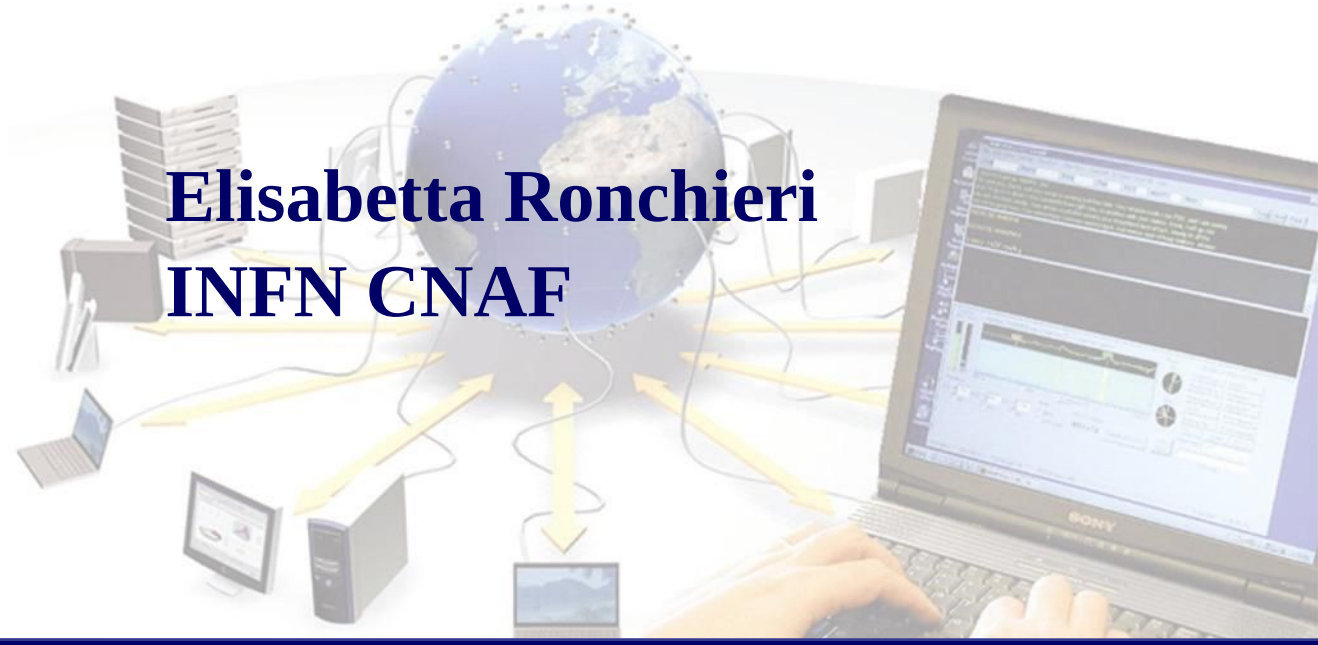
Enabling Grids for E-science in Europe

www.eu-egee.org

First Latinamerican Grid Workshop
17 November 2004

Job Services



Elisabetta Ronchieri
INFN CNAF

EGEE is a project funded by the European Union under contract IST-2003-508833

Contents

- The Workload Management System
- Job Preparation
  - Job Description Language
- Job submission and job status monitoring
- WMS Matchmaking
- Different job types
  - Interactive jobs
  - Checkpointable jobs
  - MPI jobs
  - DAG jobs
- APIs Overview

EGEE/LCG Workload Management System

Enabling Grids for E-science in Europe

- The user interacts with Grid via a **Workload Management System (WMS)**
- The Goal of WMS is the **distributed scheduling and resource management in a Grid environment**.
- What does it allow Grid users to do?
  - To submit their jobs
  - To execute them on the “best resources”
    - The WMS tries to optimize the usage of resources
  - To get information about their status
  - To retrieve their output

Job Preparation

- Information to be specified when a job has to be submitted:
  - Job characteristics
  - Job requirements and preferences on the computing resources
    - Also including software dependencies
  - Job data requirements
- Information specified using a Job Description Language (JDL)
  - Based upon Condor's *CLASSified ADvertisement language (ClassAd)*
    - Fully extensible language
    - A ClassAd
      - Constructed with the classad construction operator []
      - It is a sequence of attributes separated by semi-colon (;).
- So, the JDL allows definition of a set of attribute, the WMS takes into account when making its scheduling decision

Job Preparation

- An attribute is a pair (key, value), where value can be a Boolean, an Integer, a list of strings, ...
  - <attribute> = <value>;
- In case of literal string for values:
  - if a string itself contains double quotes, they must be escaped with a backslash
    - **`Arguments = " \"Hello\" 10";`**
  - the character “” cannot be specified in the JDL
  - special characters such as &, |, >, < are only allowed
    - if specified inside a quoted string
    - if preceded by triple \
      - **`Arguments = "-f file1\\\&file2";`**
- Comments must be preceded by a sharp character (#) or have to follow the C++ syntax
- The JDL is sensitive to blank characters and tabs
  - they should not follow the semicolon (;) at the end of a line

Job Description Language

- The supported attributes are grouped in two categories:
  - **Job Attributes**
    - Define the job itself
  - **Resources**
    - Taken into account by the RB for carrying out the matchmaking algorithm (to choose the “best” resource where to submit the job)
    - *Computing Resource*
      - Used to build expressions of Requirements and/or Rank attributes by the user
      - Have to be prefixed with “other.”
    - *Data and Storage resources (see talk Job Services With Data Requirements)*
      - Input data to process, SE where to store output data, protocols spoken by application when accessing SEs

