note that this attribute has been only kept for backward compatibility and will be soon deprecated. Use *DataRequirements* attribute 3.22 instead.

The *DataCatalog* attribute is a string representing the endpoint URI of the RLS or DLI service to contact for resolving the file names specified in the InputData attribute list, e.g.

```
DataCatalog = "http://data.example.org/CMSDataSetCatalog";
```

The specification of this attribute forces the WMS to resolve all LFNs and GUIDs in the Inputdata list that are not prefixed by *si-lfn:* or *si-guid:* using a "RLS" catalog and a "DLI" catalog upon failure.

- Mandatory: No
- **Default:** No

3.22. DATAREQUIREMENTS

The *DataRequirements* attribute is a list of classads representing the data requirements for a job. Each classad in the list contains three attributes (described in detail in the following sections 3.22.3, 3.22.1 and 3.22.2):

- *InputData*
- *DataCatalogType*
- *DataCatalog*

that represent respectively the list of input data needed by the job, the type of data catalog that has to be targeted to resolve logical names to physical names and lastly the URI of the data catalog if this is not the VO default one (endpoint known through service discovery or configuration).

The form of this attribute allows users to target experiment-specific catalogs for their jobs and to mix different input data types supported by different data catalogs in the same job description.

Note that it is possible to specify more than once the same catalog type e.g. for using the VO default one for some input data and a specific one for some other data.

Here below is reported an example of specification of the *DataRequirements* attribute:

```
DataRequirements = {
        [
          DataCatalogType = "DLI";
          DataCatalog = "https://cms.org:8877/dli";
```

```
          InputData = {"lfn:/my/test/data1",
                       "guid:44rr44rr77hh77kkaa3",
                       "lds:my.test.dataset",
                       "query:my_query"};
        ],
       [
          DataCatalogType = "SI";
          DataCatalog = "https://glite.org:9443/StorageIndex";
          InputData = {"lfn:/eo/test.file", "guid:ddffrg5451"};
        ],
       [
          DataCatalogType = "RLS";
          DataCatalog = "https://eu-datagrid.org/RLS";
          InputData = {"lfn:/atlas/test.file", "guid:ggrgrg5656"};
        ],
       [
          DataCatalogType = "RLS";
          InputData = {"lfn:/myvo/test.file",
"guid:adbdefgilm1234"};
        ],

        ....
    };
```

The following sub-sections report the description of the mentioned attributes:

3.22.1. DataCatalogType

The *DataCatalogType* attribute is a string representing the type of the data catalog to be targeted for resolving the input data logical names. Possible values, i.e. the data catalog interfaces currently supported by the WMS are:

- RLS LCG Replica Location Service (lfn, guid)
- SI gLite Storage Index (lfn, guid)
- DLI LCG Data Location Interface (lfn, guid, lds, query)

between parenthesis the corresponding supported data types.

Of course the list of values will be extended as soon as the WMS will support new interfaces.

- Mandatory: yes
- **Default:** No

3.22.2. DataCatalog

The *DataCatalog* attribute is a string representing the data catalog service endpoint URI to contact for resolving the file names specified in the InputData attribute list, e.g.

```
DataCatalog = "http://data.example.org/CMSDataSetCatalog";
```

It should be specified only if it is different from the VO default one.

- Mandatory: No
- **Default:** taken from WMS configuration or through service discovery

3.22.3. InputData

The *InputData* attribute is a string or a list of strings representing Logical File Names (LFN), Grid Unique ID-entifiers (GUID), Logical Dataset (LDS) and/or generic queries. All of them are used by the WMS to query the related Data Catalog for getting back a list of Physical File names (PFN) that are needed by the job as input for processing. LFNs and LDSs have the form of a URI and a GUID is a 40 characters UUID (http://www.ietf.org/internet-drafts/draft-mealling-uuid-urn-05.txt). The format of generic queries is experiment/catalog specific.

For more information please refer to [R3]. All these data are stored in SEs and published in data catalogs.

As mentioned above the listed items are used by the WMS at matchmaking time to get the list of corresponding PFNs and then to find the CE from which the specified files can be better accessed in order to schedule the job to run there.

Listed names have to be prefixed with *"lfn:"*, *"guid:"*, *"lds"* and *"query:"* to indicate that they are respectively LFNs, GUIDs, LDSs and generic queries. E.g.:

```
InputData = {
            "lfn:/EO.test.file" ,
            "lds:cms.test.file",
            "guid:135b7b23-4a6a-11d7-87e7-9d101f8c8b70",
            "query:select_my_DC1_files"
            };
```

As not all data catalogs support all types of input data, the specification of this attribute has to be cross-checked with the one of the *DataCatalogType* attribute described in section 3.22.1.

Wildcards are not admitted when specifying this attribute.

- Mandatory: Yes
- **Default:** No

3.23. DATAACCESSPROTOCOL

The *DataAccessProtocol* attribute is a string or a list of strings representing the protocol or the list of protocols that the application is able to "speak" for accessing files listed in *InputData* on a given SE. The RB matches indeed this attribute with the protocols supported by the SE, as published in the IS.

This is an example of the *DataAccessProtocol* attribute:

```
DataAccessProtocol = {
                "https",
                "gsiftp"
                };
```

There is no restriction to the protocol names that can be specified through this attribute. Whether a given protocol is supported or not depends on the SE implementation. The list of SE access protocols supported in the Glue schema can be considered as the authoritative source for this information. Glue 1.2 supports *gsiftp*, *nfs*, *afs*, *rfio*, *gsirfio*, *dcap*, *gsidcap*, *root*, *https*, *other*.

- Mandatory: Yes (only if *DataRequirements* or *InputData* has been specified)
- **Default:** No

3.24. OUTPUTSE

The *OutputSE* attribute is a string representing the URI of the Storage Element where the user wants to store the output data. Once specified, this attribute is used by the RB to find a CE being "close" to this SE and schedule the job there. E.g.:

```
OutputSE = "grid001.cnaf.infn.it";
```

- Mandatory: No
- Default: No

3.25. OUTPUTDATA (NOT YET SUPPORTED)

This attribute allows the user to ask for the automatic upload and registration of datasets produced by the job on the WN. Through this attribute it is possible to indicate for each output file the LFN to be used for registration and the SE on which the file has to be uploaded. The *OutputData* attribute is not mandatory.

OutputData is a list of classads where each classad contains the following three attributes:

- *OutputFile*
- *StorageElement*
- *LogicalFileName*

These three attributes are only admitted if members of one of the classads composing *OutputData*. They cannot be specified independently in the job JDL.

Here below follows an example of the *OutputData* attribute:

```
OutputData = {
                    [
                                OutputFile = "dataset_1.out ";
                                LogicalFileName = "lfn:/test/result1";
                     ],
                    [
                                OutputFile = "dataset_2.out ";
                                StorageElement = "se001.cnaf.infn.it";
                     ],
                    ]
                                OutputFile = "cms/dataset_3.out";
                                StorageElement = "se012.to.infn.it";
                                LogicalFileName = "lfn:/cms/outfile1";
                     ],
                    [
                                OutputFile = "dataset_4.out ";
                     ]
            };
```

If the attribute *OutputData* is found in the JDL then the *JobWrapper* at the end of the job calls the DM service that copies the file from the WN onto the specified SE and registers it with the given LFN. If the specified LFN is already in use, the DM service registers the file with a newly generated identifier GUID (Grid Unique Identifier).

During this process the *JobWrapper* creates a file (named "*DSUpload_<unique_jobid_string>.out*") with the results of the operation that is put automatically in the *OutputSandbox* attribute list by the UI and can then be retrieved by the user.

- Mandatory: No
- **Default:** No

The following sub-sections report the description of the mentioned attributes:

3.25.1. OutputFile (not yet supported)

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.

Wildcards are not admitted in the specification of this attribute. File names can be provided as simple file names, absolute paths or relative paths with respect to the current working directory.

- Mandatory: Yes (only if *OutputData* has been specified)
- **Default:** No

3.25.2. StorageElement (not yet supported)

This is a string representing the URI of the Storage Element where the output file specified in the corresponding *OutputFile* attribute has to be uploaded by the WMS.

It is worth noting that this attribute is not taken into account by the RB for the matchmaking, so the job could have run on a CE that is not close to the specified SE. Due to this it is suggested (unless the user has particular needs) either to omit the StorageElement specification so that the close SEs are automatically taken into account for the datasets upload or to keep its specification aligned with the one for the *OutputSE* attribute (see section 3.24) that is the one considered during the matchmaking.

- Mandatory: No
- **Default:** a SE close to the CE where the job is submitted

3.25.3. LogicalFileName (not yet supported)

This is a string representing the logical file name (LFN) the user wants to associate to the output file when registering it to the Replica Catalogue. The specified name has to be prefixed by "*lfn:*" (lowercase).

If this attribute is not specified then the corresponding output file is registered with a GUID that is assigned automatically by the WP2 services (RM).

- Mandatory: No
- **Default:** No (a GUID is assigned by DM services)

3.26. VIRTUALORGANISATION

The *VirtualOrganisation* attribute is a string representing the name of the VO the submitting user is currently working for.

The value for this attribute has to match with the VO specified within the credentials issued by VOMS or with the possible options provided by the WMProxy client commands [R15] (e.g. if the **glite-wms-job-submit** command [R15] is issued with the *--vo* option, then the value of this attribute is overwritten with the VO name specified on the command line). This is an example for this attribute:

```
VirtualOrganisation = "atlas";
```

The value for this attribute is case insensitive.

- Mandatory: Yes
- **Default:** **No**

3.27. RETRYCOUNT

The *RetryCount* attribute is an integer representing the maximum number of *deep* job re-submissions to be done in case of failure due to some grid component (i.e. not to the job itself).

Job resubmission is defined "*deep*" when the user's job has started running on the WN and then the job itself or the WMS JobWrapper has failed. It is instead defined "*shallow*" when the WMS JobWrapper has failed before starting the actual user's job.

RetryCount has to be a number equal or greater than 0 and the actual number of submission retries for a job is represented by the minimum value between *RetryCount* itself and the value of the *MaxRetryCount* parameter in the WM configuration file (default for *MaxRetryCount* is 10).

Hereafter follows an example for this attribute:

```
RetryCount = 3;
```

For example for disabling the deep job re-submission mechanism it suffices specifying:

```
RetryCount = 0;
```

Note that the *RetryCount* attribute is not taken into account for nodes of a DAG/Collection/Parametric job as the retry mechanism is not yet supported by WMS for those request types.

