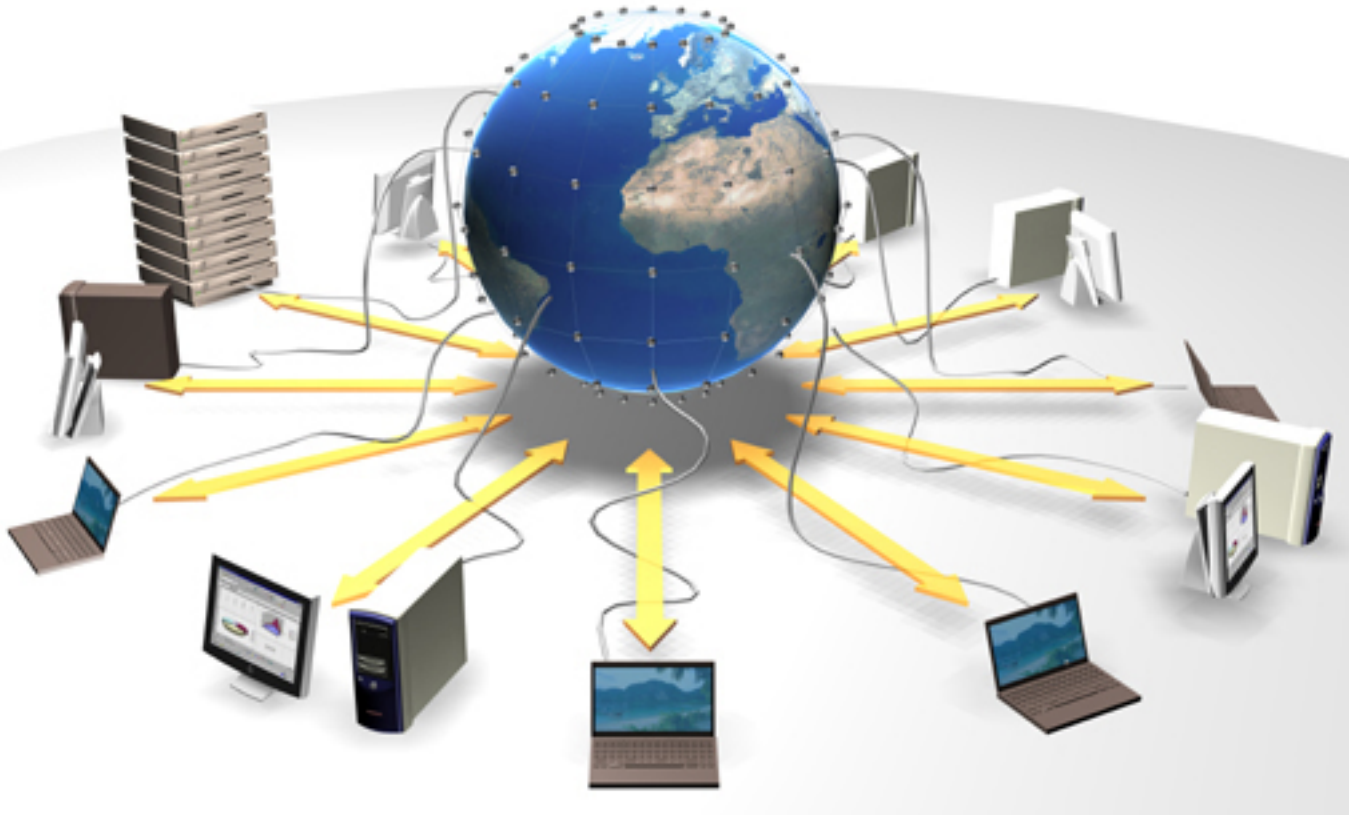
The EDG Workload Management System: User Interface

Fabrizio Pacini
Datamat SpA - EDG WP1

fabrizio.pacini@datamat.it



Overview

- JDL
- Command Line Interface
- API (C++ & JAVA)
- GUI

JDL (1/8)

Type

"Job" | "DAG" | "Reservation" | "Co-allocation" (only "Job" supported in Release 2.0)

JobType

"Normal"
"MPICH"
"Interactive"
"Checkpointable"
"Partitionable" (not yet supported in Release 2.0)

Executable

Arguments

StdInput

StdOutput

StdError

InputSandbox

OutputSandbox

-- new attributes are in red --

JDL (2 / 8)

Environment

RetryCount

MyProxyServer (triggers proxy-renewal mechanism)

HLRLocation *HLR contact* (triggers accounting)

- Format is "*host:port:HLRX509contactstring*"
- HLR is the service responsible for managing the economic transactions and the accounts of user and resources.
- After submission the UI contacts the user's HLR and authorizes the payment of that job.
- On the CE, while the job runs, a sensor system monitors the resource usage and when the job is done those data are sent to the User HLR.
- The User HLR computes the job cost according to the usage records and to the resource price and then debits the user account (if the job payment has been authorized by the user via the UI).