JDL: Relevant attributes

Enabling Grids for E-science in Europe

- **JobType**
  - *Normal* (simple, sequential job), *Interactive*, *MPICH*, *Checkpointable*
  - Or combination of them
- **Executable** (mandatory)
  - The command name
- **Arguments** (optional)
  - Job command line arguments
- **StdInput**, **StdOutput**, **StdError** (optional)
  - Standard input/output/error of the job
- **Environment (optional)**
  - List of environment settings
- **InputSandbox** (optional)
  - List of files on the UI local disk needed by the job for running
  - The listed files will automatically staged to the remote resource
- **OutputSandbox** (optional)
  - List of files, generated by the job, which have to be retrieved
- **VirtualOrganisation** (optional)
  - A different way to specify the VO of the user

JDL: Relevant attributes

- **Requirements**
  - Job requirements on the resources
  - Specified using GLUE attributes of resources published in the Information Service
  - Its value is a boolean expression
  - Only one requirements can be specified
    - if there are more than one, only the last one is taken into account
  - If not specified, default value defined in UI configuration file is considered
    - Default: *other.GlueCEStateStatus == "Production"* (the resource has to be able to accept jobs and dispatch them on WNs)

JDL: Relevant attributes

- **Requirements**
  - Other possible requirements values are below reported:
    - *other.GlueCEInfoLRMSType == "PBS" && other.GlueCEInfoTotalCPUs > 1* (the resource has to use PBS as the LRMS and whose WNs have at least two CPUs)
    - *Member("CMSIM-133", other.GlueHostApplicationSoftwareRunTimeEnvironment)* (a particular experiment software has to run on the resource and this information is published on the resource environment)
      - The *Member* operator tests if its first argument is a member of its second argument
    - *RegExp("cern.ch", other.GlueCEUniqueId)* (the job has to run on the CEs in the domain cern.ch)
    - *(other.GlueHostNetworkAdapterOutboundIP == true) && Member("VO-alice-Alien", other.GlueHostApplicationSoftwareRunTimeEnvironment) && Member("VO-alice-Alien-v4-01-Rev-01", other.GlueHostApplicationSoftwareRunTimeEnvironment) && (other.GlueCEPolicyMaxWallClockTime > 86000)* (the resource must have some packages installed VO-alice-Alien and VO-alice-Alien-v4-01-Rev-01 and the job has to run for more than 86000 seconds)

JDL: Relevant attributes

Enabling Grids for E-science in Europe

- **Rank**
  - Expresses preference (how to rank resources that have already met the Requirements expression)
  - It is expressed as a floating-point number
  - The CE with the highest rank is the one selected
  - Specified using GLUE attributes of resources published in the Information Service
  - If not specified, default value defined in the UI configuration file is considered
    - Default: - *other.GlueCEStateEstimatedResponseTime* (the lowest estimated traversal time)
    - Default: *other.GlueCEStateFreeCPUs* (the highest number of free CPUs)
  - Other possible rank value is below reported:
    - *(other.GlueCEStateWaitingJobs == 0 ? other.GlueCEStateFreeCPUs : -other.GlueCEStateWaitingJobs)* (the number of waiting jobs is used if this number is not null and the rank decreases as the number of waiting jobs gets higher; if there are not waiting jobs, the number of free CPUs is used)

Essential JDL

- **At least one has to specify the following attributes:**
  - the name of the executable
  - the files where to write the standard output and standard error of the job
  - the arguments to the executable, if needed
  - the files that must be transferred from UI to WN and viceversa

```
[
Executable = “ls -al”;
StdError = “stderr.log”;
StdOutput = “stdout.log”;
OutputSandbox = {“stderr.log”, “stdout.log”};
]
```

Example of JDL file

```
[
JobType = “Normal”;
Executable = "$(CMS)/exe/sum.exe";
InputSandbox = {"/home/user/WP1testC","/home/file*”,
"/home/user/DATA/*"};
OutputSandbox = {“sim.err”, “test.out”, “sim.log"};
Requirements  = (other. GlueHostOperatingSystemName
== “linux") && (other.GlueCEPolicyMaxWallClockTime >
10000);
Rank = other.GlueCEStateFreeCPUs;
]
```

Job Submission

edg-job-submit [-r <*res_id*>] [-c <*config file*>] [-vo <*VO*>] [-o <*output file*>] <*job.jdl*>

-r the job is submitted directly to the computing element identified by <*res_id*>

-c the configuration file <*config file*> is pointed by the UI instead of the standard configuration file

-vo the Virtual Organisation (if user is not happy with the one specified in the UI configuration file)

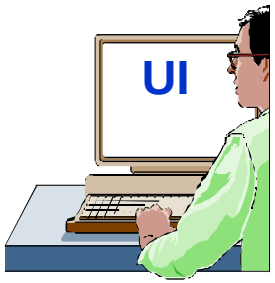
-o the generated edg_jobId is written in the <*output file*>

Useful for other commands, e.g.:

edg-job-status -i <*input file*> (or edg_jobId)

-i the status information about edg_jobId contained in the <*input file*> are displayed

Job Submission



Network Server

Workload Manager

Job Contr. CondorG

RB node

RLS

Inform. Service

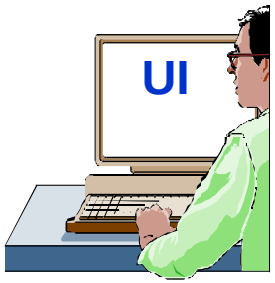
CE characts & status

SE characts & status

Computing Element

Storage Element

Job Submission



UI: allows users to access the functionalities of the WMS
(via command line, GUI, C++ and Java APIs)

Network Server

Workload Manager

Job Contr. CondorG

RB node

RLS

Inform. Service

Job Status

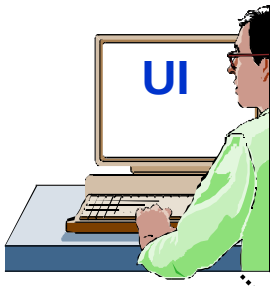
CE characts & status

SE characts & status

Computing Element

Storage Element

Job Submission



Network Server

RB node

Workload

RLS

Inform. Service

Job Status

submitted

edg-job-submit myjob.jdl

Myjob.jdl

JobType = “Normal”;
Executable = "$(CMS)/exe/sum.exe";
InputSandbox = {"/home/user/WP1testC","/home/file”, "/home/user/DATA/*"};*
OutputSandbox = {“sim.err”, “test.out”, “sim.log"};
Requirements = other. GlueHostOperatingSystemName == “linux" &&
other.GlueCEPolicyMaxWallClockTime > 10000;
Rank = other.GlueCEStateFreeCPUs;