- Mandatory: No
- **Default:** *MaxRetryCount* parameter in the WMS configuration file, i.e. 10, if no default is applied by the client

3.28. SHALLOWRETRYCOUNT

The *ShallowRetryCount* attribute is an integer representing the maximum number of *shallow* job re-submissions to be done in case of failure due to some grid component (i.e. not to the job itself).

Job resubmission is defined "*deep*" when the user's job has started running on the WN and then the job itself or the WMS JobWrapper has failed. It is instead defined "*shallow*" when the WMS JobWrapper has failed before starting the actual user's job.

ShallowRetryCount has to be a number equal or greater than 0 and the actual number of shallow submission retries for a job is represented by the minimum value between *ShallowRetryCount* itself and the value of the *MaxShallowRetryCount* parameter in the WM configuration file (default for *MaxShallowRetryCount* is 10).

Hereafter follows an example for this attribute:

```
ShallowRetryCount = 3;
```

For example for disabling the shallow job re-submission mechanism it suffices specifying:

```
ShallowRetryCount = 0;
```

Note that the shallow retry count is reset to zero each time a deep resubmission of the job is done.

Note that the *ShallowRetryCount* attribute is not taken into account for nodes of a DAG/Collection/Parametric job as the retry mechanism is not yet supported by WMS for those request types.

- Mandatory: No
- **Default:** *MaxShallowRetryCount* parameter in the WMS configuration file, i.e. 10, if no default is applied by the client

3.29. LBADDRESS

The *LBAddress* attribute is a string representing the address (*<host>[:<port>]*) of the LB server where the WMS components have to store job information. E.g.:

```
LBAddress = "tigerman.cnaf.infn.it:9000";
```

If this attribute is not specified in the JDL the WMS gets the LB server address either querying the SD service or from the WMS configuration. If the port number is not specified, the WMS takes the default port, 9000.

- Mandatory: No

– **Default:** taken from WMS configuration

3.30. MYPROXYSERVER

The *MyProxyServer* attribute specifies the hostname of a MyProxy server where the user has registered her/his long-term proxy certificate.

A MyProxy server can be used to store a long-lived user certificate which can be used by the WMS to renew the lifetime of a standard user certificate proxy (usually valid only for 12 hours). Long-running jobs may run into this limit and fail due to expiration of the user proxy. To avoid this from happening, the user can store a certificate on a MyProxy server by issuing

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

which will create a proxy valid for the specified number of hours on the given server (See http://grid.ncsa.uiuc.edu/myproxy for more details about MyProxy). The very same hostname can be specified in the JDL MyProxyServer attribute, which will trigger the WMS proxy renewal process for this user's jobs

An example of the JDL setting is:

```
MyProxyServer = "skurut.cesnet.cz";
```

If the port number is not specified, the WMS takes the default port, 7512.

In order to make sure that proxy renewal is disabled for a given job and no default MYProxy Server is taken from the configuration of your client tools, just set the *MyProxyServer* attribute to the empty string, i.e,

```
MyProxyServer = "";
```

– Mandatory: No
– **Default:** No

3.31. HLRLOCATION

The *HLRLocation* attribute is a string representing the user Home Location Register address in the format

```
<host fqdn>:<port>:[<X509contact string>]
```

HLR is the service responsible for managing the economic transactions and the accounts of user and resources. The presence of the *HLRLocation* attribute in the JDL enables accounting for the job, i.e.

- on the CE, while the job runs, a sensor monitors the resource usage and when the job is done those data (usage records) are sent to the HLR specified through the *HLRLocation* attribute
- the HLR computes the job cost according to the usage records and to the resource price and then debits the user account

An example of the JDL setting is provided hereafter:

```
HLRLocation =
"lilith.to.infn.it:56568:/O=CESNET/O=Masaryk University/CN=Miroslav Ruda"
```

- Mandatory: No
- **Default:** No

3.32. JOBPROVENANCE

The *JobProvenance* attribute is a string representing the endpoint URI of the Job Provenance service where data about the job have to be stored, e.g.

```
JobProvenance = "https://lindir.ics.muni.cz:10001";
```

The specification of this attribute in the job description makes the WMS feed the Job Provenance service with the job sandbox files.

This attribute is not mandatory. A default value for this attribute can be specified in the WMProxy client [R15] configuration file (*JobProvenance* parameter in glite_wms.conf).

In order to make sure that job provenance is disabled for a given job and no default JobProvenance Server is taken from the configuration of your client tools, just set the *JobProvenance* attribute to the empty string, i.e,

```
JobProvenance = "";
```

- Mandatory: No
- **Default:** No (unless specified in the WMProxy client configuration)

3.33. NODENUMBER

The *NodeNumber* attribute is an integer greater than 1 specifying the number of CPUs needed for a MPI job. This attribute is only allowed if the job type is MPICH (see 3.1). An example of the JDL setting is:

```
NodeNumber = 5;
```

The RB uses this attribute during the matchmaking for selecting those CE having a number of CPUs equal or greater than the one specified in *NodeNumber*.

- Mandatory: Yes (if the job type is MPICH)
- **Default:** No

3.34. JOBSTEPS

The *JobSteps* attribute can be either an integer representing the number of steps for a checkpointable or partitionable job, e.g.:

```
JobSteps = 100000;
```

or a list of strings representing labels associated to the steps of a checkpointable or partitionable job, e.g:

```
JobSteps = {"rawdata", "d0", "d1", "d2", "gomos"};
```

As described in [R11] a checkpointable application can be seen as "composed" by a set of sequential steps, where a step can represent for example the processing of a file, the analysis of an HEP event, etc. The various steps can be represented by a *main stepper* set of iterations and it is usually worth to save the state of the job after each step.
The content of the *main stepper* can be defined through the JDL attribute *JobSteps*. Note that this attribute is used by the job (linked against the chekpointing library) at run time; it is not used by the WMS. See [R4] and [R11] for details.

Please note moreover that the number of checkpoint states saved by the job does not necessarily correspond to the number of steps listed in the JobSteps attribute. This is indeed a choiche made by the developer of the application that is being submitted according to the meaning and relevance of each step for the application itself.

This attribute can only be set for checkpointable or partitionable jobs.
If the *JobStep* attribute is specified within the *JobState* classad 3.36 it should not be specified at this level. In case it is specified in both places, only the value within the *JobState* classad is taken into account by the WMS.

- Mandatory: Yes (for checkpointable and partitionable jobs)
- **Default:** No

3.35. CURRENTSTEP

The CurrentStep attribute is an integer equal or greater than 0 indicating the step number to be taken as the initial one when submitting a checkpointable or partitionable job (see [R4] and [R11] for details). If *JobSteps* is a list of labels then *CurrentStep* indicates the position of the label in the list. E.g.

```
CurrentStep = 2;
```

for the second example of previous section 3.34 would indicate the step labelled *"d1"* as the first step.

In other words the CurrentStep is used by the job at run time to initialise the iterator defined by the JobSteps attribute described in 3.34. Note that the value of this attribute is used by the job (linked against the chekpointing library) at run time; it is not used by the WMS. See [R4] and [R11] for details.

This attribute can only be set for checkpointable or partitionable jobs. If not provided by the user it is assumed to be 0.

If the *CurrentStep* attribute is specified within the *JobState* classad 3.36 it should not be specified at this level. In case it is specified in both places, only the value within the *JobState* classad is taken into account by the WMS.

- Mandatory: No
- **Default:** 0

3.36. JOBSTATE

The *JobState* attribute is a classad attribute representing a job checkpoint state that has to be used for submitting a checkpointable job so that it will start its execution from the given state and not from the beginning. E.g.:

```
JobState = [
     JobSteps = 1000;
     CurrentStep = 350;
      UserData = [
          // user defined value pairs
          DumpPath = "gsiftp://ibm139.infn.it/tmp/sav_350.dmp";
                ];
          ]
```

The *JobState* usually represents an intermediate checkpoint state saved either by a previous run of the same job or by another job. Checkpoint states can be retrieved from the LB using e.g. the **glite-job-get-chkpt** command [R16].

The *JobState* classad attributes are described in the following sections

3.36.1. JobSteps

See 3.34.

If the *JobSteps* attribute is specified within the *JobState* classad it should not be specified in the upper level classad (i.e. the job description). In case it is specified in both places, only the value within the *JobState* classad is taken into account by the WMS.

- Mandatory: Yes (for checkpointable jobs)
- **Default:** No

3.36.2. CurrentStep

See 3.35.

If the *CurrentStep* attribute is specified within the *JobState* classad it should not be specified in the upper level classad (i.e. the job description). In case it is specified in both places, only the value within the *JobState* classad is taken into account by the WMS.

- Mandatory: Mandatory: Yes (for checkpointable jobs)
- **Default:** No

3.36.3. UserData

The *UserData* attribute is a classad attribute containing user defined value,pairs providing information that are needed by the job itself at runtime (i.e. the values of the attributes within the UserData classad are not used by the WMS). They could e.g. represent the URI of a file containing the saved context of a previous run of the job, the value of a certain parameter etc.

If not specified it is considered as an empty classad, i.e. no usr data have to be passed to the job.

- Mandatory: No
- **Default:** empty classad [ ]

3.37. LISTENERPORT

The *ListenerPort* attribute is an integer (>0) that represents the port on which the condor *grid_console_shadow* process started by the client listens for the job standard streams. E.g.:

```
ListenerPort = 44000;
```

This attribute can only be included in the JDL for interactive jobs (see 3.1).

If this attribute is not included in the JDL then the **glite-wms-job-submit** command [R15] sets the listener port to the one automatically assigned by the OS.

- Mandatory: No
- **Default:** No (the port number assigned dynamically by the OS)

3.38. LISTENERHOST

The *ListenerHost* attribute is a string that represents the host name on which the condor *grid_console_shadow* process (started by the client) listens for the job standard streams coming from the WN. E.g.:

```
ListenerHost = "trinity.datamat.it";
```

It can be useful e.g. when job submission and interactive display of the job streams take place on different machines.

This attribute can only be included in the JDL for interactive jobs (see 3.1).

If this attribute is not included in the JDL then the **glite-wms-job-submit** command [R15] sets the listener host to the job submission host name.

- Mandatory: No
- **Default:** the job submission host name

3.39. LISTENERPIPENAME

The *ListenerPipeName* attribute is a string representing the name (complete path) of the pipes on which the condor *grid_console_shadow* process (started by the client) read/writes the job standard streams. E.g.:

```
ListenerPipeName = "/tmp/my-job-pipe";
```

makes the *grid_console_shadow* process read the stdin of the job from */tmp/my-job-pipe.in* and write the stdout of the job on */tmp/my-job-pipe.out.*

This attribute can only be included in the JDL for interactive jobs (see 3.1).

