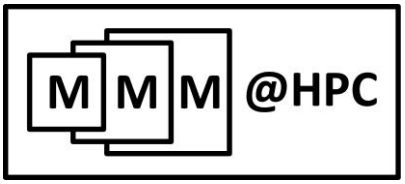
Date of publication: 08.06.2012

Document status: Submitted




Multiscale materials modelling on High Performance Computer Architectures

# DELIVERABLE D3.2

## Training Materials

**Work package WP3: NA3 User Training**

**Delivery date: M16**

**Dissemination level: PU**

**Type: Other**





The project MMM@HPC is funded by the 7th Framework Programme of the European Commission within the Research Infrastructures with grant agreement number RI-261594.

### Version history

| Revision | Date (DD/MM/ YYYY) | Created / Changed by (name) | Forwarded for changes to (name) | Description of changes | Editorial status | Comments |
|---|---|---|---|---|---|---|
| 1 | 02.03.2012 | Andrew Emerson | Ivan Kondov Peter Raback | Developed strategy, first draft | Draft | |
| 2 | 19.04.2012 | Ivan Kondov | Andrew Emerson | documentation for workflows, attached materials | Draft | |
| 3 | 31.05.2012 | Andrew Emerson | Ivan Kondov Peter Raback | Restrcutured, document according to class of student. Included feedback form, | Draft | |
| 4 | 04.06.2012 | Sampo Sillanpää, Sami Ilvonen | Ivan Kondov, Andrew Emerson | Added descriptions of CSC environment | Draft | |
| 5 | 07.06.2012 | Wolfgang Wenzel | Ivan Kondov | Added workflow description | Draft | Pls. add license circulate & submit |
| 6 | 08/06/2012 | Ivan Kondov | | Added license, spell check | Submitted | |
| | | | | | | |

Please keep all versions and send final versions to coordinator@multiscale-modelling.eu.

# Table of contents

# 1 Introduction

The aim of this deliverable is to provide training materials for courses and registered users using the MMM@HPC facilities and toolkit. For convenience, much of the text is collected together in a single document, although further documentation and manuals are provided separately (see the section **Fehler! Verweisquelle konnte nicht gefunden werden.**). It should also be pointed out that the material described here may be available in other formats, for example via the web (on-line training, feedback, animations, etc) or as CD ISO image. These sources are mentioned in the text when appropriate.

The material here is divided into 3 main sections according to the target group of users to be trained:

1. **End users, not expert in HPC**. This group may include industrial users or experimentalists with little experience in HPC, Grid computing or workflows. The documentation here will provide information on how to register for the service, apply for and use certificates, install the UNICORE client and launch pre-defined workflows.

2. **Molecular modellers**. This section is aimed at molecular modellers who understand the physical models and protocols and can perform simulations HPC in a "classical" way via interactive access. They will be taught to construct, develop and support the workflows.

3. **Grid Bean developers**. Here instead the target group are those who wish to develop new Grid Beans for use within the MMM@HPC Development Toolkit.

In addition there is also a 4th section containing supplementary material which can be used by all 3 groups of users, for example guidelines for interactive (i.e. non-UNICORE) access, or items which can be of use to trainers themselves, such as an example feedback form.

# 2 Training materials for non-expert users

| |
|---|
| **Target group**: Researchers with no or little experience in HPC and UNICORE workflows: |
| **Pre-requisities for students**: Possession of a valid X.509 certificate. |
| **Description of material**: Step-by-step procedure on how to install the UNICORE Rich Client and launch simple scripts and existing workflows. |
| **Attached Supplementary files:** None |

## Registering to use the services

Before a user can get started with the MMM@HPC infrastructure he/she will have to follow one or more administrative procedures depending on what services are required. Not all requirements are obligatory, for example currently a new user is not required to have a X.509 certificate to use CINECA's computer systems via a UNIX login, but a certificate is required for UNICORE services. In the following we list some of the common services provided and the administrative steps to be followed:

- Access to the CINECA infrastructure via the UNIX interface (i.e. SSH). As explained in the section "**Fehler! Verweisquelle konnte nicht gefunden werden.**" and ... a new user will need to register with the CINECA userdb (user database) and should indicate MMM@HPC as

the project. Once authorised the credentials (i.e. username and password) will be sent to the email address supplied.

- Access to the CSC infrastructure. To apply an academic user account to the CSC infrastructure, you can print out necessary application forms in pdf format from http://www.csc.fi/english/customers/university/useraccounts/scientificservices.pdf. Then, fill out the application and send it to CSC's postal address: CSC - IT Center for Science, User Accounts, P.O.Box 405, FI - 02101 Espoo, Finland. If you are obtaining an account through an MMM@HPC project, you should mention this in the application form.

- UNICORE services. To use UNICORE a user must

    1. Obtain a personal X.509 certificate

    2. Apply for a user account at one or both of the HPC centres, currently CSC or CINECA, as described above.

The procedure for obtaining an X.509 certificate varies according to country of residence – contact your local computer services department or computer centre for further information.

## Using UNICORE and workflows

***Preparing the certificates***

A personal X.509 certificate is required to use UNICORE, regardless of whether a user has interactive access to the HPC Infrastructure or not. Normally, certificates issued in such a way that they are stored in the browser so the first step is to export the certificate from the browser into the hard drive of the personal computer. This procedure is browser dependent but essentially the procedure is the same. For example from Firefox:

```
Options -> Advanced -> Encryption -> View Certificates
```

Then select the certificate and click Backup. You will be asked to provide a backup password and this should remember.

In this way it will be possible to store your certificate in .p12 format which is suitable for the UNICORE client.

As well as your own personal certificate you will also need the set of trusted CA certificates, the so-called “Trust store” which can be downloaded, for example, from this link:

http://winnetou.sara.nl/deisa/certs/keystore.jks

This “jks” file should be stored somewhere on your computer since it will be needed to start-up the UNICORE client.

***Installing and Starting up the UNICORE Rich Client***

The UNICORE Rich Client (URC) is freely available by following the instructions on this website:

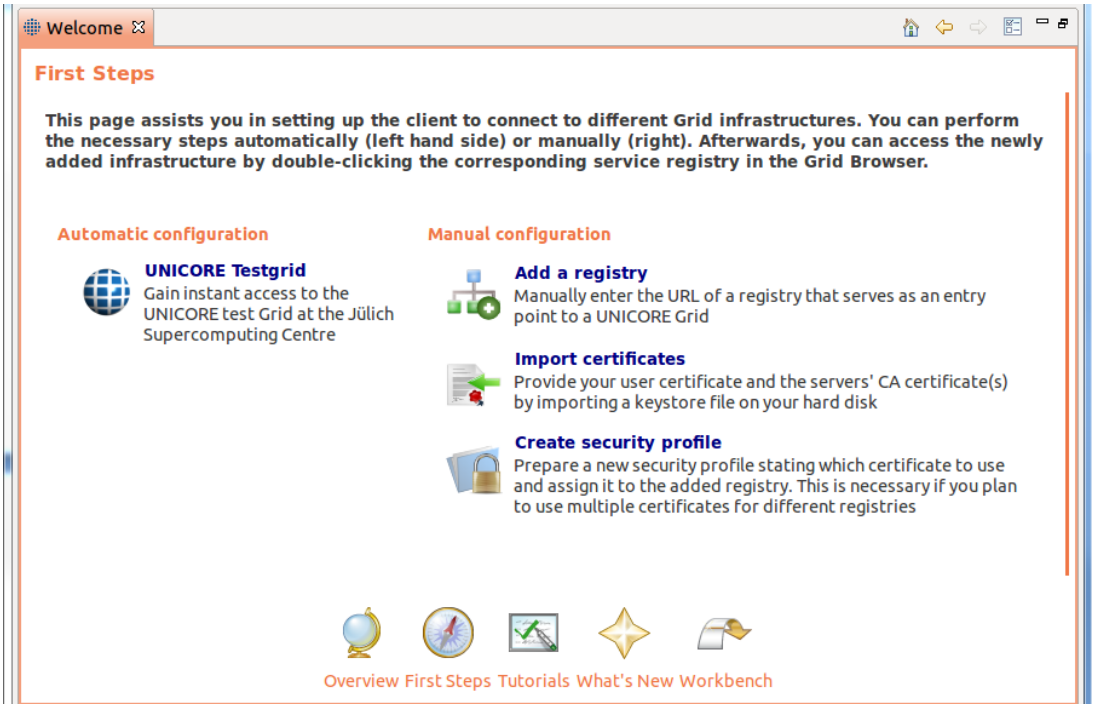
http://www.unicore.eu/download/unicore6/

The URC is available as a zip bundle for various operating systems. Extract the zip to a folder of your choice.

To run the client call "UNICORE_Rich_Client.exe" on Windows or "UNICORE_Rich_Client" on Linux. After this an authentication dialog appears which asks for you for a Keystore file and a password. For the moment we will use the default keystore file and password which comes with the URC so leave Keystore file as demouser.jks and for password "123".

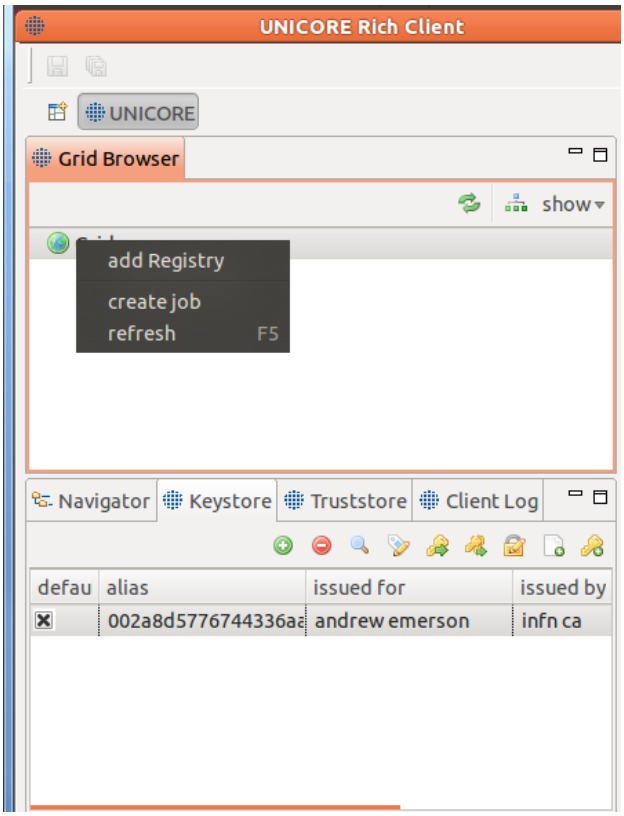
### *Setting the keystore*

You will be presented with a flash screen from which you should select the icon "First steps" where you will be taken to the following screen. The aim now is to create your own keystore by importing into the client your own certificate (the .p12 file) and the servers' CA certificates (the .jks file). You do this by selecting "Import certificates" from options under "Manual configuration". First step make sure that filetype .jks is selected and import the .jks : for password use "storepass". Then select filetype .p12 and import your certificate using the password you used when backing up the certificate from the browser. You can now enter the workbench by clicking on the Workbench icon. At this stage the first thing to do is to save your keystore file for future use. This you can do by clicking on the second key icon "Export public and private keys to keystorefile", "Select All" and then choose a file to save your keystore. You will also be asked for a password. The next time you startup the URC you should then use this file instead of the demouser.jks.