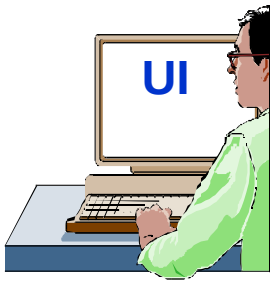
Job Description Language (JDL) to specify job characteristics and requirements

& status

Computing Element

Storage Element

Job Submission



Job

Input Sandbox files

Network Server

RB node

Workload Manager

RB storage

Job Contr. CondorG

Inform. Service

NS: network daemon responsible for accepting incoming requests

Job Status

submitted

waiting

CE charact & status

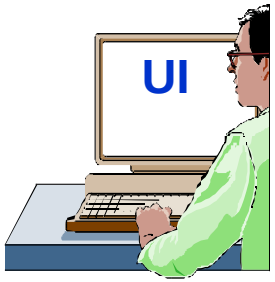
SE charact & status

Computing Element

Storage Element

Job Submission

Enabling Grids for E-science in Europe



Network Server

Workload Manager

RB storage

Job Contr. CondorG

RB node

RLS

Inform. Service

Job Status

submitted

waiting

WM: responsible to take the appropriate actions to satisfy the request

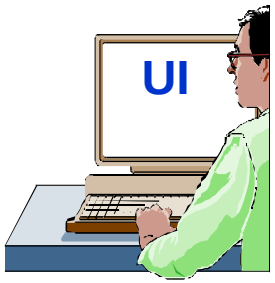
CE characts & status

SE characts & status

Computing Element

Storage Element

Job Submission



RB node

Network Server

Match-Maker/ Broker

Workload Manager

RB storage

Job Contr. CondorG

Where must this job be executed ?

RLS

Inform. Service

CE characts & status

SE characts & status

Computing Element

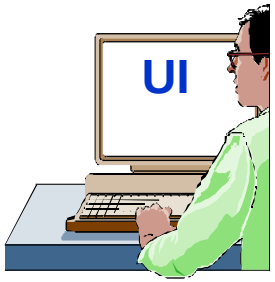
Storage Element

Job Status

submitted

waiting

Job Submission



RB node

Network Server

Match-Maker/ Broker

Workload Man

RB storage

Job Cont. CondorG

RLS

Inform. Service

Job Status

submitted

waiting

Matchmaker: responsible to find the “best” CE where to submit a job

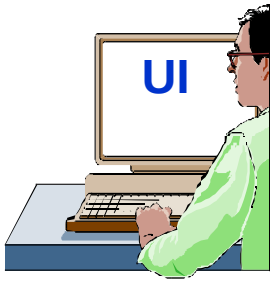
CE charact & status

SE charact & status

Computing Element

Storage Element

Job Submission



RB node

Network Server

Workload Manager

RB storage

Job Contr. CondorG

Match-Maker/ Broker

RLS

Inform. Service

Job Status

submitted

waiting

What is the status of the Grid ?

CE characts & status

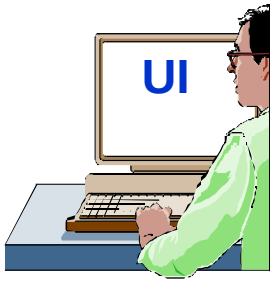
SE characts & status

Computing Element

Storage Element

Where are (which SEs) the needed data ?

Job Submission



RB node

Network Server

Match-Maker/ Broker

Workload Manager

CE choice

RB storage

Job Contr. CondorG

RLS

Inform. Service

Job Status

submitted

waiting

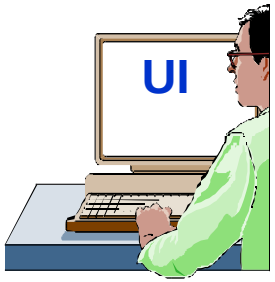
CE characts & status

SE characts & status

Computing Element

Storage Element

Job Submission



RB node

Network Server

Workload Manager

Job Adapter

Job Contr. CondorG

RB storage

RLS

Inform. Service

Job Status

submitted

waiting

CE characts

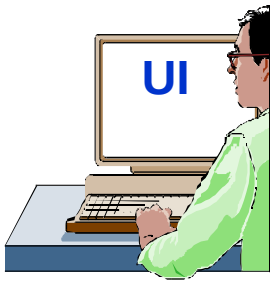
SE characts & status

JA: responsible for the final "touches" to the job before performing submission (e.g. creation of wrapper script, etc.)

Computing Element

Storage Element

Job Submission



RB node

Network Server

Workload Manager

RB storage

Job Contr. CondorG

JC: responsible for the actual job management operations (done via CondorG)