If this attribute is not included in the JDL then the **glite-wms-job-submit** command [R15] sets the listener pipe name to */tmp/listener-<jobid unique string>*.

- Mandatory: No
- **Default:** /tmp/listener-<jobid unique string>

3.40. REQUIREMENTS

The *Requirements* attribute is a Boolean ClassAd expression that uses C-like operators. It represents job requirements on resources. The Requirements expression can contain attributes that describe the

CE in the IS which are prefixed with "***other.***". This notation indicates at matchmaking time that for a given classAd (the job's JDL) the expression has to be evaluated in the context of the counterpart classAd (the CE's JDL). See [R7] for details.

All these attributes are reported in the Glue Schema for the CE (see [R6]).

To have a job scheduled to run on a given CE, this Requirements expression must evaluate to true on the given CE. The evaluation of this expression is performed by the RB during the match making phase. This is an example of requirements expression:

```
Requirements = other.GlueCEInfoLRMSType == "PBS" &&
               other.GlueCEInfoTotalCPUs > 2     &&
      Member("IDL1.7",other.GlueHostApplicationSoftwareRunTimeEnvironment);
```

The above expression requires a CE whose local resource manager is PBS, having at least 2 CPUs and the IDL software version 1.7 already installed. The classAd Member function described in [R7] returns true if the tag "IDL1.7" is a member of the other.GlueHostApplicationSoftwareRunTimeEnvironment list.

The *Requirements* attribute is mandatory in the JDL as it is a mandatory parameter for the matchmaking library.

If this attribute is not included in the JDL the **glite-wms-job-submit** command [R15] sets it to:

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

as "Production" is the nominal working state for a CE. This automatic setting is however not insured if submission is performed using client tools other than **glite-wms-job-submit**.

Requirements can be set to *true* to express no constraints on the resources.

- Mandatory: Yes
- **Default:** `No`

3.41. RANK

The *Rank* attribute is a ClassAd Floating-Point expression that states how to rank CEs that have already met the *Requirements* expression. Essentially, rank expresses a preference. A higher numeric value equals a better rank. The WMS will submit the job to the CE with the highest rank.

The *Rank* expression can contain attributes that describe the CE in the IS which are prefixed with "***other.***". This notation indicates at matchmaking time that for a given classAd (the job's JDL) the expression has to be evaluated in the context of the counterpart classAd (the CE's JDL). See [R7] for details.

All these attributes are reported in the Glue Schema for the CE (see [R6]).

The evaluation of the rank expression is performed by the RB during the match making phase. This is an example of Rank expression:

```
Rank = other.GlueCEPolicyMaxRunningJobs - other.GlueCEStateRunningJobs;
```

With the above Rank, the preferred CEs are the ones having the greatest number of free slots available for running jobs.

The *Rank* attribute is mandatory in the JDL as it is a mandatory parameter for the matchmaking library.

If this attribute is not included in the JDL the **glite-wms-job-submit** command [R15] sets it to:

```
Rank = -other.GlueCEStateEstimatedResponseTime;
```

that expresses a preference for those CEs having a shorter estimated time for traversing the local batch system queue. This automatic setting is however not insured if submission is performed using client tools other than **glite-wms-job-submit**.

Setting the Rank expression to a constant value (e.g. `Rank = 1;`) makes all CEs matching job's requirements have the same ranking. In this case the CE for submiiting the job is chose randomly by the WMS.

- Mandatory: Yes
- **Default:** `No`

3.42. FUZZYRANK

The *FuzzyRank* attribute is a Boolean attribute that enables fuzzyness in the ranking computation. In other words if this attribute is set to `true`, it forces the matchmaking algorithm to adopt a stochastic selection criteria while searching for the best matching CE. E.g. specifying:

```
FuzzyRank = true;
```

in the submitted JDL, the rank values associated to each matching CE represent the probability that each CE has to be selected as the best matching one. The selection probability is higher for higher rank values.

- Mandatory: No
- **Default:** `FALSE`

3.43. USERTAGS

The *UserTags* attribute is a classad attribute that allows the user to specify user-defined key, value pairs (where the value must be a string) that are logged at submission time to the LB and are associated to the job in the LB database. The specified user tags can be then used to build conditions when querying the LB for the status of submitted jobs (see e.g. the --user-tag option of the **glite-job-status** command [R16]). The value of a user tag can be modified or a new key, value pair can be added after the job has been submitted using the LB producer API (see method edg_wll_LogUserTag) that is provided by the *org.glite.lb.client* component.

This is an example of user tags setting in the JDL:

```
UserTags = [
                color = "red";
                position = "12";
                prodId = "cms_1234";
            ];
```

- Mandatory: No
- **Default:** No

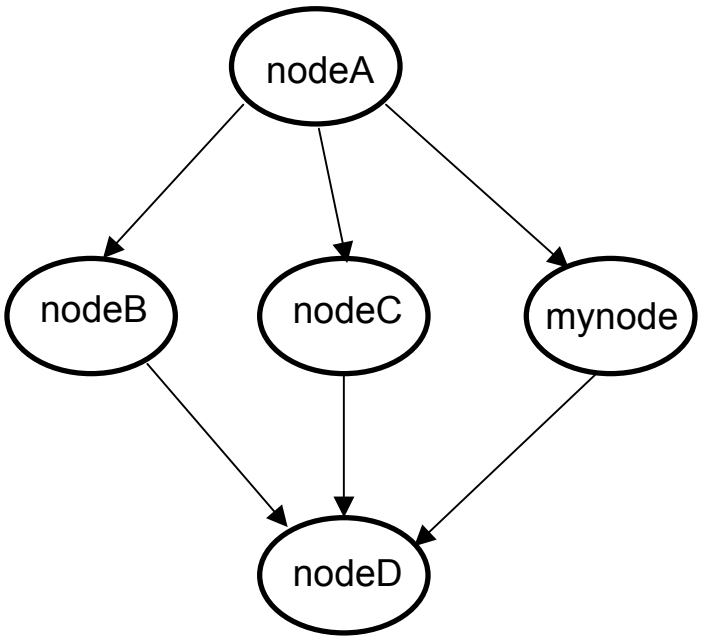
4. DAG ATTRIBUTES DESCRIPTION

A DAG (directed acyclic graph) represents a set of jobs where the input, output, or execution of one or more jobs depends on one or more other jobs. The jobs are nodes (vertices) in the graph and the edges (arcs) identify the dependencies.

Although not requiring a great number of new attributes, the structure of the JDL description for a DAG differs significantly from the one for a job and it is for sure more complex. Due to this we start by providing an example of DAG description so that the structure of the JDL in this case is clear from the beginning. The new attributes and the slight differences in the semantic for some of the known ones will be highlighted in the text.

It is important to note that upon submission of a DAG, besides the identifier associated with each node, the WMS assigns also to the DAG itself an identifier that has to be used as the handle for monitoring and controlling the whole DAG.

Let's consider the DAG in next Example 1:



Example 1 - DAG example

It can be described by the following JDL:

```
[
      Type = "dag";
      VirtualOrganisation = "EGEE";
      MyProxyServer = "skurut.cesnet.cz";
      HLRLocation =
          "eth.to.infn.it:5562:/O=CESNET/O=INFN To/CN=Andrea Guarise";
```

```
InputSandbox = {
                "/tmp/foo/*.exe",
                "/home/gliteuser/bar",
                "gsiftp://neo.datamat.it:5678/tmp/cms_sim.exe ",
                "file:///tmp/myconf"
               };
InputSandboxBaseURI = "gsiftp://matrix.datamat.it:5432/tmp";
Rank = - other.GlueHostEstimatedTraversalTime;
Requirements = other.GlueCEStateStatus == "Production";
max_nodes_running = 5;
nodes = [
            nodeA = [
                  description = [
                              JobType = "Normal";
                              Executable = "a.exe";
                              InputSandbox = {
                                    "/home/data/myfile.txt",
                                    root.InputSandbox
                                          };
                              ];
                        ];
            mynode = [
                  description = [
                              JobType = "Normal";
                              Executable = "b.exe";
                              Arguments = "1 2 3";
                              RetryCount = 3;
                              Requirements =
                                    other.GlueCEInfoTotalCPUs > 2;
                              Rank = other.GlueCEStateFreeCPUs;
                              OutputSandbox = {"myoutput.txt",
                                                "myerror.txt" };
                              OutputSandboxDestURI =
                                  "gsiftp://neo.datamat.it:5432/tmp";
                             ];
                        ];
             nodeD = [
                   description = [
                              JobType = "Checkpointable";
                              Executable = "b.exe";
                              Arguments = "1 2 3";
                              RetryCount = 3;
```

```
                                    InputSandbox = {
                                                "file:///home/pippo",
                                root.nodes.mynode.description.OutputSandbox[0]
                                                };
                                    ];
                            ];
                    nodeC = [
                        file = "/home/test/c.jdl";
                            ];

                    nodeB = [
                        file = "foo.jdl";
                        node_retry_count = 2;
                             ];
            ];
      dependencies = { { nodeA, nodeB }, { nodeA, nodeC },
                               {nodeA, mynode },
                          { { nodeB, nodeC, mynode }, nodeD }
                        };
];
```

Example 2 - JDL representation for a DAG

As shown above the structure of a DAG description consists of a certain number of nested classads in the following hierarchy:

- The root classad containing the whole description with some attributes that are associated to the DAG as a whole since they are in some sense inherited by the nodes of the DAG (this mechanism will be explained in the following).
- a children classads named *Nodes* containing the description of all the nodes of the DAG and at the same level the *dependencies* attribute specifying all dependencies among the DAG sub-jobs.
- A set of classads (whose names are free) inside *Nodes* each one corresponding to a node and containing its description

The detailed description of the mentioned attributes is provided hereafter:

4.1. TYPE

See 2.1.

In this case the value for this attribute is “DAG”.

- Mandatory: Yes

4.2. VIRTUALORGANISATION

See 3.26.

The Virtual Organisation must be the same for the DAG and all its nodes. The value of this attribute is hence inherited by all nodes descriptions. If a node already contains the *VirtualOrganisation* attribute in its description, the value of the attribute is overridden by the one specified for the DAG (if different).