JDL (3 / 8)

Virtual Organisation — *VO name*

InputData — *List of LFNs and GUIDs*

E.g.: InputData = { "lfn:mytestfile" , "guid:135b7b23-4a6a-11d7-87e7-9d101f8c8b70" };

- Logical Collection LCN not yet supported in rel 2.0 by WP2.
- The user is not allowed any longer to specify a SFN (Storage File Name).
- Through the Requirement field (see **anymatch** function later) the user can select to run close to a SE with an announced available storage big enough for the user to replicate selected input files at run time from a remote SE to the one close for further processing.

DataAccessProtocol

OutputSE

ReplicaCatalog — no more needed - removed

JDL (4/8)

OutputData (triggers output files upload and registration)

OutputFile

StorageElement

LogicalFileName

- It allows the user to ask for the automatic upload and registration of output files produced by the job on the WN.
- It allows to indicate for each output file the LFN to be used for registration and the SE on which the file has to be uploaded. These entities are represented by the new attributes *OutputFile, StorageElement, LogicalFileName.*
- Both LFN and SE are optional
  - if no LFN is indicated then it is assigned automatically by the WP2 services (ERM)
  - if no SE is indicated, the close SE is considered.
- *OutputData* is a list of classads, where each classad indicates the name of the file to be uploaded, the logical file to be used and the SE where the file has to be copied. E.g.:

JDL (4/8)


```
OutputData = {
[
        OutputFile = "dataset_1.out ";
        StorageElement = "SE_1";
        LogicalFileName = "LFN_1"
],
[
        OutputFile = "dataset_2.out ";
        StorageElement = "SE_2";
],
[
        OutputFile = "dataset_3.out ";
        LogicalFileName = "LFN_3"
 ]
            };
```

JDL (5 / 8)

- if *OutputData* is found in the JDL then the *JobWrapper* at the end of the job calls the WP2 *copyAndRegister* service that copies the file from the WN onto the specified SE and registers it with the given LFN.
- the *JobWrapper* creates a file named "*DSUpload_<unique_jobid_string>.out*" that is put automatically in the *OutputSandbox* attribute list by the UI and can then be retrieved by the user.
- This file contains the results of the upload and registration process in the following format:

<file_name>	**<lfn | Error>**
dataset_1.out	LFN_1
dataset_2.out	GUID_2
dataset_3.out	<error msg>

JDL (6 / 8)

Rank

- other.dataAccessCost triggers interaction with WP2 Optimisation service API. CE are ranked according to the cost for accessing InputData provided by WP2 getAccessCost API.

  (Rank = other.dataAccessCost;)

Requirements

- anyMatch(other.storage.localSEs , target.GlueSEFreeSpace > *200*)
- anymatch is a Classad function using the gang-matching feature
- it allows asking for a CE having a close SE with a given amount of free space (200 MB in the example above)

JDL (7 / 8)

Job type related attributes

- MPICH:

NodeNumber — specifies the number of nodes needed for a MPI job

It is mandatory for MPICH jobs

- Checkpointable:

JobSteps — specifies the list of steps for a checkpointable job.

E.g.:

JobSteps = {"step1", "step2", ...,"stepn"};

JobSteps = 100000;

- Interactive:

ListenerPort — specifies the port on which the condor shadow process listens for the job standard streams

JDL (8 / 8)

Other small changes:

- [/] should be put at begin/end of JDL (not mandatory)
- JDL case insensitive
- Attributes spanning more lines (\ no more needed to go to new line)
- Slightly change comments formats (comment start either with // or #)

Command Line Interface (1/8)

edg-job-submit [options] <jdl file>

Options:

--help, --version

--input, -i <input file path>

--resource, -r <CE Identifier>

--hours, -h <hours num>

--nomsg

--output, -o <output file path>

--noint

--debug

--logfile <log file path>

--config, -c <config file path>

--vo <vo name>

--config-vo <config-vo file path>

--chkpt <chkpt file path>

Command Line Interface (2/8)

edg-job-get-output [options] <job Id(s)>

Options:

--help

--version

--input, -i <input file path>

--dir <dir path>

--config, -c <config file path>

--noint

--debug

--logfile <log file path>

Command Line Interface (3/8)

edg-job-status [options] <job Id(s)>

Options:

--help,	--version
--all	
--input, -i	<input file path>
--output, -o	<output file path>
--noint	
--debug	
--logfile	<log file path>
--config, -c	<config file path>
--verbosity, -v	<verbosity value>
--vo	<vo name>
--config-vo	<config-vo file path>

Command Line Interface (4/8)

edg-job-get-logging-info [options] <job Id(s)>

Options:

--help

--version

--input, -i	<input file path>
--config, -c	<config file path>
--output, -o	<output file path>
--noint	
--debug	
--logfile	<log file path>
--verbosity, -v	<verbosity value>

Command Line Interface (5/8)

edg-job-cancel [options] <job Id(s)>

Options:

--help

--version

--all

--input, -i <input file path>

--config, -c <config file path>

--output, -o <output file path>

--noint

--debug

--logfile <log file path>

--config-vo <config-vo file path>

--vo <vo name>

Command Line Interface (6/8)

edg-job-list-match [options] <jdl file>

Options:

--help

--version

--verbose

--config, -c <config file path>

--output, -o <output file path>

--noint

--debug

--logfile <log file path>

--rank

--config-vo <config-vo file>

--vo <vo name>

Command Line Interface (7/8)

edg-job-attach [options] <job Id>

Options:

--help

--version

--config, -c <config file path>

--port, -p <port number>

--output, -o <output file path>

--noint

--debug

--logfile <log file path>

Command Line Interface (8/8)

edg-job-get-chkpt [options] <job Id>

Options:

--help

--version

--config, -c <config file>

--output, -o <output file path>

--noint

--debug

--logfile <log file path>

--cs <chkpt state num>

API (1/5)

- Provided classes are:
  - JobId — The **JobId** class provides a representation of the Datagrid job identifier (edg_jobId) and the methods for manipulating it
  - JobAd — Provides a representation of the job in the JDL language and the functions for building and manipulating it. It provides some helper methods that ease the construction of job descriptions being fully compliant to WP1 WMS specification.
  - Job — The **Job** class is the actual EDG job representation. Allows creating the job and controlling it during its whole lifetime.

API (2/5)

- JobCollection — Container class for Job objects that has the main purpose of allowing the execution of collective operations on sets of independent jobs. The JobCollection class is just a logical container and both not yet submitted and already submitted jobs can be inserted in it.
- UserJobs — It allows performing monitoring and control operation on “all jobs owned by a user identified by its certificate”
- UserCredential — It provides methods for managing user credentials

API (3/5)

- Shadow — It allows managing interactive jobs. It starts a daemon listening for the job standard streams coming from the WN (the bypass console shadow). It creates the named pipes where the job std streams are forwarded.
- Listener — It is an abstract class which, given a Shadow class instance, allows the user to interact graphically with the interactive job standard streams.
- JobState — Representation of a job checkpoint state

API (4/5)

Some methods of the **Job** class:

Job(JobAd ad)	Instantiates a Job object with a JobAd
Job(JobId id)	Instantiates a Job object with a JobId
JobAd getJobAd()	Get the JobAd instance
JobId getJobId()	Get the JobId instance
retrieveJobAd()	Set the JobAd member attribute of the Job instance to the job description got from the LB
getLogInfo()	Retrieve the logging information of the job
getStatus()	Retrieve the job status information

API (5/5)

Some methods of the **Job** class:

listMatchingCE(Url nsAddr) Look for matching Computing Element available resources

submit(Url ns, Url lb, String ceId) Submit the job to the NS

submit(Url ns, Url lb, String ceId, Listener ls, JobState state) Submit for interactive/checkpointable jobs

getOutput(String dirPath) Retrieve output files of a submitted job

cancel() Cancel the job from the resource broker

GUI

- Java based GUI
- Relies on the mentioned java API
- Three graphical components
  - JDL Editor — Allows building Job description in JDL. Applet version integrated with GENIUS.
  - JobSubmitter — Allows submission of multiple jobs. Support multiple NS for a given VO. All job types are supported.
  - JobMonitor — Allows monitoring and control of submitted jobs. Provides job status and logging information. Allows retrieving the job output and job cancellation.