Computing Element

RLS

Inform. Service

CE characts & status

SE characts & status

Storage Element

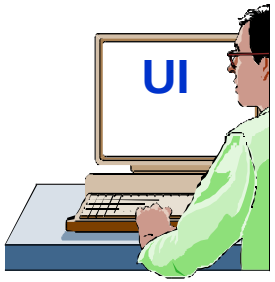
Job Status

submitted

waiting

ready

Job Submission



RB node

Network Server

Workload Manager

RB storage

Job Contr. CondorG

Input Sandbox files

Job

Computing Element

RLS

Inform. Service

CE characts & status

SE characts & status

Storage Element

Job Status

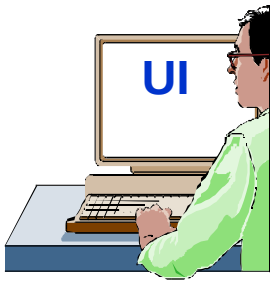
submitted

waiting

ready

scheduled

Job Submission



RB node

Network Server

Workload Manager

RB storage

Job Contr. CondorG

RLS

Inform. Service

Job Status

submitted

waiting

ready

scheduled

running

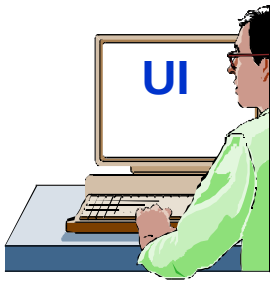
Computing Element

Job

"Grid enabled" data transfers/ accesses

Storage Element

Job Submission



RB node

Network Server

Workload Manager

RB storage

Job Contr. CondorG

Output Sandbox files

Computing Element

RLS

Inform. Service

Storage Element

Job Status

submitted

waiting

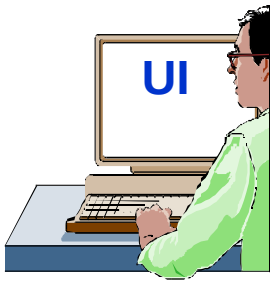
ready

scheduled

running

done

Job Submission



edg-job-get-output <dg-job-id>

RB node

Network Server

Workload Manager

RB storage

Job Contr. CondorG

RLS

Inform. Service

Computing Element

Storage Element

Job Status

submitted

waiting

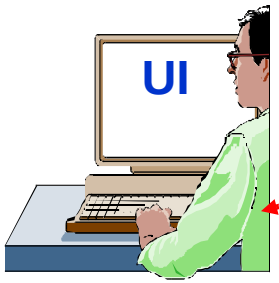
ready

scheduled

running

done

Job Submission



Output Sandbox files

Network Server

Workload Manager

RB storage

Job Contr. CondorG

RB node

RLS

Inform. Service

Computing Element

Storage Element

Job Status

submitted

waiting

ready

scheduled

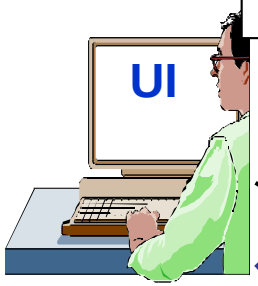
running

done

cleared

Job monitoring

edg-job-status <dg-job-id>
edg-job-get-logging-info <dg-job-id>



LB: receives and stores job events; processes corresponding job status

Job status

Network Server

Workload Manager

Logging & Bookkeeping

Job Contr. CondorG

Log Monitor

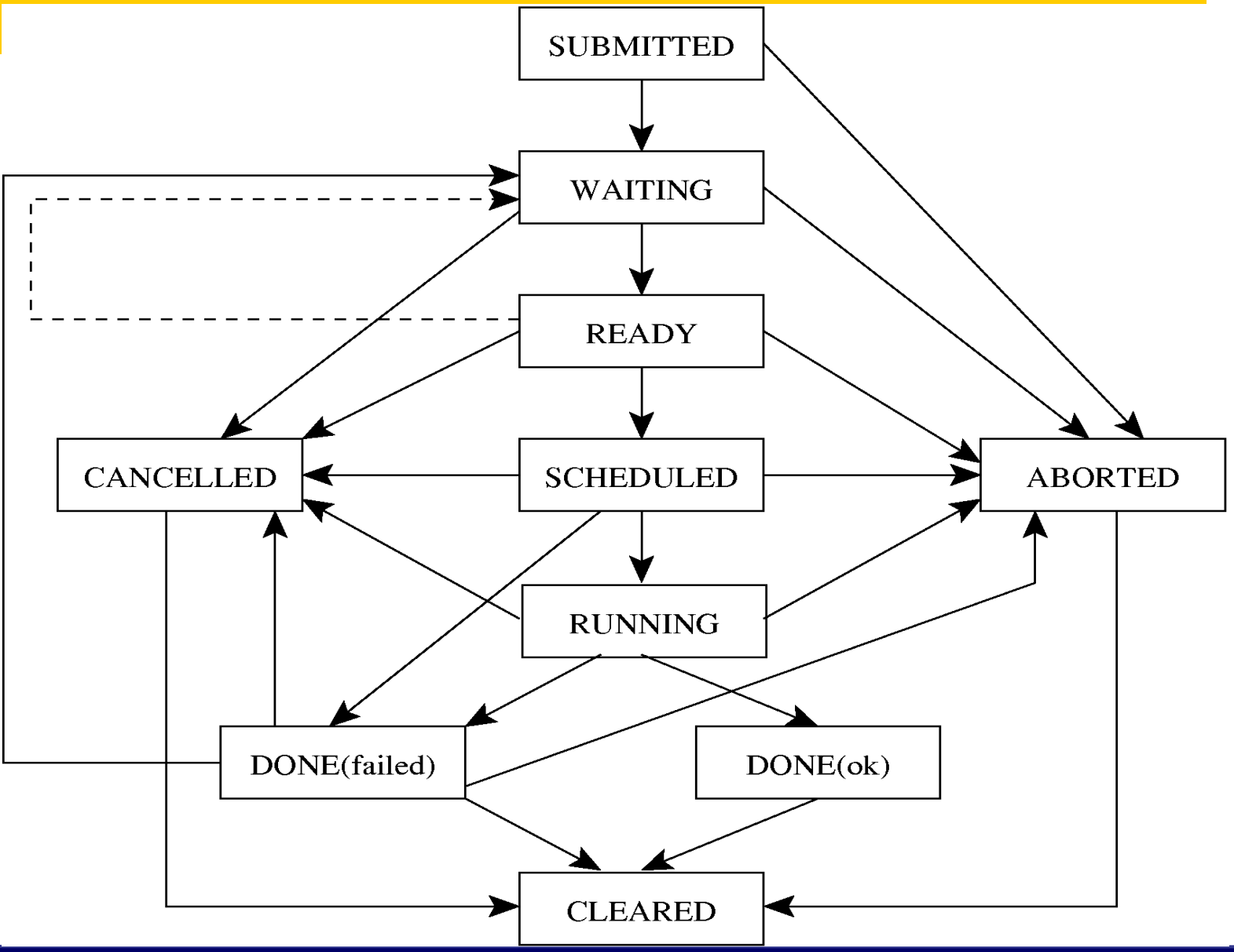
RB node

Log of job events

LM: parses CondorG log file (where CondorG logs info about jobs) and notifies LB

Computing Element

Possible job states



Job resubmission

- If something goes wrong, the WMS tries to reschedule and resubmit the job (possibly on a different resource satisfying all the requirements)
- Maximum number of resubmissions: min (RetryCount, MaxRetryCount)
  - RetryCount: JDL attribute
  - MaxRetryCount: attribute in the “RB” configuration file
- e.g., to disable job resubmission for a particular job: *RetryCount=0;* in the JDL file

Other (most relevant) UI commands

Enabling Grids for E-science in Europe

- **edg-job-list-match**
  - Lists resources matching a job description
  - Performs the matchmaking without submitting the job
- **edg-job-cancel**
  - Cancels a given job
- **edg-job-status**
  - Displays the status of the job
- **edg-job-get-output**
  - Returns the job-output (the OutputSandbox files) to the user
- **edg-job-get-logging-info**
  - Displays logging information about submitted jobs (all the events “pushed” by the various components of the WMS)
  - Very useful for debug purposes

The Matchmaking algorithm

Enabling Grids for E-science in Europe

- The matchmaker has the goal to find the best suitable CE where to execute the job
- To accomplish this task, the WMS interacts with the other EGEE/LCG components (Replica location Service, and Information Service)
- There are three different scenarios to be dealt with separately:
  - Direct job submission
  - Job submission without data-access requirements
  - Job submission with data-access requirements (*see talk Job Services With Data Requirements*)

The Matchmaking algorithm: direct job submission

Enabling Grids for E-science in Europe

- The user JDL contains a link to the resource to submit the job
- The WMS does not perform any matchmaking algorithm at all
- The job is simply submitted to the specified CE

IMPORTANT:

- If the CEId is specified then the WMS
  - neither checks whether the user who submitted the job is authorised to access the given CE, nor interacts with the RLS for the resolution of files requirements, if any
  - Only checks the JDL syntax, while converting the JDL into a ClassAd