- Mandatory: Yes
- **Default:** **No**

4.3. MAX_RUNNING_NODES

The *max_running_nodes* attribute is an integer greater than zero representing the maximum number of nodes of a DAG that can be submitted by DAGMan at a given time. E.g.

```
max_running_nodes = 25;
```

This means that if at a certain time there are N nodes of the DAG that are free of dependencies, only *max_running_nodes* of them can be submitted simultaneously whilst the remaining ones will be submitted as soon as the already submitted one complete their execution.

This attribute is not mandatory and defaults to 10.

- Mandatory: No
- **Default:** **10**

4.4. HLRLOCATION

See 3.31.

The HLR must be the same for the DAG and all its nodes. The value of this attribute is hence inherited by all nodes descriptions. If a node already contains the *HLRLocation* attribute in its description, the value of the attribute is overridden by the one specified for the DAG (if different).

- Mandatory: No
- **Default:** **No**

4.5. LBADDRESS

See 3.28

The LB Server address must be the same for the DAG and all its nodes. The value of this attribute is hence inherited by all nodes descriptions. If a node already contains the *LBAddress* attribute in its description, the value of the attribute is overridden by the one specified for the DAG (if different).

- Mandatory: No
- **Default:** taken from WMS configuration

4.6. MYPROXYSERVER

See 3.30

The MyProxy Server must be the same for the DAG and all its nodes. The value of this attribute is hence inherited by all nodes descriptions. If a node already contains the *MyProxyServer* attribute in its description, the value of the attribute is overridden by the one specified for the DAG (if different).

- Mandatory: No
- **Default: No**

4.7. JOBPROVENANCE

See 3.32

The Job Provenance Server must be the same for the DAG and all its nodes. The value of this attribute is hence inherited by all nodes descriptions. If a node already contains the *JobProvenance* attribute in its description, the value of the attribute is overridden by the one specified for the DAG (if different).

- Mandatory: No
- **Default: No**

4.8. ALLOWZIPPEDISB

See 3.12.

For DAG requests this attribute is only take into account at this level. If the *AllowZippedISB* attribute is set to true, a single compressed archive is created for the input sandbox files of all DAG nodes.

The *AllowZippedISB* attributes (if any) specified within the nodes description are ignored by WMS.

- Mandatory: No
- **Default:** false

4.9. ZIPPEDISB

See 3.13.

For DAG requests the archive file indicated through this attribute contains the input sandbox files of all DAG nodes.

For DAG requests this attribute is only take into account at this level; the *ZippedISB* attributes (if any) specified within the nodes description are ignored by WMS.

- Mandatory: No
- **Default:** false

4.10. PERUSALFILEENABLE

See 3.16.

All nodes that do not contain the *PerusalFileEnable* attribute in their descriptions inherit the value of this attribute from the one specified for the DAG.

The *PerusalFileEnable* attribute is not mandatory.

- Mandatory: No
- **Default:** false

4.11. NODESCOLLOCATION

The *NodesCollocation* attribute is a Boolean attribute. When set to true, i.e.

```
NodesCollocation = true;
```

makes the WMS submit all the nodes of the DAG to the same CE selected performing the matchmaking based on the *Requirements* and *Rank* expression specified in the JDL at the level of the DAG.

The *NodesCollocation* attribute is not mandatory. If not specified in the JDL it is assumed to be set to false.

- Mandatory: No
- **Default:** `false`

4.12. USERTAGS

See 3.43.

The key,value pairs specified for the DAG are only applied to the DAG as a whole and not to the nodes that can have their own tags specified within their descriptions.

- Mandatory: No
- **Default:** `No`

4.13. REQUIREMENTS AND RANK

See 3.40 and 3.41.

All nodes that do not contain the Requirements and/or Rank expressions in their descriptions inherit the value of these attributes from the one specified for the DAG. E.g. in Example 2, *nodeD* inherits *Requirements* and *Rank* from the DAG whilst *mynode* doesn't.

- Mandatory: No
- **Default:** **No**

4.14. INPUTSANDBOX AND INPUTSANDBOXBASEURI

See 3.7 and 3.8.

All nodes that do not contain the *InputSandbox* and/or the *InputSandboxBaseURI* attributes in their descriptions inherit the value of these attributes from the one specified for the DAG. This rules applies also to the single attribute, i.e. a node that only contains the *InputSandbox* in its description, inherits the *InputSandboxBaseURI* (this is the case for *nodeD* in Example 2) from the DAG and vice-versa. Note however that the *InputSandboxBaseURI* is not applied to files already specified as URI or prefixed with the "file:" URI scheme. Nodes representing jobs without input sandbox have to contain the following specification in their description (empy *InputSandbox* list):

```
InputSandbox = {};
```

The inheritance of the *InputSandbox* attribute allows the introduction of the concept of the "shared sandbox", i.e. a sandbox that is common to multiple jobs (some of the nodes of the DAG) and that needs to be transferred on the WMS node only once.

As it can be seen in Example 2 it is possible within the description of a node, to make a reference to an attribute either of the DAG or of another node. E.g. in *nodeA* the *InputSandbox* includes the input sandbox of the DAG, i.e. the resulting sandbox for the node is:

```
InputSandbox = {
                "/home/data/myfile.txt",
                "/tmp/foo",
                "/home/gliteuser/bar",
                "gsiftp://neo.datamat.it:5678/tmp/cms_sim.exe ",
                "file:///tmp/myconf"
              };
```

Another example of attribute reference is the *InputSandbox* of *nodeD* that refers to a file in the output sandbox of the node *mynode*. The resulting InputSandbox list for *nodeD* is:

```
InputSandbox = {
                   "file:///home/pippo",
                   "gsiftp://neo.datamat.it:5432/tmp/myoutput.txt";
                };
```

Both attributes described in this section are:

- Mandatory: No
- **Default: No**

4.15. OUTPUTSANDBOXBASEDESTURI

See 3.11.

All nodes that contain neither the *OutputSandboxDestURI* nor the *OutputSandboxBaseDestURI* attributes in their descriptions inherit the value of *OutputSandboxBaseDestURI* from the one specified for the DAG.

- Mandatory: No
- **Default: No**

4.16. NODES

The *Nodes* attribute is the core of the DAG description and it is used for specifying the nodes and their dependencies. It is a classad containing all the nodes of the DAG and the attribute named *Dependencies*. Each node corresponds to a classad attribute (*nodeA*, *nodeB*, *nodeC*, *nodeD* and *mynode* in the previous Example 2), whose name can be freely set by the user, which contains the description of the job representing the node. Such description can be provided through the following attributes:

4.16.1. File

The *File* attribute is a string representing the absolute path (or relative to the home of the user) on the local file system to a file containing the JDL description of a Job. It is important to note that this kind of representation can only be used when submitting to the WMS through a client (like e.g. **glite-wms-job-submit** [R15]) able to resolve the path locally and to expand the JDL with the full description before passing it to the WMS.

The job description provided within the file pointed by *File* is subjected to all rules reported in section 3. The only limitation that applies is on the *JobType* allowed for a DAG node that as already signalled in section 2 can be neither *Parametric* nor *Partitionable*.

The *File* attribute cannot be specified together with the *Description* attribute (see 4.16.2) within the same node description.

- Mandatory: No (provided *Description* is specified)
- **Default: No**

4.16.2. Description

The *Description* attribute is a classad representing the JDL description of the job representing the node of the DAG.

The job description provided by this attribute is subjected to all rules reported in section 3. The only limitation that applies is on the allowed *JobType* that as already signalled in section 2 can be neither *Parametric* nor *Partitionable*.

The *Description* attribute cannot be specified together with the *File* attribute (see 4.16.1) within the same node description.

- Mandatory: No (provided *File* is specified)
- **Default: No**

4.17. DEPENDENCIES

The *Dependencies* attribute is a list of lists representing the dependencies between the nodes of the DAG described in the *nodes* attribute. Each sub-list specifies dependencies between a sub set of nodes.

The *dependencies* specification (taking e.g. the one in Example 2) must be read as follows:

`{ nodeA, nodeB }` means that *nodeB* cannot start before *nodeA* has completed its execution successfully

`{ nodeA, nodeC }` means that *nodeC* cannot start before *nodeA* has completed its execution successfully

`{ nodeA, mynode }` means that *mynode* cannot start before *nodeA* has completed its execution successfully

`{ { nodeB, nodeC, mynode }, nodeD }` means that *nodeD* cannot start before *nodeB, nodeC* and *mynode* have completed their execution successfully

The form `{ { nodeB, nodeC, mynode }, nodeD }` is equivalent to:

```
{ nodeB, nodeD },
{ nodeC, nodeD },
{ mynode, nodeD }
```

and it is also equivalent to:

```
{ { nodeB, nodeC, mynode }, { nodeD } }
```

The description of a pipeline of jobs could be accomplished by specifying the following dependencies:

```
{ { nodeA, nodeB}, {nodeB, nodeC}, {nodeC, nodeD }, {nodeD, nodeE} … }
```

The *dependencies* attribute is mandatory for a DAG description although it can be an empty list (i.e. `dependencies = {};`) and can also be specified within the *nodes* classad.

- Mandatory: Yes
- **Default:** **No**

4.18. OUTPUTSANDBOX

The JDL description of a DAG must not contain any *OutputSandbox* attribute occurrence besides the ones in the descriptions of the nodes. The *OutputSandbox* of the DAG (i.e. the files retrieved e.g. using

the **glite-wms-job-output** command [R15]) has indeed to be considered as the sum of the output sandboxes of all its nodes.

5. PARTITIONABLE JOB ATTRIBUTES DESCRIPTION

We describe here in detail the JDL for a partitionable job.

The structure of the JDL for a partitionable job slightly differs from the one for the other job types as it can encompass two further job descriptions specifying the "pre" and "post" jobs characteristics.

The main idea behind the concept of partitionable jobs is the ability to describe the processing of a job as a set of independent steps/iterations (note that the various steps must be independent), so that the job can be thought of as composed by a set of independent sub-jobs each one taking care of a step or of a sub-set of steps, and which can be executed in parallel.