### *Selecting the Registry*

The next step is to select the Registry to use for running your UNICORE workflows. This can be done by right clicking on the Grid Symbol in the Grid Browser:

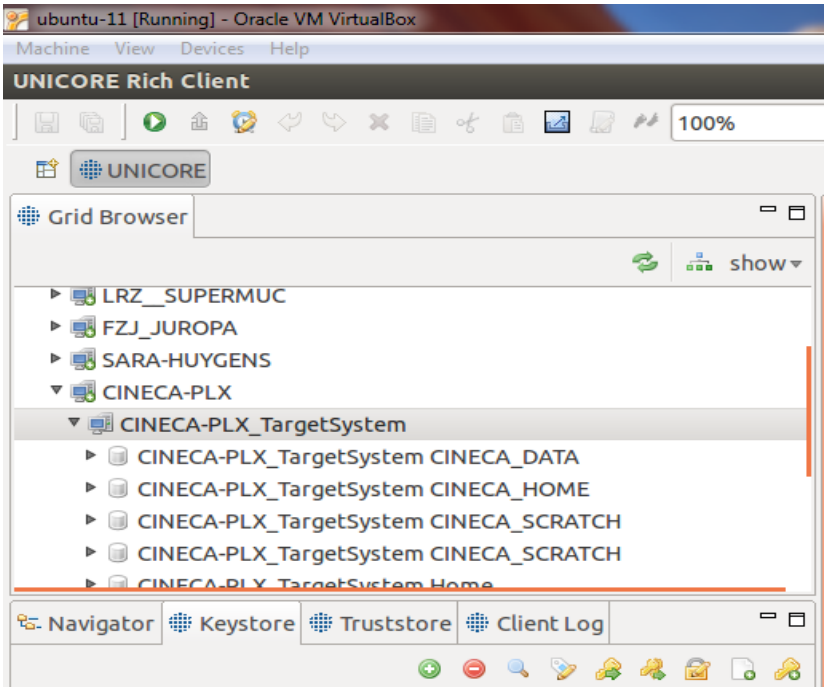



In this project a number of registries are possible. You can select one from the following table (see Table 1).

| **Site** | **UNICORE Registry** |
|---|---|
| CINECA | https://grid.cineca.it:9111/PRACE/services/Registry?res=default_registry |
| CSC | https://unicore.csc.fi:8081/MMM@HPC/services/Registry?res=default_registry |
| KIST | - |
| KIT | https://unicorex.scc.kit.edu:8080/KIT-UNICORE/services/Registry?res=default_registry |

**Table 1 UNICORE registries in the MMM@HPC project**

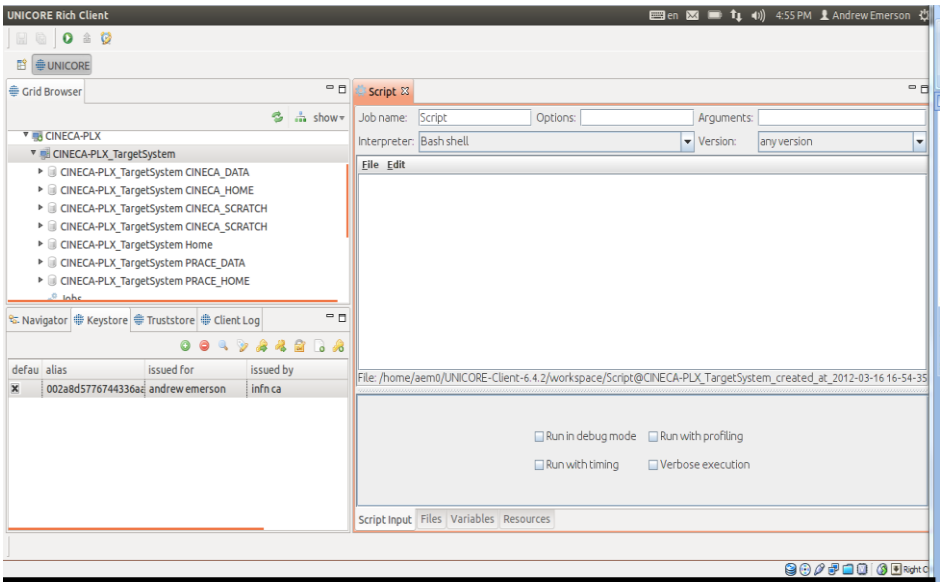
If you then click on the refresh symbol then a list of services will come up. If we assume you used the CINECA registry then you will see something like this is in the Grid Browser:



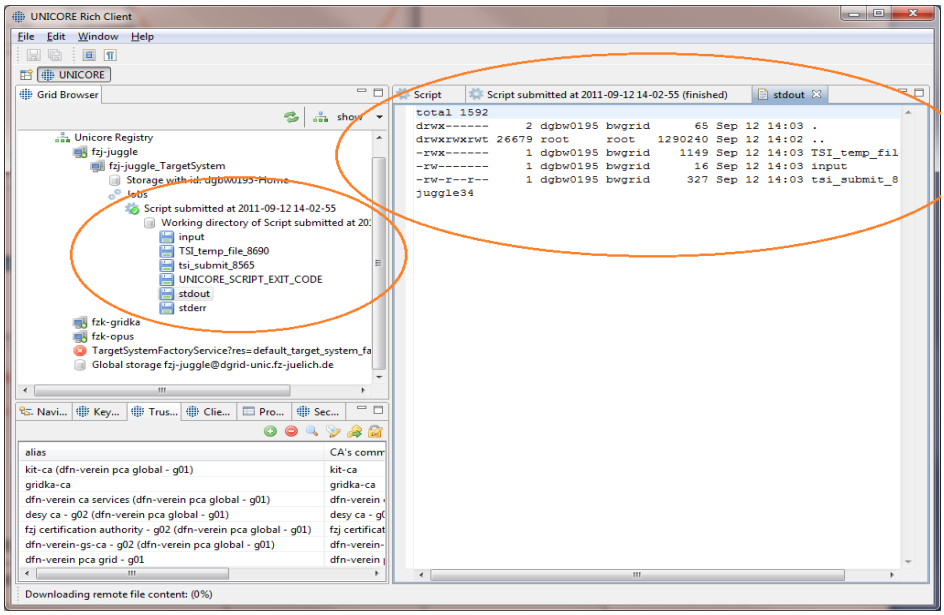

You are now ready to start launching jobs via UNICORE.

### *Running a simple job script*

To submit a job make a right click on one of the TargetSystems in the Grid Browser, select “create job” and select the “Script application”. Click on the Finish button. In the main window of the URC a form will be visible where you can enter a Bash script that will be executed on the selected UNICORE resource. Click the green play button in the menu bar for submitting a job to the resource.

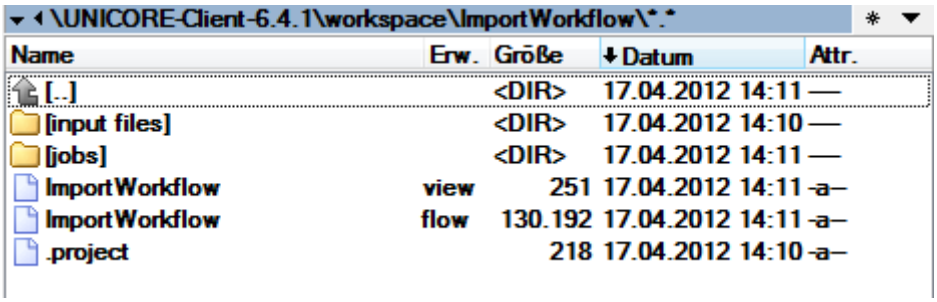


The current state of the job and the results can be monitored in the Grid Browser



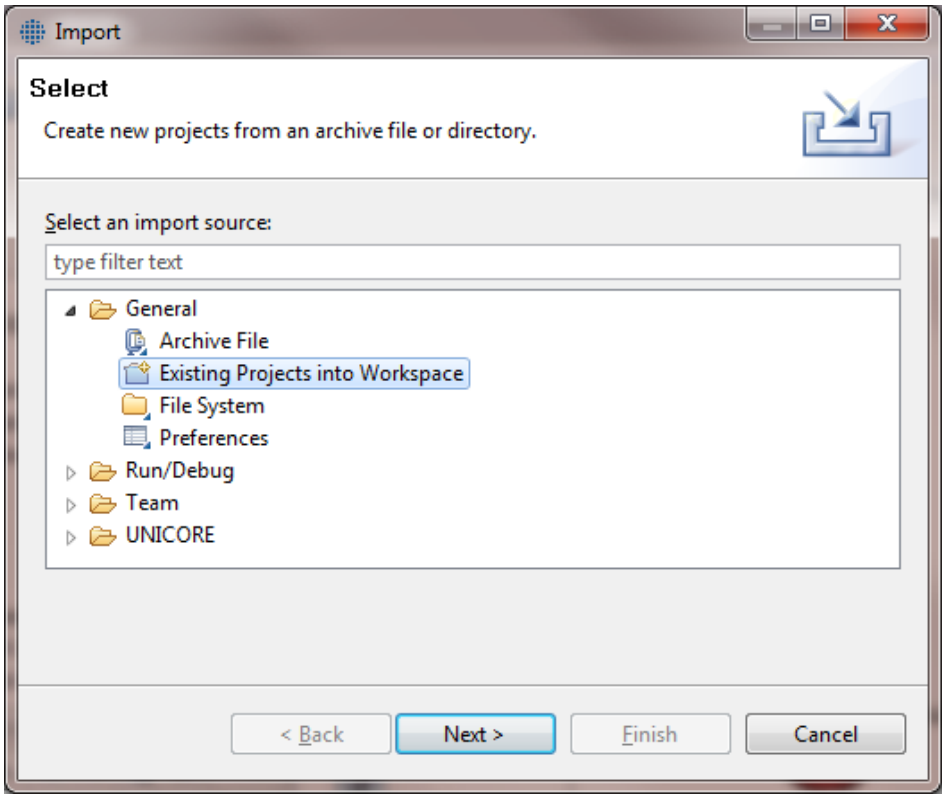

### *Loading a predefined workflow*

Once a workflow project has been created it can be shared with other people. Each workflow project is built of a couple of files and directories