- The user run the edg-job-submit –resource <ce_id> <nome.jdl> command

  ce_id = hostaname:port/jobmanager-lsf-grid01

The Matchmaking algorithm: job submission without data access requirements

Enabling Grids for E-science in Europe

- The user JDL contains some requirements
- Once the JDL has been received by the WMS and converted in ClassAd, the WMS invokes the matchmaker
- The matchmaker has to find if the characteristics and status of Grid resources match the job requirements
- There are two phases:
  - Requirements check:
    - The Matchmaker contacts the GOUT/II in order to create a set of suitable CEs compliant with user requirements and where the user is authorized to submit jobs
    - The Matchmaker creates the set of suitable CEs
  - Ranking phase:
    - The Matchmaker contacts directly the LDAP (GRIS) server of the involved CEs to obtain the values of those attributes that are in the rank JDL expression

The Matchmaking algorithm: job submission without data access requirements

Enabling Grids for E-science in Europe

- The matchmaker can select a CE randomly, if there are two or more CEs that meet all the requirements and have the same rank
- In general, the CE with maximum rank value is selected

- IMPORTANT:
  - The CE attributes involved in the JDL requirements refers to static information
  - All the information cached in the IS represent a good source for matches among job requirements and CE features
  - In the first phase it is more efficient to contact the GOUT/II, than querying each CE
  - The rank attributes refers to variable varying in time very frequently
  - In the second phase it is more efficient to contact each suitable CE, rather than using the GOUT/II as source of information

The Matchmaking algorithm: job submission without data access requirements

Enabling Grids for E-science in Europe

- The matchmaker can adopt a stochastic selection while searching for the best matching CE, enabling fuzzyness in the matchmaking algorithm
- The user has to set the JDL FuzzyRank attribute to true
- The rank value represents the probability that each CE has to be selected as the best matching one
- The higher the probability is, the higher the rank value is

Interactive Job

- The Interactive job is a job whose standard streams are forwarded to the submitting client
- OutBound connectivity is required between UI and WN
- The user has to set the JDL JobType attribute to interactive
- When an interactive job is submitted, the edg-job-submit command
  - starts a Grid console shadow process in the background that listens on a port assigned by the Operating System
  - opens a new window where the incoming job streams are forwarded
- The DISPLAY environment variable has to be set correctly, because an X window is open
  - The generated X window shows Standard Error, Standard Output, Job Identifier
  - Via X window, the user can send Standard Input

Interactive Job

```
[
JobType = "Interactive";
Executable = "interactive.sh";
InputSandbox = "interactive.sh";
ListenerPort = 21000;
]
```

NOTE:

- The port can be forced through the ListenerPort attribute in the JDL
- It is not necessary to specify the OutputSandbox attribute in the JDL because the output will be sent to the interactive window

Interactive Job

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

interactive.sh

Standard Output

```
Welcome!
What is your name?
$ Elisabetta
Bye Bye Elisabetta.
**************************
*INTERACTIVE JOB FINISHED
**************************
```

- Presents a Welcome message to the user
- Asks and waits for an input (the user's name)
- The user's name is shown back
- The job finished

Interactive Job

- The user can specify some options:
  - --nogui
    - makes the command provide a simple standard non-graphical interaction with the running job
  - --nolisten
    - allows the user to interact with the job through her/his own tools
  - --noint
    - every interactive question to the user is skipped.
    - All warning messages and errors are written to the file edg-job-attach_<UID>_<PID>_<timestamp>.log file under /tmp directory as default