Upon submission of a partitionable job the WMS generates a DAG (see section 4) whose nodes are the sub-jobs, the prejobs and the postjob (a.k.a. the aggregator):

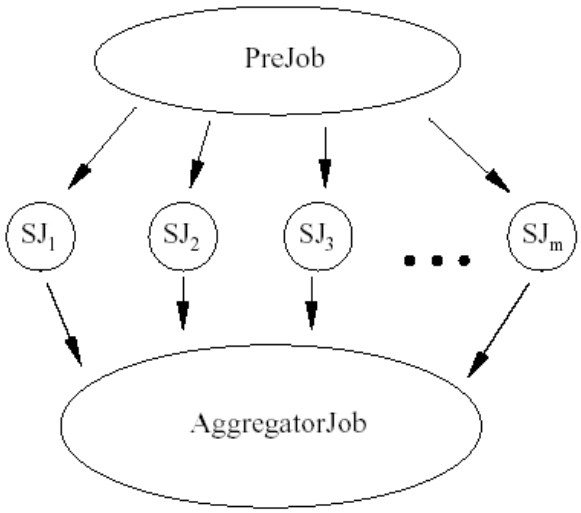


Figure 1 - DAG generated for a partitionable job

In the DAG in Figure 1 each node SJ_i peforms a sub-set of the partitionable job steps. Section 5.1 describes how the partition is done.

Here below is shown an example of a partitionable job:

```
[
  JobType = "partitionable";
  VirtualOrganisation = "EGEE";
  Executable = "hsum" ;
  JobSteps = {"cms0", "cms1", "cms2", "cms3", "orca"};
  StepWeight = {7.5, 25, 37.5, 15, 15};
  CurrentStep = 0;
```

```
    Stdoutput = "std.out" ;
    StdError =  "std.err" ;
    InputSandbox = "/home/cms/prod/hsum";
    OutputSandbox ={"std.out" ,"std.err"} ;
    prejob=[
            Executable = "prod_prepa";
            InputSandbox = "/home/cms/prod_prepa";
            rank = other.GlueCEStateFreeCPUs;
            requirements =  other.GlueCEInfoTotalCPUs > 2 ;
           ];
    postjob=[
             JobType = "checkpointable";
             Executable = "aggregator" ;
             Arguments = "5";
             InputSandbox = "/home/cms/prod/aggregator";
             rank = -other.GlueCEStateEstimatedResponseTime ;
             requirements =  other.GlueCEStateStatus == "Production" ;
            ];
    requirements = Member("GATE-1.0-3",
                   other.GlueHostApplicationSoftwareRunTimeEnvironment);
    rank = -other.GlueCEStateEstimatedResponseTime ;
]
```

Example 3 - JDL representation for a Partitionable Job

As premised and as it is shown in Example 3, besides specifying the characteristics and the requirements of the partitionable job, the description also contains the JDL for the *postjob* or "job aggregator" (the job responsible for collating and merging together the results for the various sub-jobs in which the original job is being partitioned) and a *prejob*, that is a "preparation" job which must be executed before the various sub-jobs. Note that both the *prejob* and the *postjob* executables are optional and have to be provided by the user.

The two attributes *JobSteps* and *StepWeight* respectively list the job independent steps and the "weights" associated to the various elements of the list. The latter information has to be intended as a "hint" given by the user to help the WMS in performing a good partitioning.

The JDL description representing a partitionable job is transformed in a JDL representing a DAG (see section 4) composed by *m*+2 nodes: *m* nodes representing the *m* sub-jobs in which the job has been split (which can be executed in parallel) and 2 other nodes, representing the pre-job (which is executed before the *m* sub-jobs), and the job aggregator (which can start its execution when the other *m* sub-jobs have terminated their run).

Please note that a partitionable job must also be checkpointable (it is implicitly assumed by the WMS). The postjob must be checkpointable too.

The description of the partitionable-specific attributes is provided in the following sub-sections.

5.1. JOBSTEPS

As a partitionable job must also be checkpointable, its description should encompass the JobSteps attribute. See section 3.34 for the description of this attribute

Please note that the number of sub-jobs in which the partitionable job is split does not necessarily correspond to the number of steps listed in the JobSteps attribute. When the WMS handles a partitionable job, given the JobSteps list (assume it contains N elements) and the number of CEs satisfying job's requirements (assume they are M), it partitions the JobSteps list in M sub-lists of steps and generates a DAG. Each sub-list corresponds to a sub-job of the DAG. Each sub-job of the DAG performs approximately N/M steps of the JobSteps list. The partition is done trying to keep the sub-list cardinality as similar as possible, i.e.assuming that each step has the same weight (see section 5.3).

Only if $M \geq N$ the number of sub-jobs equals the *JobSteps* list cardinality.

5.2. CURRENTSTEP

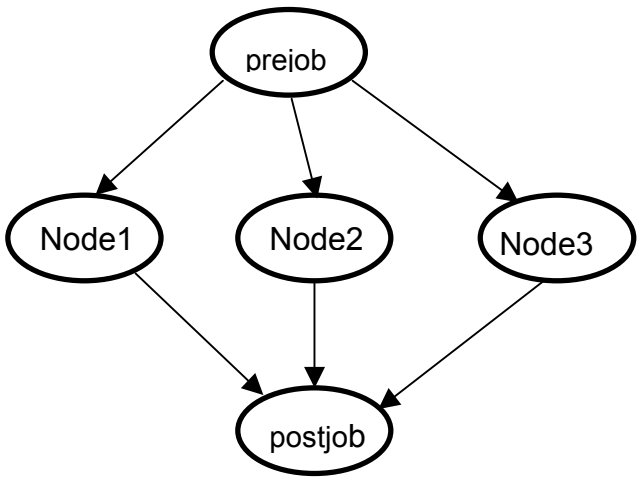
As a partitionable job must also be checkpointable, its description should encompass the CurrentStep attribute. See section 3.35 for the description of this attribute

5.3. STEPWEIGHT

The StepWeight attribute is a list of floats representing the "weights" for the elements of the *JobSteps* list (section 3.34) for a partitionable job (see [R4] and [R11] for details). E.g.:

```
StepWeight = {7.5, 15, 55, 15, 7.5};
```

It has to be interpreted as a "hint" to help the partitioning process of the WMS, responsible to partition the *JobSteps* list into sub-lists (see section 5.1). If the user doesn't specify this attribute, then all the *JobSteps* elements are considered with the same weight.
For the partitionable job shown in Example 3, assuming the WMS finds three available CEs the resulting DAG would be e.g.:



Where Node1 performs steps *cms0* and *cms1*, Node2 performs step *cms2* and Node3 performs steps *cms3* and *orca*.

This attribute can only be specified for partitionable jobs.

- Mandatory: No
- **Default:** Same weight for each step

5.4. PREJOB

The *PreJob* attribute is a classad representing the job that has to be executed before all the sub-jobs in which a partitionable job is split. E.g.

```
PreJob = [
            Type = "job";
            JobType = "Normal";
            VirtualOrganisation = "my_vo";
            Executable = "hsum";
            …
          ]
```

This attribute can only be specified for *Partitionable* jobs. The *PreJob* classad has to be considered as a job description and hence it is subjected to the same rules. The only limitation on the job type it represents is that it cannot be a *partitionable* or a *parametric* job.

- Mandatory: No
- **Default:** No

5.5. POSTJOB

The PostJob attribute is a classad representing the job that has to be executed after all the sub-jobs in which a partitionable job is split have completed their execution. It is responsible for collating and merging together the results for the various sub-jobs (that have saved their final states) in which the original job is being partitioned. It can be considered as an “aggregator”.

```
PostJob = [
            Type = "job";
            JobType = "Checkpointable";
            VirtualOrganisation = "my_vo";
            Executable = "aggregator";
            …
          ]
```

This attribute can only be specified for *Partitionable* jobs. The *PostJob* classad has to be considered as a job description and hence it is subjected to the same rules. As it has to "retrieve" the final states of the sub-jobs to collect together these partial results, it has to be a *checkpointable* job (as all the sub-jobs of a partitionable job are).

- Mandatory: No
- **Default:** No

6. PARAMETRIC JOB ATTRIBUTES DESCRIPTION

We describe here in detail the JDL for a parametric job.

A *Parametric* job is a job having one or more parametric attributes in the JDL. The parametric attributes vary their values according to another attribute (named *Parameters*) also specified in the job description. The submission of a *Parametric* job results in the submission of a set of jobs having the same descriptions apart from the values of the parametric attributes. Both the parametric job and all jobs resulting from the submission of it are assigned by the WMS with an identifier so that it is possible to monitor and control each of them separately and as a single entity (through the parametric job handle).

To clarify the concept, let us consider the following *Parametric* job example:

```
[
  JobType = "Parametric";
  Executable = "cms_sim.exe";
  StdInput = "input_PARAM_.txt";
  StdOutput = "myoutput_PARAM_.txt";
  StdError = "myerror_PARAM_.txt";
  Parameters = 10000;
  ParameterStart = 1000;
  ParameterStep = 10;
  InputSandbox = {
                 "file:///home/cms/cms_sim.exe",
                 "file:///home/cms/data/input_PARAM_.txt "
                 };
  OutputSandbox = {
                  "myoutput_PARAM_.txt",
                  "myerror_PARAM_.txt" };
  OutputSandboxDestURI =  "gsiftp://neo.datamat.it:5432/tmp";
  Requirements = other.GlueCEInfoTotalCPUs > 2;
  Rank = other.GlueCEStateFreeCPUs;
]
```

Example 4 – JDL representation for a Parametric Job

As shown in the above Example 4, besides the attributes available for *normal* jobs (described in section 3), the JDL for a parametric job contains three additional specific attributes:

- *Parameters*
- *ParameterStart*
- *ParameterStep*

which represent respectively the set of values the parametric attributes should assume, the starting value for the variation (if the parameter is a number) and the step for each variation. Moreover the

parametric attributes in the JDL, i.e. the ones whose values should vary, contain the "_PARAM_" instruction within their values.

The submission of the JDL in Example 4 will result in the generation of N jobs, where

```
N = (Parameters – ParameterStart)/ParameterStep
```

each one containing the same JDL but the parametric attributes for which the "_PARAM_" instruction is replaced with the actual value of the parameter, e.g. for the *StdInput* attribute of Example 4 it would be respectively in the resulting jobs:

- `StdInput = "input1000.txt";`
- `StdInput = "input1010.txt";`
- `StdInput = "input1020.txt";`
- …
- `StdInput = "input9990.txt";`

The applied value of the parameter is exported to the job run time environment through the variable *ParamenterValue* set by the WMS *JobWrapper* script before the job is started. E.g. the job related to the first bullet above would find the environment variable *ParameterValue* set equal to *1000* when running. Moreover each sub-job that is generated by the submission of a parametric job is assigned by the WMS with the "*node_<P>*" name where P is the value of the parameter for that given instance of the job (this can be seen as *Node Name* field of the **glite-job-status** command output [R16]).