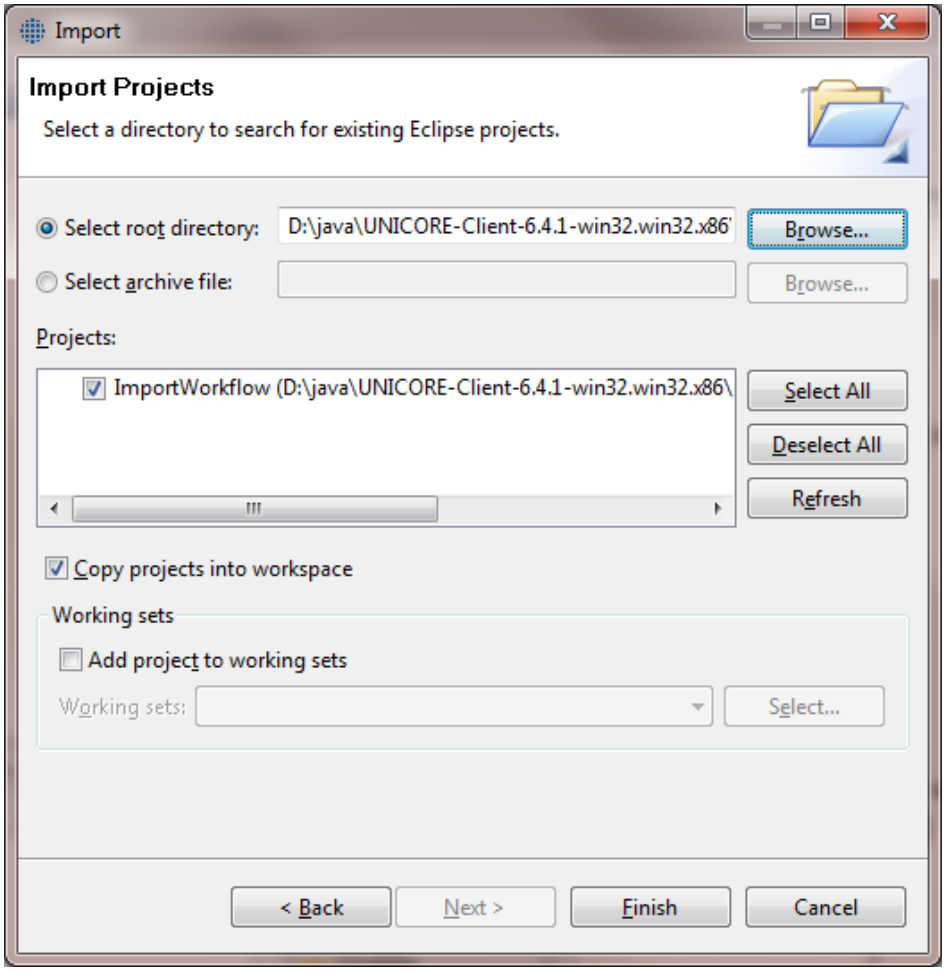



On the one hand you can export a workflow project and share its files with others respectively you can reuse workflows created by other by importing the project files to the URC.

The import of a project is quiet as easy as creating a new one. Go to the File menu and select the Import entry. An import dialog appears. Select as folder “General” and select the entry “Existing Projects into workspace”. Then click on the Next button.

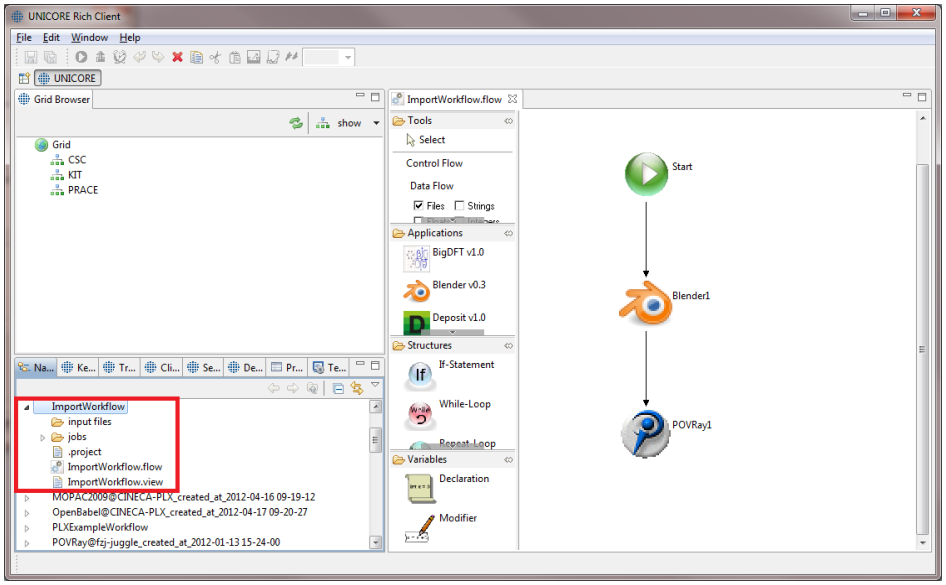



You move to the next import step. Select the root directory of the workflow project you want to import. Also select the "Copy projects into workspace" checkbox for a tight integration.



After clicking the Finish button the imported workflow project is available in the Navigator view.

A double click on ImportWorkflow.flow in the Navigator window will open the workflow project in the workflow editor.



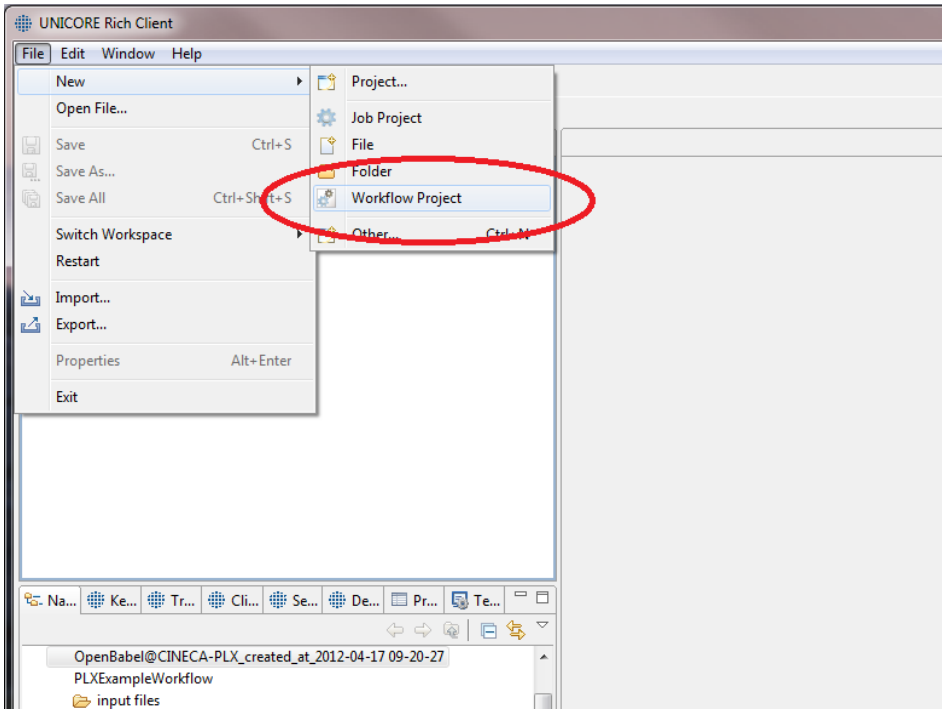

The workflow is now sent to the workflow engine. As mentioned before the status of the workflow and the individual jobs can be monitored in the Grid Browser window of the URC.

# 3 Training materials for materials modellers

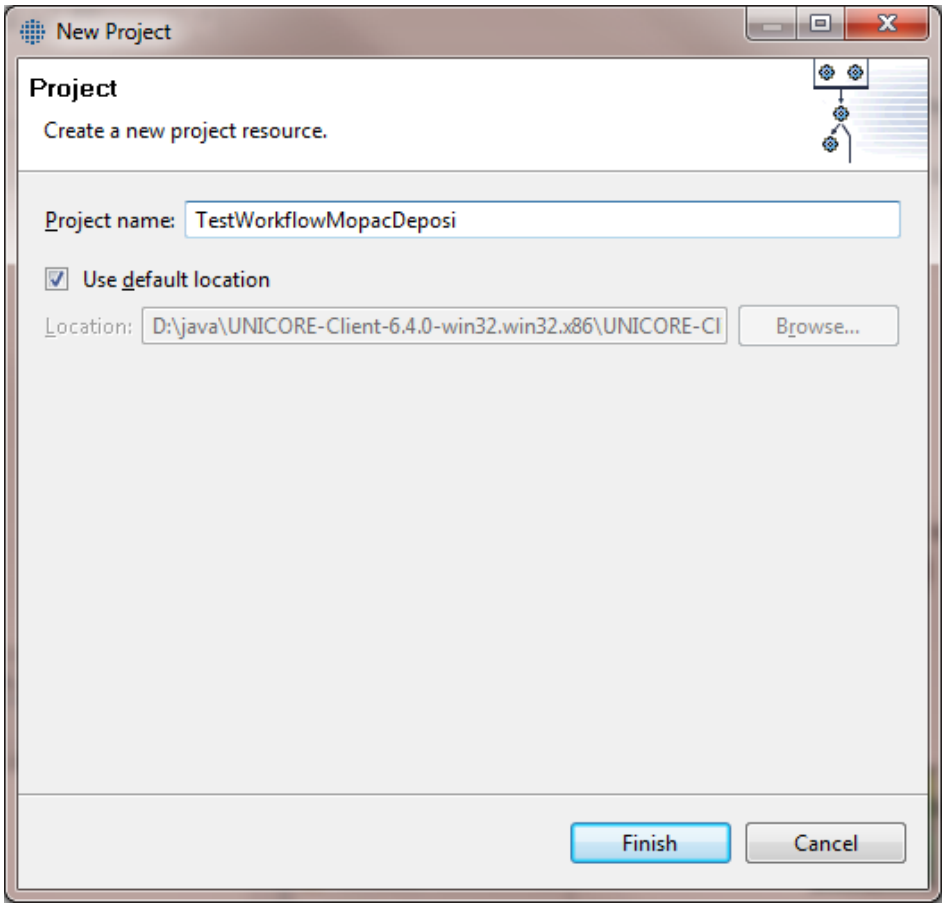
| |
|---|
| **Target group**: Molecular modellers |
| **Pre-requisities for students**: Basic knowledge of HPC and workflows. It is assumed that the students already possess X.509 certificates, have installed an URC client and can run simple scripts via UNICORE. |
| **Description of material**: Documentation on how to construct to workflows with Grid Beans for molecular modelling. |
| **Attached Supplementary files:** |

***Constructing a workflow***

Creating a new workflow project in the UNICORE Rich Client (URC) is fairly simple. In the main menu select: File -> New -> Workflow Project.