Logical Checkpointing Job

- The Checkpointing job is a job that can be decomposed in several steps
- In every step the job state can be saved in the LB and retrieved later in case of failures
- The job state is a set of pairs <key, value> defined by the user
- The job can start running from a previously saved state and not from the beginning again
- The user has to set the JDL JobType attribute to checkpointable

Logical Checkpointing Job

Enabling Grids for E-science in Europe

- When a checkpointable job is submitted and starts from the beginning, the user run simply the edg-job-submit command
  - the number of steps, that represents the job phases, can be specified by the JobSteps attribute
    - e.g. JobSteps = 2;
  - the list of labels, that represents the job phases, can be specified by the JobSteps attribute
    - e.g. JobSteps = {“genuary”, “february”};
- The latest job state can be obtained by using the edg-job-get-chkpt <jobid> command
- A specific job state can be obtained by using the edg-job-get-chkpt –cs <state_num> <jobid> command
- When a checkpointable job has to start from an intermediate job state, the user run the edg-job-submit command using the –chkpt <state_jdl> option where <state_jdl> is a valid job state file, where the state of a previously submitted job was saved

Other (most relevant) UI commands

- **edg-job-attach**
  - Starts an interactive session for previously submitted interactive jobs
  - Srarts a listener process on the UI machine
- **edg-job-get-chkpt**
  - Allows the user to retrieve one or more checkpoint states by a previously submitted job

MPI Job

- There are a lot of libraries supporting parallel jobs, but we decided to support MPICH.
- The MPI job is run in parallel on several processors
- The user has to set the JDL **JobType** attribute to MPICH and specify the **NodeNumber** attribute that's the required number of CPUs
- When a MPI job is submitted, the UI adds
  - in the **Requirements** attribute

    `Member("MpiCH", other.GlueHostApplicationSoftwareRunTimeEnvironment)` (the MPICH runtime environment must be installed on the CE)

    `other.GlueCEInfoTotalCPUs >= NodeNumber` (a number of CPUs must be at least be equal to the required number of nodes)
  - In the Rank attribute

    `other.GlueCEStateFreeCPUs` (it is chosen the CE with the largest number of free CPUs)

MPI Job

```
[
JobType = "MPICH";
NodeNumber = 4;
Executable = "MPItest.sh";
Argument = "cpi 4";
InputSandbox = {"MPItest.sh", "cpi"};
OutputSandbox = "executable.out";
Requirements = other.GlueCEInfoLRMSType == “PBS” ||
other.GlueCEInfoLRMSType == “LSF”;
]
```

- The NodeNumber entry is the number of threads of MPI job
- The MPItest.sh script only works if PBS or LSF is the local job manager
  - If you want to submit your MPI programs you have to compile them against MPICH library

MPI Job

```
[
JobType = "MPICH";
NodeNumber = 4;
Executable = "MPItest.sh";
Argument = "cpi 4";
InputSandbox = {"MPItest.sh", "cpi"};
OutputSandbox = "executable.out";
Requirements = other.GlueCEInfoLRMSType == “PBS” ||
other.GlueCEInfoLRMSType == “LSF”;
]
```

- The first argument cpi is the binary to be executed
- The second one 4 represents the number of CPUs to be reserved for parallel execution
- The MPItest.sh script sets the environment HOST_NODEFILE
  - the path of a file that contains the list of WNs allocated for parallel execution

MPI Job: MPITest.sh

Enabling Grids for E-science in Europe

```
#!/bin/sh
#
# this parameter is the binary to be
   executed
EXE=$1
# this parameter is the number of
   CPU's to be reserved for parallel
   execution
CPU_NEEDED=$2
# prints the name of the master node
echo "Running on: $HOSTNAME"
echo "********************************"
if [ -f "$PWD/.BrokerInfo" ] ; then
TEST_LSF=`edg-brokerinfo getCE | cut
   -d/ -f2 | grep lsf`
else
TEST_LSF=`ps -ef | grep sbatchd | grep
   -v grep`
fi
if [ "x$TEST_LSF" = "x" ] ; then
# prints the name of the file containing
   the nodes allocated for parallel
   execution
echo "PBS Nodefile: $PBS_NODEFILE"
# print the names of the nodes .....
cat $PBS_NODEFILE
echo "************************************"
HOST_NODEFILE=$PBS_NODEFILE
else
# print the names of the nodes ......
echo "LSF Hosts: $LSB_HOSTS"
# loops over the nodes allocated for
   parallel execution
HOST_NODEFILE=`pwd`/lsf_nodefile.$$
for host in ${LSB_HOSTS}
do
echo $host >> ${HOST_NODEFILE}
done
fi
```