In the job description of Example 4 the *Parameters* attribute is specified as a number to represent (coupled with the *ParameterStart* attribute value, if any) a range for the parameterisation (note that the value of *Parameters* is not included in the range, i.e. values vary form *ParametersStart* to *Parameters-1*). Alternatively the *Parameters* attribute can be specified as a list of values that will be used for assigning values to the parametric attributes. E.g. a JDL as follows:

```
[
  …
  Executable = "my_sim.exe";
  Arguments = "_PARAM_";
  Parameters = {alpha, beta, gamma};
  …
]
```

results in the submission of three jobs only differing for the value of the argument (respectively "alpha", "beta" and "gamma") passed to the executable (se also 9.7). Note that when *Parameters* is specified as a list of values, it does not make sense anymore to specify the *ParameterStart* and *ParameterStep* attributes.

Here below is provided the detailed description of each of the mentioned attributes.

6.1. PARAMETERS

The *Parameters* attribute is either an integer representing the upper bound (not included) or the lower bound, in case it is negative, for the values that can be assumed by the parametric attributes (i.e. the ones containing the "_PARAM_" instruction), e.g.

```
Parameters = 100000;
```

or a list of items representing the values on which the parametric attributes have to range, e.g.

```
Parameters = {raw, d0, d1, d2};
```

It is important to note that when the *Parameters* attribute is specified as a list, the values of the list must not have a "type" (e.g. have not to be enclosed within quotes if they are strings) as typing will be automatically enforced according to the type of the parametric attributes.

Obviously this attribute can only be set for parametric jobs.

- Mandatory: Yes
- **Default:** `No`

6.2. PARAMETERSTART

The *ParameterStart* attribute is an integer indicating the initial value to take into account for the parameter scanning of a *Parametric* job. E.g.

```
Parameters = 100000;
ParameterStart = 15000;
```

means that the parametric attributes values will range starting from 15000 up to 985000.

This attribute can only be set for a parametric job if the *Parameters* attribute has been specified as a number. If not provided by the user it is assumed to be 0.

- Mandatory: No
- **Default:** 0

6.3. PARAMETERSTEP

The *ParameterStep* attribute is an integer representing the size of each variation of the parameter (i.e. the difference between two subsequent values of the parameter) when ranging between the values of *ParameterStart* and *Parameters*. E.g.

```
Parameters = 100000;
ParameterStart = 15000;
ParameterStep = 1000;
```

means that for each generated job the value of the parameter will be increased of 1000 in the interval [15000, 100000).

This attribute can only be set for a parametric job if the *Parameters* attribute has been specified as a number. If not provided by the user it is assumed to be 1.

- Mandatory: No
- **Default:** 1

6.4. NODESCOLLOCATION

The *NodesCollocation* attribute is a Boolean attribute. When set to true, i.e.

```
NodesCollocation = true;
```

makes the WMS submit all the instances of the parametric job to the same CE selected performing the matchmaking based on the *Requirements* and *Rank* expression specified in the JDL.

The *NodesCollocation* attribute is not mandatory. If not specified in the JDL it is assumed to be set to false.

- Mandatory: No
- **Default:** `false`

7. JOBS COLLECTION ATTRIBUTES DESCRIPTION

A job *Collection* is a set of independent jobs that for some reasons (known to the user) have to be submitted, monitored and controlled as a single request. As it happens for a DAG, upon submission, besides the identifiers for the sub-jobs, the collection is associated with a jobId that can be used as the unique handle for the whole set of jobs.

The JDL description for a Collection is quite simple as it basically consists of a list of classads (the sub-jobs) plus some attributes that analogously to the DAG are associated to the Collection as a whole.

Here below is reported an example of a job Collection description that we will use as a basis for providing further details:

```
[
      Type = "collection";
      VirtualOrganisation = "EGEE";
      MyProxyServer = "skurut.cesnet.cz";
      HLRLocation =
          "eth.to.infn.it:5562:/O=CESNET/O=INFN To/CN=Andrea Guarise";
      InputSandbox = {
                      "/tmp/foo",
                      "/home/gliteuser/bar",
                      "gsiftp://neo.datamat.it:5678/tmp/cms_sim.exe ",
                      "file:///tmp/myconf"
                     };
      InputSandboxBaseURI = "gsiftp://matrix.datamat.it:5432/tmp";
      Rank = other.GlueHostEstimatedTraversalTime;
      Requirements = other.GlueCEStateStatus == "Production";
      nodes = {
                  [
                   JobType = "Normal";
                   Executable = "a.exe";
                   InputSandbox = {
                                    "/home/data/myfile.txt",
                                    root.InputSandbox
                                    };
                  ],
                  [
                    JobType = "Normal";
                    Executable = "b.exe";
                    Arguments = "1 2 3";
                    RetryCount = 3;
                    Requirements =
                              other.GlueCEInfoTotalCPUs > 2;
                    Rank = other.GlueCEStateFreeCPUs;
```

```
                    OutputSandbox = {"myoutput.txt",
                                     "myerror.txt" };
                    OutputSandboxDestURI =
                                 "gsiftp://neo.datamat.it:5432/tmp";
                  ],
                  [
                    NodeName = "mysubjob "
                    JobType = "Checkpointable";
                    Executable = "b.exe";
                    Arguments = "1 2 3";
                    RetryCount = 3;
                    InputSandbox = {
                                    "file:///home/pippo",
                              root.nodes[1].OutputSandbox[0]
                                    };
                    ],
                    [
                     file = "/home/test/c.jdl";
                    ]
            };
]
```

Example 5 – JDL representation for a Collection

It can be easily noted that the structure of the JDL for a collections differs from the one for a DAG. The *nodes* attribute is indeed a list containing plain classads with no further nesting (i.e. the *description* attribute doesn't exist for a collection). Moreover as the jobs of a collection are supposed to be independent, the *dependencies* attribute cannot be included in the collection description. Apart from this and the fact that the *Type* attribute has to be "Collection" in this case, all the attributes described for a DAG (see section 5) preserve the same meaning for a collection and the same "inheritance" rules apply.

The attributes of a collection are described in detail in the following sub-sections.

7.1. TYPE

See 2.1.

In this case the value for this attribute is "Collection".

- Mandatory: Yes

7.2. VIRTUALORGANISATION

See 3.26.

The Virtual Organisation must be the same for the Collection and all its jobs. The value of this attribute is hence inherited by all jobs descriptions. If a job already contains the *VirtualOrganisation* attribute in its description, the value of the attribute is overridden by the one specified for the Collection (if different).

- Mandatory: Yes
- **Default:** **No**

7.3. HLRLOCATION

See 3.31.

The HLR must be the same for the Collection and all its sub-jobs. The value of this attribute is hence inherited by all sub-jobs descriptions. If a jobs already contains the *HLRLocation* attribute in its description, the value of the attribute is overridden by the one specified for the Collection (if different).

- Mandatory: No
- **Default:** **No**

7.4. LBADDRESS

See 3.28.

The LB Server address must be the same for the Collection and all its sub-jobs. The value of this attribute is hence inherited by all sub-jobs descriptions. If a jobs already contains the *LBAddress* attribute in its description, the value of the attribute is overridden by the one specified for the Collection (if different).

- Mandatory: No
- **Default:** taken from WMS configuration

7.5. MYPROXYSERVER

See 3.30.

The MYProxy Server must be the same for the Collection and all its sub-jobs. The value of this attribute is hence inherited by all sub-jobs descriptions. If a jobs already contains the *MyProxyServer* attribute in its description, the value of the attribute is overridden by the one specified for the Collection (if different).

- Mandatory: No
- **Default:** **No**

7.6. JOBPROVENANCE

See 3.32

The Job Provenance Server must be the same for the Collection and all its sub-jobs. The value of this attribute is hence inherited by all nodes descriptions. If a node already contains the *JobProvenance*

attribute in its description, the value of the attribute is overridden by the one specified for the Collection (if different).

- Mandatory: No
- **Default: No**

7.7. ALLOWZIPPEDISB

See 3.12.

For Collection requests this attribute is only take into account at this level. If the *AllowZippedISB* attribute is set to true, a single compressed archive is created for the input sandbox files of all Collection sub-jobs.

The *AllowZippedISB* attributes (if any) specified within the sub-jobs description are ignored by WMS.

- Mandatory: No
- **Default:** false

7.8. ZIPPEDISB

See 3.13.

For Collection requests the archive file indicated through this attribute contains the input sandbox files of all collection sub-jobs.

For Collection requests this attribute is only take into account at this level; the *ZippedISB* attributes (if any) specified within the sub-jobs description are ignored by WMS.

- Mandatory: No
- **Default:** false

7.9. PERUSALFILEENABLE

See 3.16.

All nodes that do not contain the *PerusalFileEnable* attribute in their descriptions inherit the value of this attribute from the one specified for the Collection.

The *PerusalFileEnable* attribute is not mandatory.

- Mandatory: No
- **Default:** false

7.10. NODESCOLLOCATION

The *NodesCollocation* attribute is a Boolean attribute. When set to true, i.e.

```
NodesCollocation = true;
```

makes the WMS submit all the sub-jobs of the Collection to the same CE selected performing the matchmaking based on the *Requirements* and *Rank* expression specified in the JDL at the level of the Collection.

The *NodesCollocation* attribute is not mandatory. If not specified in the JDL it is assumed to be set to false.

- Mandatory: No
- **Default:** `false`

7.11. USERTAGS

See 3.43.

The key,value pairs specified for the Collection are only applied to the Collection as a whole and not to the sub-jobs that can have their own tags specified within their descriptions.

- Mandatory: No
- **Default:** `No`

7.12. REQUIREMENTS AND RANK

See 3.40 and 3.41.

All jobs that do not contain the Requirements and/or Rank expressions in their descriptions inherit the value of these attributes from the one specified for the Collection. E.g. in Example 5, the first and the third jobs in the *nodes* list inherit *Requirements* and *Rank* from the Collection whilst the second job doesn't.

- Mandatory: No
- **Default:** **No**

7.13. INPUTSANDBOX AND INPUTSANDBOXBASEURI

See 3.7 and 3.8.

All jobs that do not contain the *InputSandbox* and/or the *InputSandboxBaseURI* attributes in their descriptions inherit the value of these attributes from the one specified for the collection. This rules also applies to the single attribute, i.e. a job that only contains the *InputSandbox* in its description, inherits the *InputSandboxBaseURI* (this is the case for the third job in the *nodes* list in Example 5) from the collection. The same happens if only *InputSandboxBaseURI* is specified in the sub-jobs descriptions.