Select a name and a location for the workflow project and click on Finish.




After this the Client is changing to the workflow perspective and opens the workflow editor. The following is a short description of all workflow related elements and views in the URC (see Figure 1 Workflow elements in the URC):

Number 1 shows the extended button bar for workflows. The buttons trigger actions like:

- Submitting a workflow to a specific workflow engine
- Set a termination time for the workflow
- Export the given workflow to an xml file for further usage in other clients

The Grid Browser view showed in 2 makes it possible to monitor submitted workflows and to retrieve the results of individual workflow jobs. Workflow engines can be found as part of the UNICORE registries. A double click on a workflow engine will expand the tree under the node and shows all the workflows that have been submitted to this workflow engine. A double click on a specific workflow will retrieve further information about it and the performed jobs.

The workflow editor view, marked with the number 3, builds the centrepiece element for workflows. The editor is split into two main areas. On the left hand you find the 4 sections: **Tools**, **Applications**, **Structures** and **Variables** which offers the building blocks for professional workflows. Located on the right hand you find the editor area where you can use these building blocks by dragging and dropping them into it and concatenate applications to create a workflow.

Number 4 is the Navigator view which shows all workflow projects that are accessible by the client. Each project contains several files and directories. The flow and view files describe the workflow in detail. These files store information about the execution order of the applications. Furthermore this

directory contains the configuration of the individual applications as well as the associated input files and the resource requirements you specify for an application. All Variables and Files of submitted workflows are recorded and can be found in the project folder. So it's easy to resubmit a workflow with a specific configuration.

The MMM@HPC project relies on prior work from other projects, where large number of grid beans for packages widely used in the materials sciences were developed. In addition the partners of MMM@HPC are presently developing new grid beans for complementary techniques that are used in the application projects in this Consortium of which have been developed by the partners. The combination of these grid beans into workflows facilitates rapid development and testing of novel computational approaches for multiscale modeling in the materials sciences. In particular data conversion grid beans, such as the OpenMolGrid, that have been incorporated had users tackle complex data conversion issues when crossing from one scale to the next. MMM@HPC on the website maintains a depository for the workflows developed in this project, such that these can be downloaded by other interested scientists either for direct application of suitable modification. Access to grid beans is facilitated through the grid bean deployment service.

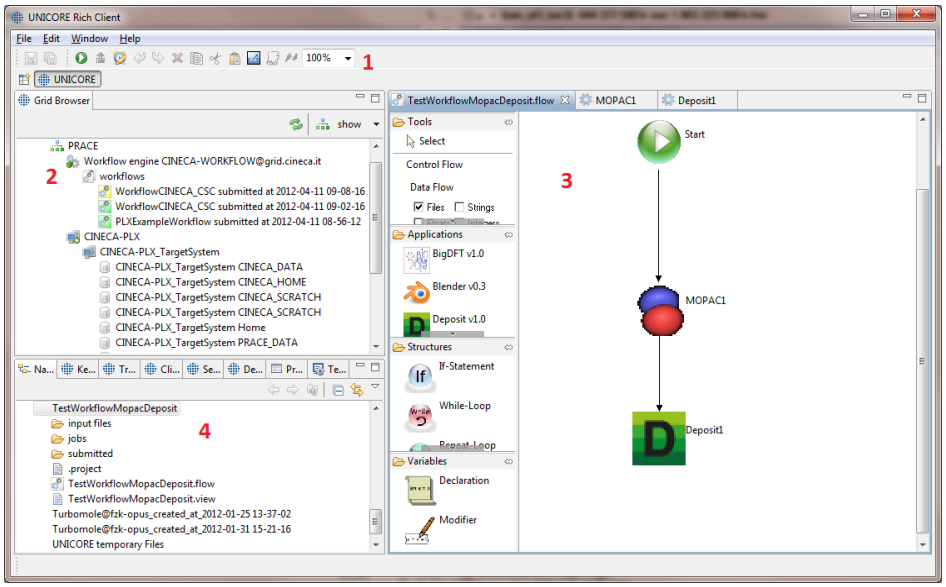
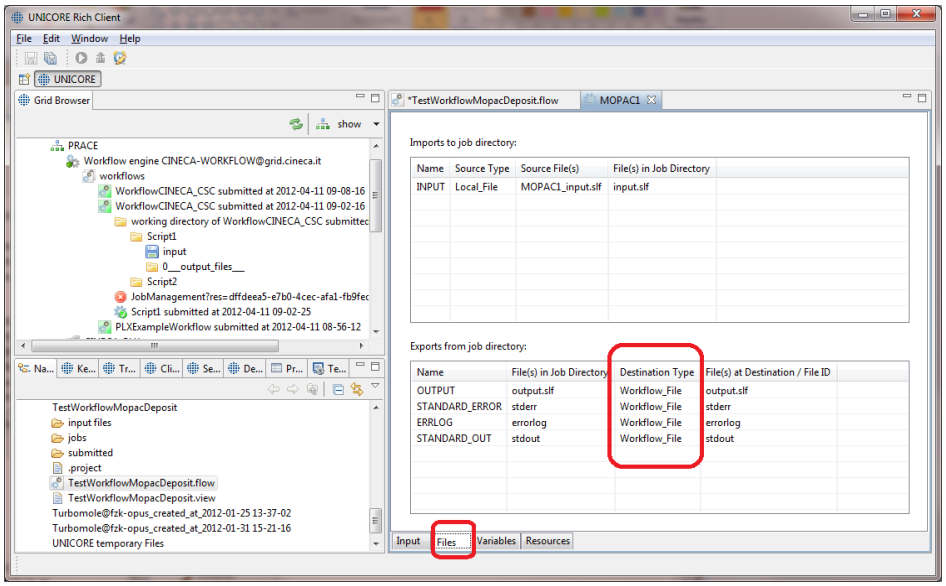


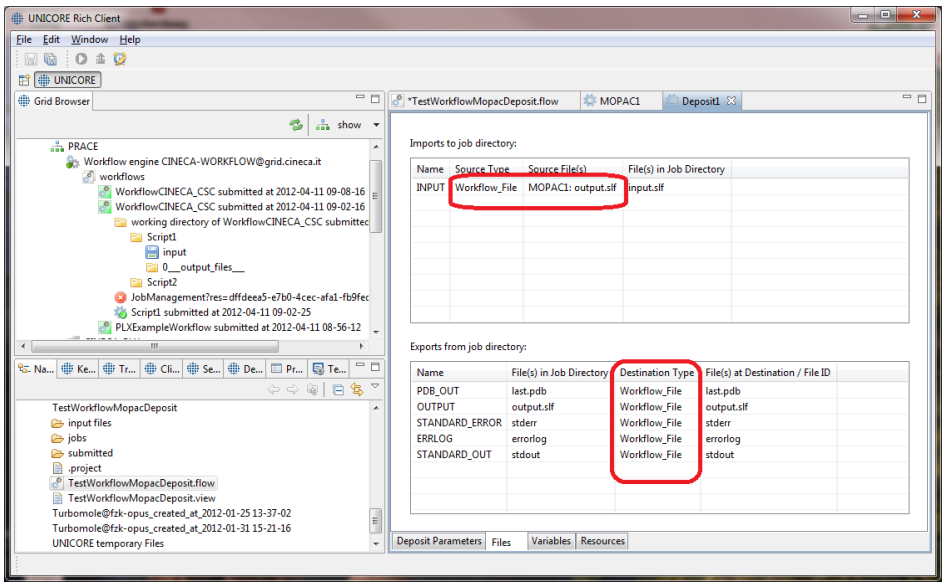

**Figure 1 Workflow elements in the URC**

The workflow editor is very handy in usage. For creating a workflow like shown in **Fehler! Verweisquelle konnte nicht gefunden werden.**, you have to select the corresponding applications from the Application section. Then dragging and dropping them into the editor section. Repeat this for all applications you need in the workflow. Connect the applications (This can be done easily by dragging one application symbol on another) to other applications or use Structure likes loops or conditions to create the application flow you needed.

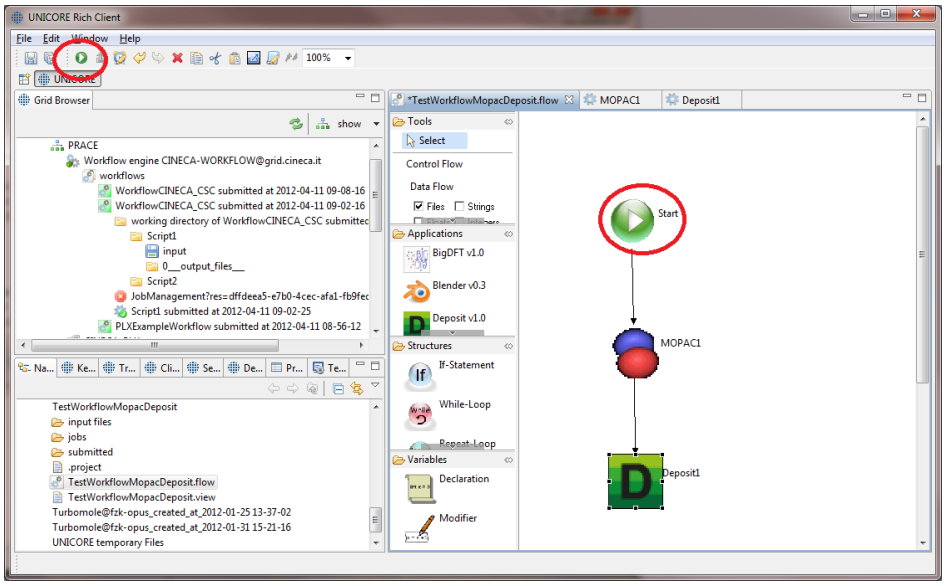
A double click on The MOPAC application will open the detail view. The most interesting point for workflows is the Files tab. This tab gives you the option to specify which Input/Output files of a job will be handled as a workflow file and therefore can be routed to other applications. The example specifies the destination Type of all MOPAC Output Files as "Workflow_File", so adjacent workflow applications can use them as input.