MPI Job: MPITest.sh

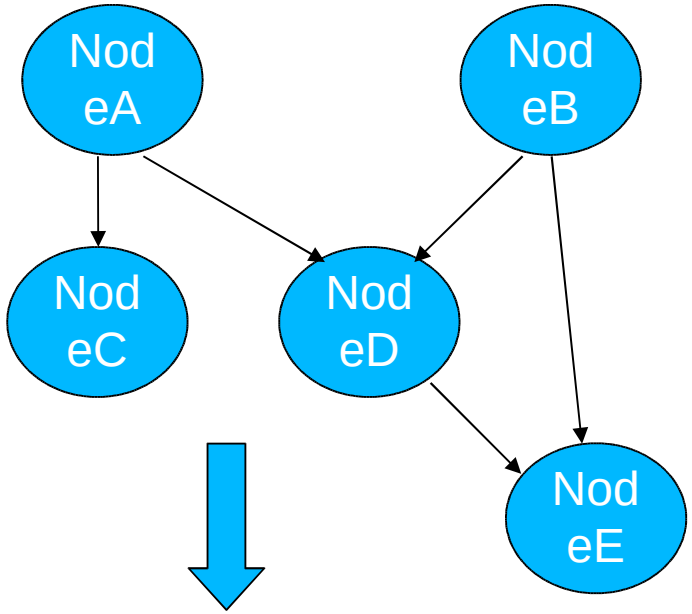
```
echo "********************************"
# prints the working directory on the
   master node
echo "Current dir: $PWD"
echo "********************************"
for i in `cat $HOST_NODEFILE` ; do
echo "Mirroring via SSH to $i"
# creates the working directories on
   all the nodes allocated for parallel
   execution
ssh $i mkdir -p `pwd`
# copies the needed files on all the
   nodes allocated for parallel
   execution
/usr/bin/scp -rp ./* $i:`pwd`
# checks that all files are present on
   all the nodes allocated for parallel
   execution
echo `pwd`
ssh $i ls `pwd`
# sets the permissions of the files
ssh $i chmod 755 `pwd`/$EXE
ssh $i ls -alR `pwd`
echo "@@@@@@@@@@@@@@@"
done
# execute the parallel job with mpirun
echo "********************************"
echo "Executing $EXE"
chmod 755 $EXE
ls -l
mpirun -np $CPU_NEEDED -machinefile
   $HOST_NODEFILE `pwd`/$EXE >
   executable.out
echo "********************************"
```

MPI Job: Output

```
Process 0 of 4 on grid022.ct.infn.it

pi is approximately 3.1415926544231239, Error is
  0.0000000008333307

wall clock time = 10.007429

Process 2 of 4 on grid020.ct.infn.it

Process 3 of 4 on grid026.ct.infn.it

Process 1 of 4 on grid021.ct.infn.it
```

What is a DAG

- DAG means Directed Acyclic Graph
- Each node represents a job
- Each edge represents a temporal dependency between two nodes
  - e.g. NodeC starts only after NodeA has finished
- A dependency represents a constraint on the time a node can be executed
  - Limited scope, it may be extended in the future
- Dedendencies are represented as “expression lists” in the ClassAd language




```
dependencies = {
{NodeA, {NodeC, NodeD}},
{NodeB, NodeD},
{NodeB, NodeD}, NodeE}
}
```

DAG Job

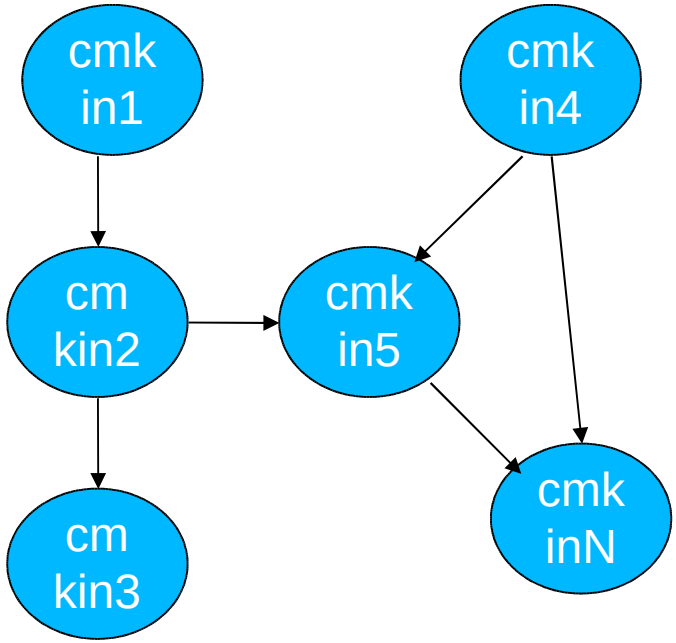
- The DAG job is a Directed Acyclic Graph Job
- The sub-jobs are scheduled only when the corresponding DAG node is ready
- The user has to set the JDL JobType attribute to dag, nodes attributes that contains the description of the nodes, and dependencies attributes

NOTE:

- A plug-in has been implemented to map an EGEE DAG submission to a Condor DAG submission
- Some improvements have been applied to the ClassAd API to better address WMS need

DAG Job

```
nodes = {
   cmkin1 = [
      file = “bckg_01.jdl" ;
   ],
   cmkin2 = [
      file = “bckg_02.jdl" ;
   ],
   ……
   cmkinN = [
      file = “bckg_0N.jdl" ;
   ]
};
dependencies = {
   {cmkin1, cmkin2},
   {cmkin2, cmkin3},
   {cmkin2, cmkin5},
   {{cmkin4, cmkin5}, cmkinN}
}
```



WMS APIs

Enabling Grids for E-science in Europe

- The WMS makes C++ and Java APIs available for UI, LB consumer and client.
- In the following document:

  http://server11.infn.it/workload-grid/docs/DataGrid-01-TEN-0118-1_2.pdf

  details about the rpms containing the APIs are given.
- Correspondent doxygen documentation can be found in share/doc area. Ex.:

  $EDG_LOCATION/share/doc/edg-wl-ui-api-cpp-lcg2.1.49/html

WMS APIs

Enabling Grids for E-science in Europe

% ./workload Hello.jdl lxb0704.cern.ch 7772 lxb0704.cern.ch 9000


```
#include <iostream>
#include <string>

#include "edg/workload/logging/client/JobStatus.h"
#include "edg/workload/common/utilities/Exceptions.h"
#include "edg/workload/common/requestad/JobAd.h"
#include "edg/workload/userinterface/client/Job.h"


using namespace std ;
using namespace edg::workload::common::utilities ;
using namespace edg::workload::logging::client ;
/* ************************************************************************
 *  Example based on edg-wl-job-submit.cpp, edg-wl-job-status.cpp
 *  for further examples see also:

 http://isscvs.cern.ch:8180/cgi-
bin/cvsweb.cgi/workload/userinterface/test/?cvsroot=lcgware

 *
 *  author: Heinz.Stockinger@cern.ch
 *
 *  Example usage on GILDA:
 *  ./workload Hello.jdl grid004.ct.infn.it 7772 grid004.ct.infn.it 9000
 *
 */
```