Note however that the *InputSandboxBaseURI* is not applied to files already specified as URI or prefixed with the "file://" URI scheme. Jobs of the collection without input sandbox have to contain the following specification in their description (empy *InputSandbox* list):

```
InputSandbox = {};
```

so that the attribute is not inherited from the collection.

The inheritance of the *InputSandbox* attribute allows the introduction of the concept of the "shared sandbox", i.e. a sandbox that is common to multiple jobs of the collection and that needs to be transferred on the WMS node only once.

As it can be seen in Example 5 it is possible within the description of a node, to make a reference to an attribute either of the collection or of another node. E.g. in the first job of the *nodes* list the *InputSandbox* attribute includes the input sandbox of the collection, i.e. the resulting sandbox for the node is:

```
InputSandbox = {
                "/home/data/myfile.txt",
                "/tmp/foo",
                "/home/gliteuser/bar",
                "gsiftp://neo.datamat.it:5678/tmp/cms_sim.exe ",
                "file:///tmp/myconf"
              };
```

Another example of attribute reference is the *InputSandbox* attribute of the third job of the *nodes* list that refers to the first file in the output sandbox of the second job of the *nodes* list (although this kind of reference is discouraged as the jobs of a collection are intended as independent and hence the user cannot assume that the output sandbox of a given job is ready before another job is started; DAGs have to be used at this purpose). The resulting *InputSandbox* list for the third job of the *nodes* list is:

```
InputSandbox = {
                  "file:///home/pippo",
                  "gsiftp://neo.datamat.it:5432/tmp/myoutput.txt";
               };
```

Both attributes described in this section are:

- Mandatory: No
- **Default: No**

7.14. OUTPUTSANDBOXBASEDESTURI

See 3.11.

All sub-jobs that contain neither the *OutputSandboxDestURI* nor the *OutputSandboxBaseDestURI* attributes in their descriptions inherit the value of *OutputSandboxBaseDestURI* from the one specified for the Collection.

- Mandatory: No
- **Default: No**

7.15. NODES

The *Nodes* attribute is the core of the Collection description and it is used for specifying the jobs the collection is made of. It is a list containing all the classads describing the jobs.

The job descriptions provided in the *nodes* list are subjected to all rules reported in section 3. The only limitation that applies is on the allowed *JobType* that as already signalled in section 2 can be neither *Parametric* nor *Partitionable*.

The description of a node of the collection can be either provided directly in the classad (like e.g. the first job in the *nodes* list of Example 5) or through a pointer to a JDL file (like e.g. the last job in the *nodes* list of Example 5) by using the *File* attribute. A collection job can be assigned with a name by means of the *NodeName* attribute as it is done e.g. in the third job of the *nodes* list of Example 4. This is very useful when monitoring the collection as the user can easily associate the given names to the automatically assigned job identifiers.

The above-mentioned attributes are described here below:

7.15.1. File

The *File* attribute is a string representing the absolute path (or relative to the current working directory) on the local file system to a file containing the JDL description of a Job. It is important to note that this kind of representation can only be used when submitting to the WMS through a client (like e.g. **glite-wms-job-submit** [R15]) able to resolve the path locally and to expand the JDL with the full description before passing it to the WMS.

The job description provided within the file pointed by the *File* attribute is subjected to all rules reported in section 3. The only limitation that applies is on the *JobType* allowed for a Collection job that as already signalled in section 2 can be neither *Parametric* nor *Partitionable*.

- Mandatory: No
- **Default: No**

7.15.2. NodeName

The *NodeName* attribute is a string representing the name that the user wants to associate to the job of a collection so that the job can be easily identified at job monitoring time (see e.g. the specification of the third job of the *nodes* list of Example 5).

For those jobs of a collection whose description does not contain this attribute, the *NodeName* is automatically assigned by the WMS and set equal to:

```
NodeName = "node<N>";
```

Where N is a number starting from 0 that indicates the position of the job classad in the *nodes* list. E.g. in the collection of Example 5, assuming that the description contained in `/home/test/c.jdl` does not specifie the *NodeName* attribute, the job names would be: node0, node1, mysubjob, node3.

Note that when the *File* attribute is used in the collection description, then the *NodeName* can also be specified at the same level as the File attribute.

Note moreover that due to the attributes naming rules imposed by the Condor classad library, the *NodeName* string attribute cannot start with a number.

- Mandatory: No
- **Default:** node<N> assigned by WMS

7.16. OUTPUTSANDBOX

The JDL description of a Collection must not contain any *OutputSandbox* attribute occurrence besides the ones in the descriptions of the jobs of the collection. The *OutputSandbox* of the Collection (i.e. the files retrieved e.g. using the **glite-wms-job-output** command [R15]) must indeed to be considered as the sum of the output sandboxes of all its jobs.

8. SPECIAL JDL EXPRESSIONS

Next sections briefly describe how it is possible to drive the resources discovery and selection process by means of special expressions for the *Requirements* and *Rank* attributes.

8.1. GANG-MATCHING

The matchmaking mainly occurs as a two-step process: entities (i.e., servers and customers) requiring matchmaking services express their characteristics, requirements and preferences to a matchmaker in *classified advertisements* (Step 1). Attributes of candidate classads are accessed via the pseudo-attribute `other` and the matchmaker employs a very generic *matchmaking algorithm* to evaluate the requirements and rank of the involved entities (Step 2). When the RB performs the matchmaking for scheduling a job, the involved entities are the job (whose classads has been provided by the user) and the CE (whose classad is built by the RB with the information from IS).

If we consider for example a job that requires a CE and a determined amount of free space on a close SE to run successfully, the matchmaking solution to this problem requires three participants in the match (i.e., job, CE and SE), which cannot be accommodated by conventional (bilateral) matchmaking. The gangmatching feature of the classads library provides a multilateral matchmaking formalism to address this deficiency.

In order to exploit this new important extension of the classads library it suffices including the appropriate classads built-in functions in the requirements expression.

A useful example, as already premised, is the usage of gangmatching to require a certain amount of free space on a SE close to the execution CE. This can be achieved specifying the job *Requirements* expression as follows:

```
Requirements =  anyMatch(   other.storage.CloseSEs ,
                            target.GlueSAStateAvailableSpace > 200);
```

This makes indeed the RB find the CEs having a close SE with at least 200 MB of free space available for the VO the user belongs to.

The newly supported classads built-in functions are:

- `anyMatch()`
- `whichMatch()`
- `allMatch()`

Information and details about gangmatching and usage of this functions are provided in document [R4].

9. JDL EXAMPLES

In the following sections are reported simple examples of JDL describing different types of jobs and requests.

9.1. JOB WITHOUT DATA REQUIREMENTS

```
[
    Type = "job";
    JobType = "normal";
    Executable = "script.sh";
    Arguments = "60";
    StdOutput = "sim.out";
    StdError = "sim.err";
    MyProxyServer = "skurut.cesnet.cz";
   OutputSandbox={
                  "sim.err",
                  "sim.out"
                  };
    OutputSandboxDestURI = {
           "gsiftp://matrix.datamat.it:5432/tmp/sim.err",
           "gsiftp://grid003.ct.infn.it:6789/home/cms/sim.out",
                  };
    // This attribute triggers accounting
    HLRLocation     =     "lilith.to.infn.it:56568:/C=IT/O=INFN/OU=Personal
        Certificate/L=Torino/CN=Andrea Guarise/Email=A.Guarise@to.infn.it";
    InputSandbox = {
                       "file:///home/fpacini/GUI/sbin/script.sh"
                 };
    rank = (other.GlueCEPolicyMaxRunningJobs-other.GlueCEStateRunningJobs);
    // This is the default requirements expression
    requirements = other.GlueCEStateStatus == "Production" ;
]
```

9.2. JOB WITH DATA REQUIREMENTS

```
[
    Type = "job";
    // JobType is not mandatory - If not specified "normal" is the default
    JobType = "normal";
    VirtualOrganisation = "cms";
    Executable = "test.sh";
    Arguments = "1 20000 sim1";
```

```
    StdInput = "file2";
    StdOutput = "sim.out";
    StdError = "sim.err";
    OutputSandbox = {"sim.out", "sim.err"};
    // disable job deep re-submission in case of failure
    RetryCount = 0;
    ShallowRetryCount = 5;
    DataRequirements = {
        [
         InputData = {
                  "lfn:/mydata/file1",  "lfn:/mydata/file2",
                  "guid:135b7b23-4a6a-11d7-87e7-9d101f8c8b70"
                    };
         DataCatalogType = "RLS";
         DataCatalog = "https://lcg.cern.ch/RLS";
        ],
        [
         InputData = {
                 "lfn:/grid/egee/my_test/test_LFN",
                 "lfn:/data/mydatafile2",
                 "guid:34r57b2312-6a33-p45d9982-4e101f8f3b61"
                    };
         DataCatalogType = "SI";
     // Do not specify this attribute if you want to use the VO default SI
        DataCatalog = "https://lxb.cern.ch:8443/egee/data/service/SEIndex";
        ]
     };
   DataAccessProtocol = {"rfio","gsiftp","gsidcap"};
    OutputSE = "grid011.pd.infn.it";
    InputSandbox = {
        "/home/fpacini/JNI/test.sh",
        "/home/fpacini/DATA/file2",
        "/home/fpacini/DATA/sim.dat",
        "/home/fpacini/HandsOn-0409/WP1testA"
    };
    rank = - other.GlueCEStateEstimatedResponseTime;
    // Do not send my job to a CE located at NIKHEF
    requirements     =     (!(RegExp("*nikhef*",other.GlueCEUniqueID)))     &&
                                  (other.GlueCEInfoLRMSType=="lsf");
]
```

9.3. JOB WITH OUTPUT DATA (NOT YET SUPPORTED)

```
[
    Type = "job";
    JobType = "normal";
    VirtualOrganisation = "cms";
    Executable = "test.sh";
    Arguments = "1 20000 sim1";
    StdInput = "file2";
    StdOutput = "sim.out";
    StdError = "sim.err";
    OutputSandbox = {
                        "sim.out",
                        "sim.err"
                    };
    RetryCount = 2;
    OutputData = {
        [
            // No StorageElement is specified - Close SE is taken
            OutputFile = "dataset1.out";
            LogicalFileName = "lfn:/myout/data.1"
        ],
        [
            OutputFile = "dataset2.out";
            LogicalFileName = "lfn:/myout/data.2"
        ]
    };
    InputSandbox = {
                    "/home/mytest/JNI/test.sh",
                    "/home/mytest/DATA/file2",
                    "/home/mytest/DATA/sim.dat"
                  };
    Environment = "SIM_ROOT=/usr/local/";
    rank = other.GlueHostMainMemoryRAMSize;
    // job is submitted to a CE having a close SE with at least 5GB free
    requirements = anyMatch( other.storage.CloseSEs,
                    target.GlueSAStateAvailableSpace > 5120);
 ]
```