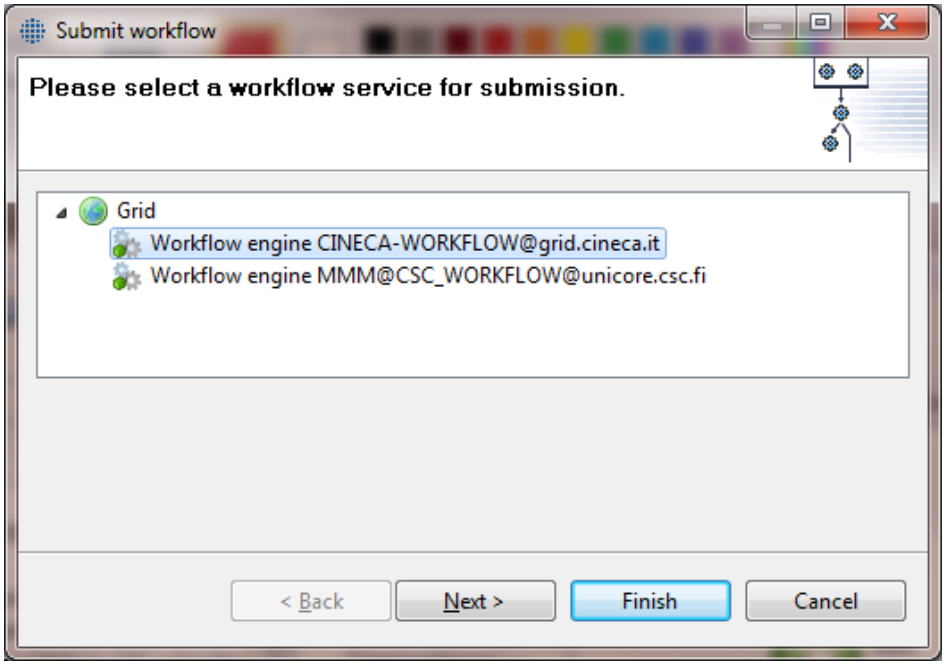

Go back to the workflow editor and make a double click on the Deposit application to open up the detail view. Open the Files tab and configure the input files. Select as Source Type “Workflow_File” for the file named INPUT. By clicking in the “Source Files” cell of this file you will get a list of all available workflow files. Select MOPAC1_output.slf which is the result of the previous workflow step.



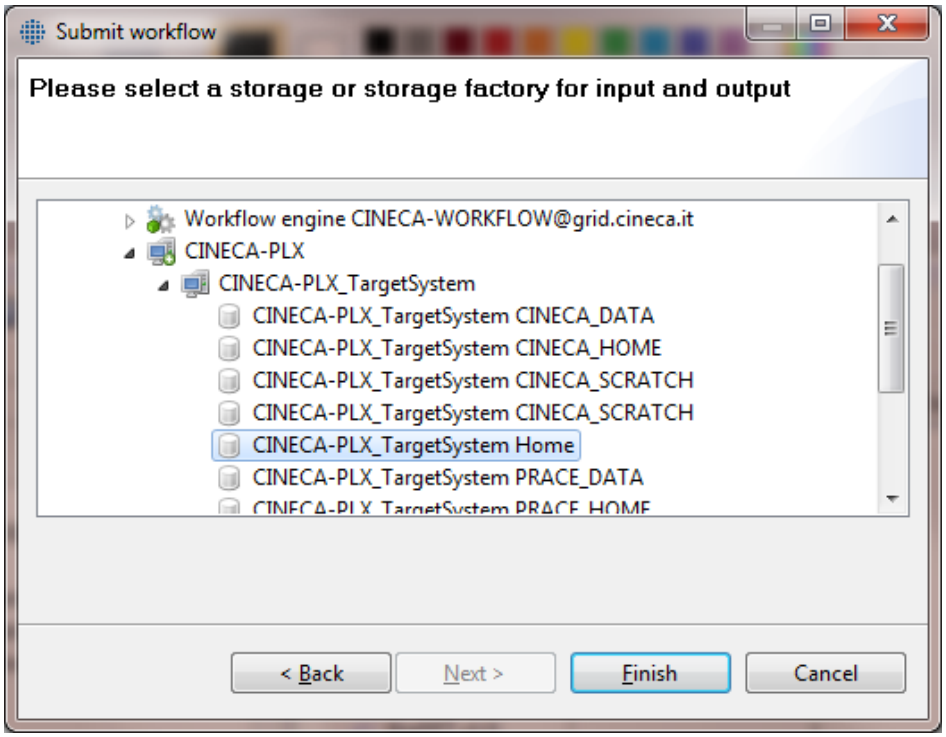

After configuring the workflow files you should continue in parameterising the applications MOPAC and Deposit in their detail views for a concrete simulation. Then the workflow is ready to submit to a workflow engine. You can trigger the submission of a workflow in two ways. Either click on the small play button in the button bar on the top of the client or make a double click on the big play symbol which marks the starting point in the workflow editor.




After pressing one of the buttons, a submission dialog will appear. Select a workflow engine and click on Next.




In the next step you have to select a storage or storage factory where you want to save the "Workflow_Files". These files are needed for the communication between the individual workflow steps. Finally submit the workflow by clicking on the Finish button.




Several documentations and tutorial videos included on the UNICORE homepage[1] shows in more detail how all the building blocks of the workflow editor works together. Additional material on workflows have been provided in deliverables D4.4 and D10.1. A collection of workflows will be provided in deliverable D8.3.

# 4 Training materials for GridBean Developers

| |
|---|
| **Target group**: Grid bean developers. |
| **Pre-requisities for students**: Experience with XML and JAVA would be useful. |
| **Description of material**: Developer manual for constructing Grid Beans. |
| **Attached Supplementary files:** GridbeanDevelopersGuide.pdf |

Note that the full documentation for this section is present in the attached file, GridbeanDevelopers-Guide.pdf.

## Introduction

A Grid one of the main concepts of the GPE(Grid Programming Environment) and is responsible for:

- generating a job description for grid services;
- providing graphical user interface for input data;

[1] http://www.unicore.eu/documentation/tutorials/unicore6/

• providing graphical user interface for output data;

• providing interface for user's interaction with grid services.

GridBeans are divided into several modules:

• one job description generation module;

• one or more user interface modules.

GridBeans are developed using the GridBean SDK and deployed as GridBean Services. GPE clients contact GridBean Services for available GridBeans and download the selected ones. Workflows may comprise many job descriptions created by different GridBeans.

## Developing GridBeans

***GPEJob***

One of the main data structures used in GridBeans is the abstraction of Jobs.

The `JSDLJob` is a Java interface for creating a JSDL document. As this document is generic the only things it can specify are:

• resource requirements;

• files to stage-in/stage-out;

• job name.

The `GPEJob` is a Java interface for a GPE atomic job description. It is an extension of JSDLJob.

The following additional attributes may be specified in one GPE Job:

• application name and version;

• named application parameters;

• application working directory;

• names of stdout and stderr files.

The application name and version uniquely select the application to be run on the target system. Named application parameters are substituted into the selected application.

***T ypical GridBean structure***

A typical GridBean consists of 2 modules:

• one job description generation module;

• one user interface module.

The job description generation module(GridBean Model) extends `AbstractGridBean` which inherits 2 interfaces and thus provides the following functions:

• storing named GridBean parameters (from the interface `IGridBeanModel`);

• generating job description using these stored parameters (from the interface `IGridBean`).

There may be several user interface modules. They inherit from `IGridBeanPlugin` and provide the following:

• the set of input panels (from the interface `IGridBeanPanel`);

• the set of output panels (from the interface `IGridBeanPanel`);

• methods for loading data from and storing data to the GridBean Model;

• validating input data.

***Create a GridBean Model***

Each GridBean has an underlying model object. You need this model object for:

• storing data

• and creating the actual job (usually in JSDL),that may be submitted to a target system or will be used within a workflow definition.

This object defines the logic of the GridBean and is independent of any graphical user interface that the GridBean provides. In order to implement the GridBean Model, create a new class that inherits from the class `com.intel.gpe.gridbeans.AbstractGridBean`.

The GridBean Model can be used similarly to a Hashmap. Use its setmethod for storing arbitrary data in it. Use the getmethod for retrieving stored data. The keys for storing and retrieving data should always be globally unique.

The client application requests the GridBean to generate a job description by calling

`IGridBean.setupJobDefinition(Job)`.

The typical code for generating a job description is as follows:

```
public void setupJobDefinition(Job job) throws GridBeanException {
super .setupJobDefinition(job);
if (job instanceof GPEJob) {
GPEJob gpeJob = (GPEJob) job;
gpeJob.setApplicationName("Date");
gpeJob.setApplicationV ersion("1.0");
gpeJob.setW orkingDirectory(
GPEConstants.JobManagement.TEMPORARY_DIR_NAME );
}
else { throw new GridBeanException("Invalid job type."); }
}
```

***Define all input and output parameters***

In the GridBean Model, all input and output parameters for the job to be created should be declared. This step refers to input and output files that the application takes/produces and environment variables that should be set during job execution. For declaring an input or output parameter , do the following:

• Choose and declare a qualified name(QName) for the parameter .

-Environment variable: the local part of the QName is used as the variable's name

-Input or output file:an environment variable with this name will also be created. The value of this environment variable will be assigned the name of the file in the job's working directory

• Define the parameter by creating an Object of type

`com.intel.gpe.gridbeans.IGridBeanParameter`. Have a look at

`com.intel.gpe.gridbeans.parameters.ParameterUtils` for convenient ways to do this. This object carries information like the parameter name, type(e.g. single file, fileset, environment variable)and whether it is an input or output parameter .