Additional examples in CVS

WMS APIs

Enabling Grids for E-science in Europe

```
int main (int argc,char *argv[])
{

 cout << "Workload Management API Example " << endl;

 try{
  if (argc < 6 ||  strcmp(argv[1],"--help") == 0)  {
    cout << "Usage : " << argv[0]
              << "  <JDL file>  <ns host> <ns port> <lbHost> <lbPort> [<ce_id>]"
              << endl;
    return -1;
  }

  edg::workload::common::requestad::JobAd jab;

  jab.fromFile ( argv[1] ) ;
  edg::workload::userinterface::Job   job(jab);
  job.setLoggerLevel (6) ;

  cout << "Submit job to " << argv[2] << ":" <<  argv[3]  << endl;
  cout << "LB address: "<< argv[4] << ":" <<  argv[5]  << endl;
  cout << "Please wait..." << endl;

  // We now submit the job. If a CE is given (argv[6]), we send it directly
  // to the specified CE
  //
  if (argc ==6)
    job.submit (argv[2], atoi(argv[3]), argv[4], atoi(argv[5]),  "") ;
  else
    job.submit (argv[2], atoi(argv[3]), argv[4], atoi(argv[5]), argv[6] ) ;

  cout << "Job Submission OK; JobID= "
            << job.getJobId()->toString()  << endl << flush ;
```

- The JobAd class provides users with management operations on JDL files
- We instantiate a Job object that corresponds to our JDL file and handles our job

WMS APIs

Enabling Grids for E-science in Europe

```
// Print some detailed error information in case the job did not
   // succeed.
   //
   if ((status.status == 8) || (status.status == 9)) {
     printStatus(status);
     exit(-1);
   }

   // Now that the job has successfully finished, we retrieve the output
   //
   string outputDir = "/tmp";
   job.getOutput(outputDir);

   cout << "\nThe output has been retrieved and stored in the directory "
            << outputDir << endl;

   return 0;

 } catch (Exception &exc){
   cerr << "\nWMS Error\n";
   cerr <<  exc.printStackTrace();
 }
 return -1;
}
```

The job finished successfully. We can retrieve the

WMS APIs

Enabling Grids for E-science in Europe


```
 CC          = gcc-3.2.2
GLOBUS_FLAVOR  = gcc32


ARES_LIBS =  -lares
BOOST_LIBS = -L/opt/boost/gcc-3.2.2/lib/release -lboost_fs \
                -lboost_thread -lpthread -lboost_regex
CLASSAD_LIBS = -L/opt/classads/gcc-3.2.2/lib -lclassad
EXPAT_LIBS =  -lexpat
GLOBUS_THR_LIBS =  -L/opt/globus/lib -lglobus_gass_copy_gcc32dbgpthr \
    -lglobus_ftp_client_gcc32dbgpthr -lglobus_gass_transfer_gcc32dbgpthr \
    -lglobus_ftp_control_gcc32dbgpthr -lglobus_io_gcc32dbgpthr \
    -lglobus_gss_assist_gcc32dbgpthr -lglobus_gssapi_gsi_gcc32dbgpthr \
    -lglobus_gsi_proxy_core_gcc32dbgpthr \
    -lglobus_gsi_credential_gcc32dbgpthr \
    -lglobus_gsi_callback_gcc32dbgpthr -lglobus_oldgaa_gcc32dbgpthr \
   -lglobus_gsi_sysconfig_gcc32dbgpthr \
   -lglobus_gsi_cert_utils_gcc32dbgpthr \
   -lglobus_openssl_gcc32dbgpthr -lglobus_proxy_ssl_gcc32dbgpthr \
   -lglobus_openssl_error_gcc32dbgpthr -lssl_gcc32dbgpthr \
   -lcrypto_gcc32dbgpthr -lglobus_common_gcc32dbgpthr

GLOBUS_COMMON_THR_LIBS =  -L/opt/globus/lib -L/opt/globus/lib \
                                -lglobus_common_gcc32dbgpthr
GLOBUS_SSL_THR_LIBS =  -L/opt/globus/lib -L/opt/globus/lib \
                            -lssl_gcc32dbgpthr -lcrypto_gcc32dbgpthr
VOMS_CPP_LIBS = -L/opt/edg/lib -lvomsapi_gcc32dbgpthr

all: workload
```

Makefile

```
workload: workload.o
        $(CC) -o workload \
        -L${EDG_LOCATION}/lib -ledg_wl_common_requestad \
                –lpthread \
                -ledg_wl_userinterface_client \
        -ledg_wl_exceptions -ledg_wl_logging \
                -ledg_wl_loggingpp \
        -ledg_wl_globus_ftp_util -ledg_wl_util \
                -ledg_wl_common_requestad \
        -ledg_wl_jobid -ledg_wl_logger -ledg_wl_gsisocket_pp \
        -ledg_wl_checkpointing -ledg_wl_ssl_helpers \
                -ledg_wl_ssl_pthr_helpers \
        $(VOMS_CPP_LIBS) \
        $(CLASSAD_LIBS) $(EXPAT_LIBS) $(ARES_LIBS) \
        $(BOOST_LIBS) \
        $(GLOBUS_THR_LIBS) \
        $(GLOBUS_COMMON_THR_LIBS) \
        $(GLOBUS_SSL_THR_LIBS) \
        workload.o

workload.o: workload.cpp
        $(CC) -I ${EDG_LOCATION}/include \
        -I/opt/classads/gcc-3.2.2/include -c workload.cpp

clean:
        rm -rf workload workload.o
```

Summary

- We explained the main functionality of the Workload Management System
- The JDL file describes a user job
- A set of commands allow the user to get status information and retrieve relevant data
- APIs are available in C++ and Java for UI, and LB.
- We exercized the UI C++ APIs