9.4. JOB WITH INPUT AND OUTPUT DATA (NOT YET SUPPORTED)

```
[
    Type = "job";
```

```
// JobType is not mandatory - If not specified "normal" is the default
JobType = "normal";
VirtualOrganisation = "cms";
Executable = "test.sh";
Arguments = "1 20000 sim1";
StdInput = "file2";
StdOutput = "sim.out";
StdError = "sim.err";
OutputSandbox = {"sim.out", "sim.err"};
// disable job re-submission in case of failure
RetryCount = 0;
DataRequirements = {
    [
     InputData = {
              "lfn:/mydata/file1",  "lfn:/mydata/file2",
              "guid:135b7b23-4a6a-11d7-87e7-9d101f8c8b70"
                };
     DataCatalogType = "RLS";
     DataCatalog = "https://lcg.cern.ch/RLS";
    ]  };
DataAccessProtocol = {"gsiftp", "file"};
OutputData = {
    [
        OutputFile = "dataset1.out";
        StorageElement = "grid011.pd.infn.it";
        LogicalFileName = "lfn:/myout/data.1"
    ],
    [
        OutputFile = "dataset2.out";
        LogicalFileName = "lfn:/myout/data.2"
    ],
    [
        OutputFile = "dataset3.out"
    ],
    [
        OutputFile = "dataset4.out";
        StorageElement = "grid001.ct.infn.it"
    ]
};
OutputSE = "grid011.pd.infn.it";
InputSandbox = {
             "/home/fpacini/JNI/test.sh", "/home/fpacini/DATA/file2",
```

```
                "/home/fpacini/DATA/sim.dat",
                "/home/fpacini/HandsOn-0409/WP1testA"
                };
    rank = - other.GlueCEStateEstimatedResponseTime;
    // Do not send my job to a CE located at NIKHEF
    requirements    =    (!(RegExp("*nikhef*",other.GlueCEUniqueID)))    &&
                                (other.GlueCEInfoLRMSType=="lsf");

]
```

9.5. INTERACTIVE JOB

```
  [
    Type = "job";
    JobType = "interactive";
    VirtualOrganisation = "my_vo";
    Executable = "scriptint.sh";
    RetryCount = 1;
    // grid_console_shadow listens on this port. If not specified is
    // assigned by the OS
    ListenerPort = 9600;
    FuzzyRank = true;
    InputSandbox = {
        "/home/fpacini/JDL2.0/fox/scriptint.sh",
        "/home/fpacini/JDL2.0/fox/cpi",
        "/home/fpacini/DATA/sim.dat"
    };
    requirements  =  (other.GlueHostOperatingSystemRelease  ==  "LINUX")  &&
                       (other.GlueHostMainMemoryRAMSize >= 128)
    // this is needed for Interactive jobs. Don't need to specify it. It is
    // added automatically by UI
                    && (other.GlueHostNetworkAdapterOutboundIP);
    rank = other.GlueHostBenchmarkSF00
]
```

9.6. CHECKPOINTABLE JOB

```
[
    Type = "job";
    JobType = "checkpointable";
    VirtualOrganisation = "my_vo";
    // This is the total number of steps for the job. Not mandatory
    // if your job already knows it
```

```
    JobSteps = 10000000;
    CurrentStep = 1;
    Executable = "hsum";
    Arguments = "200000 gsiftp://lxde01.pd.infn.it/tmp/root_test/";
    StdOutput = "sim.out";
    StdError = "sim.err";
    // Old style specification of InputData. Still accepted for backward
    // compatibility. DataCatalogType is assumed to be "RLS"
    InputData = {
        "lfn:/data/wp1-test-file-01-lfn",
        "lfn:/data/wp1-test-file-02-lfn",
        "lfn:/data/wp1-test-file-04-lfn"
    };
    DataAccessProtocol = {"https", "rfio"};
    OutputSandbox = {
        "sim.err",
        "sim.out"
    };
    RetryCount = 3;
    InputSandbox = {
        "file:///home/fpacini/GUI/sbin/hsum"
    };
    // This is the default rank expression
    rank = -other.GlueCEStateEstimatedResponseTime;
    // semicolon ";" can be omitted for last attribute specification
    requirements = other.GlueCEInfoLRMSType=="pbs"
]
```

9.7. PARAMETRIC JOB

```
    [
      JobType = "Parametric";
      Executable = "sim.exe";
      StdOutput = "myoutput.txt";
      StdError = "myerror.txt"
      // The iteration is done on the values of this list
      Parameters = {/DC1/test/data, /DC2/test/data,
                    /DC3/test/data, /DC4/test/data};
      DataRequirements = {
                  [
                   DataCatalogType = "RLS";
                   // _PARAM_ assumes values from the Parameters list
                   InputData = {"lfn:_PARAM_",
```

```
                                    " guid:135b7b23-4a6a-11d7-87e7-
                          9d101f8c8b70"};
                    ]
                  };
        InputSandbox = {
                          "gsiftp://neo.datamt.it:3344/home/cms/sim.exe",
                          };
        OutputSandbox = {
                           "myoutput.txt",
                           "myerror.txt" };
        Requirements = other.GlueCEInfoTotalCPUs > 2;
        Rank = other.GlueCEStateFreeCPUs;
      ]
```

See also Example 4.

9.8. MPI JOB

```
[
    Type = "job";
    JobType = "mpich";
    VirtualOrganisation = "egtest";
    // This is the minimum number of CPU needed by the job
    NodeNumber = 6;
    Executable = "cpi";
   StdOutput = "sim.out";
    StdError = "sim.err";
    OutputSandbox = {
        "sim.err",
        "sim.out"
    };
    OutputSandboxDestURI = "gsiftp://lxplus.cern.ch:3344/home/egtes";
    // This attribute triggers the proxy-renewal mechanism
    MyProxyServer = "skurut.cesnet.cz";
    RetryCount = 3;
    InputSandbox = {
        "file:///home/fpacini/JDL2.0/cpi"
    };
   requirements      =      other.GlueHostNetworkAdapterOutboundIP      &&
     Member("IDL2.0",other.GlueHostApplicationSoftwareRunTimeEnvironment);
   rank = other.GlueCEStateFreeCPUs;;
]
```

9.9. DAG

```
[
   type = "dag";
   VirtualOrganisation = "EGEE";
   max_nodes_running = 10;
   // shared by all nodes not specifying their own InputSandbox
   InputSandbox =  {
                    "cdfSim.sh",
                    "run_cdfSim.tcl"
                   };
   requirements = other.GlueCEInfoLRMSType=="LSF";
   rank = other.GlueCEInfoTotalCPUs ;
   nodes = [
     nodeA = [
               file ="cdfSimA.jdl" ;
             ];
     nodeB = [
               file ="cdfSimB.jdl" ;
             ];
     nodeC = [
               file ="cdfSimC.jdl" ;
             ];
     nodeD = [
               file ="cdfSimD.jdl" ;
             ];
     nodeE = [
               file ="cdfSimE.jdl" ;
             ];
     nodeF = [
               file ="cdfSimF.jdl" ;
             ];
     nodeG = [
               file ="cdfSimG.jdl" ;
             ];
     nodeH = [
               file ="cdfSimH.jdl" ;
             ];
     nodeI = [
               file ="cdfSimI.jdl" ;
             ];
     nodeL = [
               file ="cdfSimL.jdl" ;
             ];
     nodeM = [
               file ="cdfSimM.jdl" ;
             ];
     nodeN = [
               file ="cdfSimN.jdl" ;
             ];
     nodeO = [
               description = [
                       JobType =
                       Executable    = "cdfSim.sh";
```

```
                    Arguments     = "13";
                    StdOutput     = "cdfSim.out";
                    StdError      = "cdfSim.err";
                    OutputSandbox = { "cdfSim.out",
                                      "cdfSim.err",
                                      "run_cdfSim.tcl_13.log"};
                    OutputSandboxDestURI =
                                      "gsiftp://gr1.infn.it:5432/tmp";
                                      ];
                                             ];
  nodeP = [
            file ="cdfSimP.jdl" ;
          ];
  nodeQ = [
            file ="cdfSimQ.jdl" ;            ];
  nodeR = [
            file ="cdfSimR.jdl" ;
          ];
  nodeS = [
            file ="cdfSimS.jdl" ;
          ];
  nodeT = [
            file ="cdfSimT.jdl" ;
          ];
        ];
  dependencies = {
    { nodeA, nodeB },
    {{nodeB, nodeC}, nodeD},
    {nodeD, nodeE},
    {nodeE, {nodeF, nodeG, nodeH , nodeI, nodeL, nodeM, nodeN, nodeO}},
    {{nodeG, nodeO}, nodeP},
    {nodeP, nodeQ},{nodeP,nodeR},
    {{nodeP, nodeR}, nodeS},
    {nodeH, nodeT}
  }
];
```

See also Example 2.

9.10. COLLECTION

```
[
    Type = "collection";
    VirtualOrganisation = "EGEE";
    MyProxyServer = "skurut.cesnet.cz";
    InputSandbox = {
                    "/tmp/foo",
                    "/home/gliteuser/bar",
                    "gsiftp://neo.datamat.it:5678/tmp/cms_sim.exe ",
                    "file:///tmp/myconf"
                   }; // inherited by all j*.jdl jobs
```

```
    InputSandboxBaseURI = "gsiftp://matrix.datamat.it:5432/tmp";
    nodes = {
                [
                 NodeName = "chkpt_job";
                 JobType = "Checkpointable";
                 Executable = "hsub.exe";
                 JobSteps = 10000000;
                 CurrentStep = 1;
                 Arguments = "gsiftp://lxde01.pd.infn.it/tmp/root_tst/";
                 RetryCount = 3;
                 InputSandbox = "file:///home/pippo";
                 rank = (-other.GlueCEStateEstimatedResponseTime);
                 requirements = other.GlueCEInfoLRMSType=="pbs";
                ],
                 [
                  file = "/home/sim_test/j1.jdl";
                 ],
                 [
                  file = "/home/sim_test/j2.jdl";
                 ],
                 [
                  file = "/home/sim_test/j3.jdl";
                 ],
                 [
                  file = "/home/sim_test/j4.jdl";
                 ]
          };
]
```

See also Example 5.