• Add the IGridBeanParameter object to either the input or output parameters of the GridBean-Model:

```
getInputParameters().add(param)
```

• Set the initial value of the parameter in the GridBean Model (using the chosen QName as a key and an Object of type
`com.intel.gpe.gridbeans.IGridBeanParameterValue` as value).

Consider the following example from the Povray GridBean:

```
QName[] envVariables = {WIDTH,HEIGHT ,ANTIALIASING_THRESHOLD, ANIMA-
TION, INITIAL_FRAME_NUMBER,FINAL_FRAME_NUMBER,SUBSET_START
,SUBSET_END,INITIAL_CLOCK_V ALUE,FINAL_CLOCK_V ALUE};

String[] initialValues =
{"320","200","0.5","false","0","0","0","0","0","0"};

getInputParameters().addAll(ParameterUtils.createEnvParameters(envV
ariables));

List<IGridBeanParameterV alue> values = Parameter-
Utils.createEnvParameterV alues(env Variables,

initialValues);

for (int i = 0; i< initialV alues.length; i++) {

set(envV ariables[i],values.get(i));

}
```

First, an array is created, containing all chosen QNames for the environment variables known to POVRay. Then an array of initial values for these variables is provided. The `ParameterUtils`

class is used for creating the environment variable parameters and the objects representing their initial values. The parameters are added to the GridBean's input parameters, their values are stored in the model. Input and output parameters belong to one of the following types(`see GridBeanParameterType`):

- File
- File Set
- Environment variable
- Resources

***Create GridBean user interface***

GridBeans may provide one or more graphical user interfaces, called "plugins", for user interaction. Plugins are responsible for:

- the communication between a user and the GridBean and the possibility to modify the GridBean Model.(e.g. a user can set the value of an environment variable that is used as a parameter for the job definition that the GridBean produces).
- the support for additional platforms and windowing toolkits. E.g. a GridBean may also provide a portal plugin defining a GUI which can be displayed inside a web portal.

A plugin provides:

- A set of input panels;
- A set of output panels.

The following steps need to be taken in order to write a Swing based GridBean plugin

- Create one or more panels that inherit from

  `com.intel.gpe.gridbeans.plugins.swing.panels.GridBeanPanel`.

- Create a new class that extends

  `com.intel.gpe.gridbeans.plugins.swing.GridBeanPlugin`.

- Override its parent's initialize method (still call `super.initialize()`.
- Within the initialize method, add all your panels with the addIn/OutputPanelmethods.

Each GridBean panel will have a couple of controls, usually created in its `buildComponents()` method. For linking a control to a parameter ,a value translator and a value validator should be specified.

A translator is an object implementing `IValueTranslator` that translates the original value contained in the graphical component into the GridBean's internal value representation. E.g. the value of the text field component that is used for specification of some numerical data is of type String but it may be translated and stored in theGridBean model as

`EnvironmentVariableParameterValue`.

A validator is an object implementing `IValueValidator` that validates the translated value of the graphical component and provides the error message in the case of invalid value. E.g. the validator may check that an inputvalue is not empty .

The typical code for creating input panel components looks like this:

```
privatevoid buildComponents()throws DataSetException{

setLayout(new GridBagLayout());

JT extField nameT extField = new JT extField();

add(new JLabel("Name:"), LayoutT ools.makegbc(0, 0, 1, 0, false));

GridBagConstraints c = LayoutT ools.makegbc(1,0, 1, 0, true);

add(nameTextField,c);

linkJobNameT extField(POVRayGridBean.JOBNAME,nameT extField);

JT extField widthField = new JT extField();

add(new JLabel("W idth:"),LayoutT ools.makegbc(1,1, 1, 0, false));

add(widthField,LayoutT ools.makegbc(2, 1, 1, 0, true));

linkT extField(POVRayGridBean.WIDTH, widthField);

setV alueTranslator(POVRayGridBean.WIDTH, StringV alueTranslator
.getInstance());

setV alueV alidator(POVRayGridBean.WIDTH, IntegerV alueV alidator
.getInstance());

JT extField heightField=new JT extField();

add(new JLabel("Height:"), LayoutT ools.makegbc(3, 1, 1, 0, false));

add(heightField, LayoutT ools.makegbc(4, 1, 1, 0, true));

linkT extField(POVRayGridBean.HEIGHT,heightField);

setV alueTranslator(POVRayGridBean.HEIGHT , StringV alueTranslator
.getInstance());

setV alueV alidator(POVRayGridBean.HEIGHT , IntegerV alueV alidator
.getInstance());

}
```

In this example, the text fields are linked to the parameter with a special QName (e.g. POVRayGrid-Bean.WIDTH, this is an environment variable parameter for telling POVRay the width of the image to be rendered).

***Define a configuration file gridbean.xml***

Each GridBean has a configuration file gridbean.xml. In this file you should define the name of your grid bean, your name, version, the name of the application, a description for your GridBean and the paths to your developed GridBean Model and your GridBean Plugins. You also should declare an

application name and an application version. These names should be the same as the application name and version which you definein your GridBeanModel.

In the following example you will see an example for the configuration file.

```
<gb:GridBeansInfo xmlns:gb="http://gpe.intel.com/gridbeans">

<gb:Name>Script</gb:Name>

<gb:Author>V alentinaHuber</gb:Author>

<gb:V ersion>2.0</gb:V ersion>

<gb:Application>Script</gb:Application>

<gb:Description>ScriptGridBean</gb:Description>

<gb:GridBean>de.fzj.gpe.gridbeans.script.ScriptGridBean</gb:GridBean
>

<gb:Plugin
type="gb:Swing">de.fzj.gpe.gridbeans.script.plugin.ScriptPlugin</gb:
Plugin>

<gb:ApplicationName>Perl</gb:ApplicationName>

<gb:ApplicationV ersion>1.0</gb:ApplicationV ersion>

</gb:GridBeansInfo>
```

***The GridBean Model may have multiple views***

When writing a GridBean you should always bear in mind that your GUI plugin is only one view on the GridBean Model. There might be additional GUI elements in a client that alter the model. For instance, the Eclipse based client provides generic panels for defining file imports and exports for jobs. These panels will operate on your GridBeanModel and change the values of file input and output parameters! In order to keep up with the current state of theGridBean Model, it is necessary to use its built-in property change support. After adding a property change listener to the model, you will be notified of any changes that occur .

# 5 Supplementary and general material

## Interactive access to HPC resources

### *Using CINECA's PLX GPU Cluster*

#### *Description*

PLX is a linux cluster based on nodes with Intel Westmere processors and nVIDIA Tesla M2070 GPUs linked by an Infiniband network: for full specifications see this link http://hpc.cineca.it. The important feature of PLX is that it is currently the largest GPU cluster in Europe and if able to use fully the GPUs, is capable of more than 500 Tflops performance (without the GPUs it reaches only about 32 Tflops). In this section we explain how to use and run jobs on PLX via the Unix operating

system, rather than through UNICORE. A basic knowledge of UNIX, and in particular the Linux, dialect is assumed. Please note also the full documentation for PLX which can be found at CINECA's HPC portal http://hpc.cineca.it.

***Obtaining an account and logging into the system***

To use PLX one needs an "account" which can be obtained via the following routes:

- Via ISCRA, the Italian national call for computer time
- A PRACE DECI Tier-1 allocation
- Through a project such as MMM@HPC which has been an allocation of time on PLX

With the exception of PRACE DECI projects, each user will be asked to register as described in section XX. If the account request has been successful then a username and password will be sent to the email address specified during registration. Note that although a personal x.509 certificate is not required to run on PLX, the supplied username and password are strictly personal and must not be shared with other users.

To login to the system one uses the SSH program which is present by default on all Unix systems while for MS Windows systems there are various free and commercial packages available (e.g. the free PUTTY client):

```
ssh username@login.plx.cineca.it
```

For those with certificates and have registered their DNs access via globus/gsissh is also possible:

```
grid-proxy-init
gsissh gssh.plx.cineca.it -p 2222
```

For more information on using globus and certificates we suggest users consult the documentation available from PRACE (http://www.prace-ri.eu/PRACE-User-Documenation).

***Disks and directories***

In we show the current disk and filesystem layout of the two major clusters of CINECA. In particular, users of PLX should note that they have access to three types of filesystem, identified by the environment variables and with the features as described below:

**$HOME**

Permanent, backed-up,and local to SP6. Quota = 2GB. For source code or important input files.

**$CINECA_DATA**

Permanent, no backup, and shared with other CINECA systems. Mounted only on login nodes (i.e. not visible in normal batch jobs). Quota=100Gb can be extended on request. Intended as a temporary backup area and file transfer between PLX-SP6.

**$CINECA_SCRATCH**

Large, parallel filesystem (GPFS). Temporary (files older than 30 days automatically deleted), no backup. No quota. Run your simulations and calculations here.

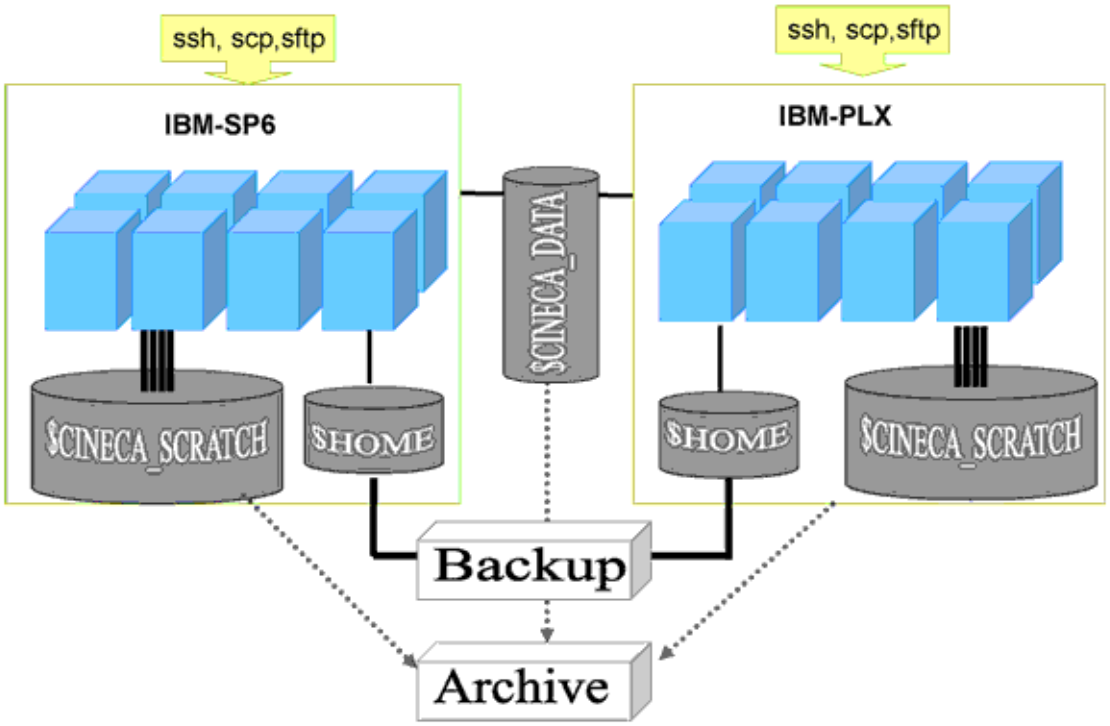



**Figure 2 Disks and Filesystems at CINECA**

The most important point here is that users should run their jobs in the directory specified by the variable $CINECA_SCRATCH which not only provides the most physical storage, but is also designed for running jobs in parallel.

***PLX module environment***

PLX uses the *module* system to provide access to applications, compilers, libraries and so on. Modules typically set shell variables such as **$PATH** (for commands) or **$LD_LIBRARY_PATH** (for libs). Avoids "cluttering up" the default user shell settings which the user may never use. *Unloading* the module will return the environment to the state existing before the module was loaded.

Example module commands are as follows:

```
module load gromacs   # load default version of gromacs

module load namd/2.8  #  load version 2.8 of NAMD

module unload lammps  # remove lammps module from environment

module show cmake     #  show  what  the  cmake  module  does  (but  dont
load it)

module list           # list modules loaded by user

module avail          # list modules available

module purge          # remove all modules from environment

module load profile/advanced # load advanced module profile
```

```
module help fftw# get help on the module
```

Note that some modules have dependencies, i.e. require other modules before they can be loaded. For example:

```
module load amber/11

WARNING: amber/11 cannot be loaded due to missing prereq.

HINT: the following modules must be loaded first: IntelMPI/4.0-binary
```

To ensure that the required modules are loaded first it is usually possible to use the special autoload module,

```
module load autoload amber/11
```

***Compilation of programs***

On PLX three makes of compiler are provided:

- gnu
- pgi
- intel

and two implementations of the MPI library

- openmpi
- intelMPI

With the exception of GNU version 4.1.2 you need to load the appropriate compiler module:

```
module load intel
```

You can then compile your program:

```
ifort –o myprog.x myprog.f
```

Often multiple compiler versions are maintained to allow backwards compatibility:

```
module av pgi

pgi/11.1(default) pgi/11.6          pgi/11.7
```

In order to compile with libraries available as modules you would normally do something like this (e.g. for FFTW):

```
module load intel fftw

module list

ifort –I$FFTW_INC –o myprog myprog.f –L$FFTW_LIB  -lfftw3f
```

The fftw module defines the **FFTW_INC, FFTW_LIB** variables which can then be used in the compilation.

MPI programs are usually compiled with the "wrapped" versions of the compilers:

```
module load gnu openmpi/1.3.3--gnu--4.1.2

mpif90 -o myexec myprof.f90 (uses the gfortran compiler)
```

***Batch submission with PBS***

The login nodes are meant for simple operations such as editing or short (non-CUDA) compilations. Anything else must be done on the compute nodes via the PBS batch system. In particular, **do not run parallel MPI programs on the login nodes**.

UNIX commands requiring >10 min CPU also need to be done on the compute nodes.

The batch system on PLX is called PBS Pro. You can access the compute nodes via a PBS interactive session as follows:

```
qsub -A <account_no> -I -l select=1:ncpus=2:mpiprocs=2 -q debug
```

(ssh access to the compute nodes is disabled for non-staff users)

Once PBS finds a compute node with the requested resources it will log you on the node.

Some common submission options:

```
qsub
-l <resources>  # resource request
-I              # interactive mode
-q <queue>      #  queue name
-A <account number>    # necessary for some projects (not PRACE)
```

An important option is the resource request option **–l**

For the wall time limit

```
-l walltime=hh:mm:ss
```

while to reserve nodes, cpus, mpi processes etc

```
-l select=1:ncpus=12:mpiprocs=12
```

This reserves 1 "chunk" with 12 cores where we can run 12 MPI processes.

It is often more convenient to write a batch script with PBS directives:

```
#PBS -A MMM_myproj
#PBS -N job_name
#PBS -l walltime=1:00:00
#PBS -l select=16:mpiprocs=8:ncpus=8
#PBS -o job.out
#PBS -q parallel
```

```
cd $PBS_O_WORKDIR          # cd to submission directory
module load autoload namd  # load namd module
mpirun -np 128 namd2 md.namd    # use mpirun to run MPI prog namd
```

If these lines are in a file **job.pbs**, then to submit a job just do the following command

```
qsub job.pbs
```

Once you issue a qsub command you will be given an identifier, the *jobid*, which can be used to refer the job (see later).

Notice that:

- we have asked for 16 chunks, each with 8 cores and 8 MPI processes
- **mpirun** to run an MPI program
- the option **–o** to rename the standard output

*Other PBS examples**:***

Variable chunk sizes

```
#PBS -l select=2:ncpus=8:mpiprocs=8+1:ncpus=5:mpiprocs=5
```

2 chunks of 8 cpus + 1 chunk 5 of cores.

Hybrid MPI/openmp

```
#PBS -l select=2:ncpus=8:mpiprocs=1
```

2 chunks: for each chunk 1 MPI/task + 8 threads.

Job dependencies (“chaining”)

```
qsub -W depend=afterok:JOBID.node351.plx.cineca.it job.sh
```

Wait for job JOBID to finish before starting.

Finally, in order to check the status of your job you can use the qstat command

```
qstat –u $USER
```

while to delete a running or queued job use qdel together with jobid

```
qdel 134658.plx.cineca.it
```

***Using GPUs on PLX***

To use the full potential of PLX applications need to exploit the two GPUs present on each node. At present most GPU-accelerated applications use NVIDIA’s CUDA library for accessing the GPUs and currently this is the only supported method on PLX. As of the time of writing few applications have been CUDA-enabled but this number is expected to grow in the future. The current module configuration of PLX lists provides CUDA-enabled versions of these programs:

- ACEMD (molecular dynamics)
- Amber 11 (molecular dynamics)
- Gromacs 4.5 (in serial only)
- NAMD 2.8 (molecular dynamics)
- Quantum Espresso (selected modules only)

Not all features of these programs can be run on GPUs so users should check first with the relevant documentation.

It is important that when GPUs are required for an application they are explicitly requested from PBS as in the following example:

```
#PBS -N namd_cuda
#PBS -l walltime=20:00
#PBS -l select=1:ncpus=12:mpiprocs=12:ngpus=2
#PBS -A cinstaff
#PBS -q parallel
module load autoload namd/2.8/cuda
module load cuda
# the following line is needed due to bug with CUDA 4.0 and Infini-
band
export CUDA_NIC_INTEROP=1
cd $PBS_O_WORKDIR
cmd="namd2 +idlepoll dhfr.namd"
mpirun -np 2 $cmd
```

Notice also that in this example it has been necessary to load the cuda module present on PLX. This module can also be used to compile your own CUDA programs but you should remember to do the compilation on the compute nodes of PLX since the various drivers are not present on the login nodes of the system. For example,

```
qsub -A <account_no> -I -l select=1:ncpus=1:ngpus=2 -q debug
module load gnu
module load cuda
nvcc -arch=sm_20 -I$CUDA_INC -L$CUDA_LIB -lcublas -o myprog myprog.c
```

***Support and Documentation***

With the exception of PRACE users all questions and requests for support should be directed towards the CINECA helpdesk via the email address:

superc@cineca.it

PRACE users should use instead the PRACE help desk:

support@prace-ri.eu

Both helpdesks use a trouble ticket system to manage support requests.

Documentation is available from CINECA's HPC portal:

http://hpc.cineca.it

PRACE-specific documentation is available from the PRACE site: http://www.prace-ri.eu

## Using resources at the CSC computer centre

***Description***

Louhi is a Cray XT4/XT5 Massively Parallel Processor (MPP) supercomputer and it has a theoretical peak performance of 102.2 Tflops/s, not counting the service nodes handling logins, I/O, etc., and the estimated Linpack performance of about 76.5 Tflop/s.

The facility contains 4048 compute cores in 1012 compute nodes, which are located in 11 cabinets. Each XT4 compute node contains one quad-core processors and thus 4 cores. The Cray XT5 part contains 6816 compute cores in 852 compute nodes, which are located in 9 cabinets. Each XT5 compute node contains two quad-core processors and thus 8 cores. Each processor and its memory form a NUMA (NonUniform Memory Access) node. Thus the XT5 compute node contains two NUMA nodes.

The processors of XT5 and XT4 are quad-core 2.3-GHz AMD Opteron 64-bit (Barcelona or AMD Family 10h) processors except in two XT5 cabinets (180 compute nodes, 1440 cores) belonging to the PRACE (Partnership for Advanced Computing in Europe) which are quad-core 2.7-GHz AMD Opteron 64-bit (Shanghai) processors. There is dedicated memory of 1 GB or 2 GB per core. The architecture of the quad-core Opteron processors is well suited for the floating-point computation and memory traffic requirements common in high performance computing, making good sustained performance possible. The performance is enhanced by using the Compute Node Linux operating system in the compute nodes which strips the overhead of the operating system to the minimum.

***Disks***

Louhi has several local disks for system files and computational data. The most important one for Louhi users is the work disk, under /wrk directory, where all users have a personal directory specified by the $WRKDIR variable. All computational work should be performed in $WRKDIR for performance reasons and because compute nodes see only Lustre file systems. The file system in /wrk is Lustre, a high-performance parallel file system designed to handle the huge I/O demands of parallel programs. You can found more information about disk system from http://www.csc.fi/english/pages/louhi_guide/hardware/disks/index_html.

***Logging in***

The outside world knows Louhi as louhi.csc.fi. The short alias name louhi can be used inside the domain csc.fi. When logging into Louhi, you will actually get connected to one of the six login nodes. These nodes provide the usual system services needed for program development and the preparation of the production runs, e.g., the compilers. You will get connected to login node which has the lowest load at the login moment. The logging process has its own section in the Louhi User's Guide: http://www.csc.fi/english/pages/louhi_guide/using_louhi/logging_in/index_html.

***Storing and moving files, program development and batch jobs***

Information about storing and moving files, program development and batch jobs are available from: http://www.csc.fi/english/pages/louhi_guide/using_louhi/storing_files,

http://www.csc.fi/english/pages/louhi_guide/program_development and http://www.csc.fi/english/pages/louhi_guide/batch_jobs, respectively.

***Support and documentation***

The full Louhi User's Guide can be read at: http://www.csc.fi/english/english/pages/louhi_guide. CSC's Service Desk is open between 8.30-16.00 (EET) hours from Monday to Friday at help-desk@csc.fi.

## User Feedback Surveys

In this section we give the text of a sample user feedback survey which will be given to students during training courses. The content was partially inspired by the user feedback forms used at CINECA for HPC courses, but the format and most of the content is original to MMM@HPC. However, we have kept the principle of keeping surveys *short*; in our experience students will not fill in long feedback forms. This survey has also been tailored for training courses involving the use of molecular modelling software on HPC platforms but there will be customised versions also for each training session such as those involving UNICORE and Workflows, those dedicated to molecular modelling and so on. The feedback form itself has been created with the SurveyMonkey software (http://www.surveymonkey.com), and of course is expected to be completed on-line. The form below can be found at the URL: xxxx but when training courses start, a URL based on a project portal will be used. To see how this form would look in a web page, see **Figure 3**.

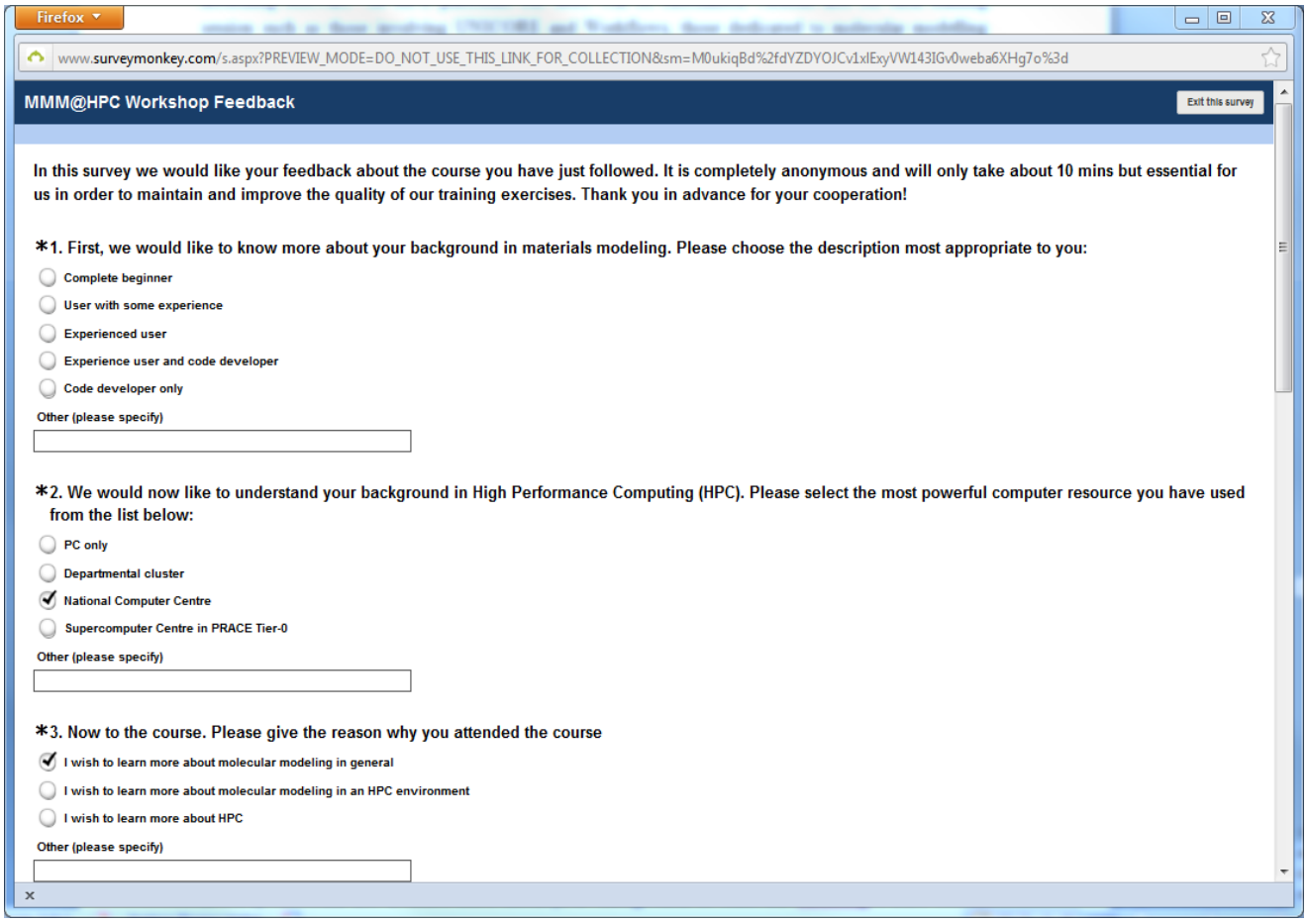



**Figure 3 Snapshot of the web page showing the feedback form.**

**MMM@HPC Workshop Feedback**

In this survey we would like your feedback about the course you have just followed. It is completely anonymous and will only take about 10 mins but essential for us in order to maintain and improve the quality of our training exercises. Thank you in advance for your cooperation!

**1. First, we would like to know more about your background in materials modeling. Please choose the description most appropriate to you:**

- Complete beginner
- User with some experience
- Experienced user
- Experience user and code developer
- Code developer only

Other (please specify)

**2. We would now like to understand your background in High Performance Computing (HPC). Please select the most powerful computer resource you have used from the list below:**

- PC only
- Departmental cluster
- National Computer Centre
- Supercomputer Centre in PRACE Tier-0

Other (please specify)

**3. Now to the course. Please give the reason why you attended the course**

- I wish to learn more about molecular modeling in general
- I wish to learn more about molecular modeling in an HPC environment

○ I wish to learn more about HPC

Other (please specify)

***4*. How do you rate the quality of the lessons overall? Please select one from the following**

○ Very poor

○ Poor

○ Fair

○ Good

○ Very good

**5. Please now rate each of the trainers in the course. Remember your responses will be completely anonymous.**

| | **Very poor** | **Poor** | **Fair** | **Good** | **Very Good** |
|---|---|---|---|---|---|
| **Trainer 1** | ○ Very poor | ○ Poor | ○ Fair | ○ Good | ○ Very Good |
| **Trainer 2** | ○ Very poor | ○ Poor | ○ Fair | ○ Good | ○ Very Good |
| **Trainer 3** | ○ Very poor | ○ Poor | ○ Fair | ○ Good | ○ Very Good |

**6. How would you rate the quality of the teaching materials?**

○ Very poor

◉ Poor

○ Fair

○ Good

- Very Good

**7. How would you rate the suitability of the facilities (computer room, PCs etc)?**

- Very Poor
- Poor
- Fair
- Good
- Very Good

## Managing the UNICORE Keystore for Advanced Users

To access UNICORE resources it's mandatory to setup a Java Keystore which includes the users grid certificate and some additional CA certificates (which the users trusts and that signed the resource certificates). A Java Keystore is, as the name implies, a container for cryptographic keys such as certificates.

If you have exported your grid certificate from Firefox you can create a Java Keystore with a graphical user interface like the KeyStore Explorer http://www.lazgosoftware.com/kse/index.html The tool is available for Windows, Mac and Linux and easy to setup: a snapshot of the start-up screen is shown in Figure 4.

How to create a new Java KeyStore?

- Click on "Create a new KeyStore" and select JKS as type of the new keystore.

After this you can add keys and certificates in different formats to the keystore. A common use case is to add your grid credentials to the keystore which you have exported from Firefox as PKCS12 format. For this go to the menu "Tools -> Import Keypair" select PKCS #12 as type of key pair to import and specify the p12 file on the file system and the decryption password for the file.

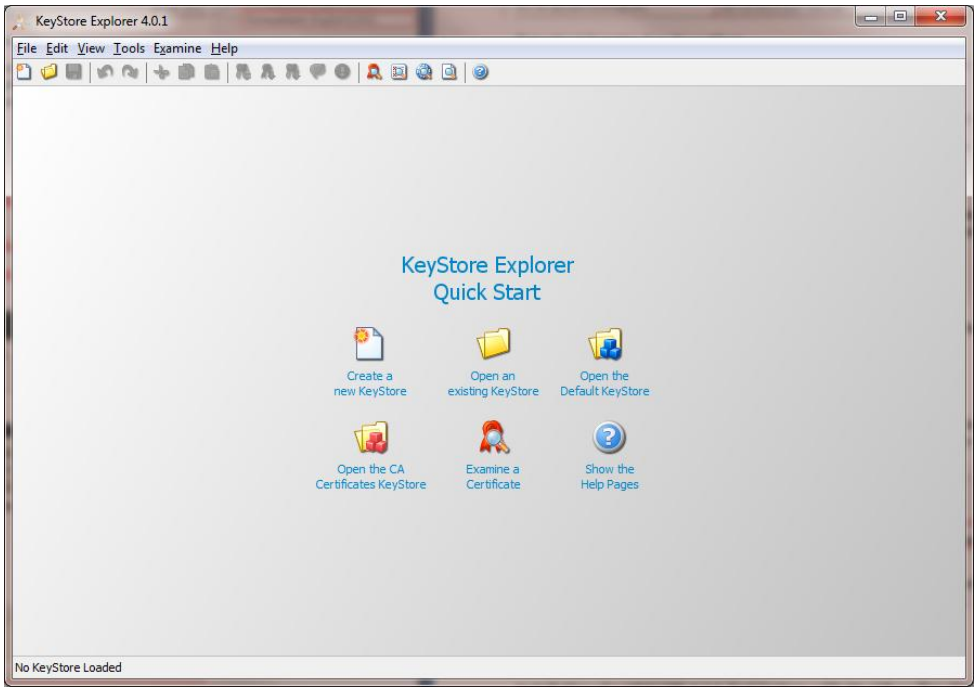



**Figure 4 The KeyStore Explorer**

As well as your user certificate you have to import the CA certificates that you trust. These are needed for a secure communication with the UNICORE resources that you will access. The following screenshot Figure 5) shows a Java KeyStore which includes the gird user certificate and several CA certificates:

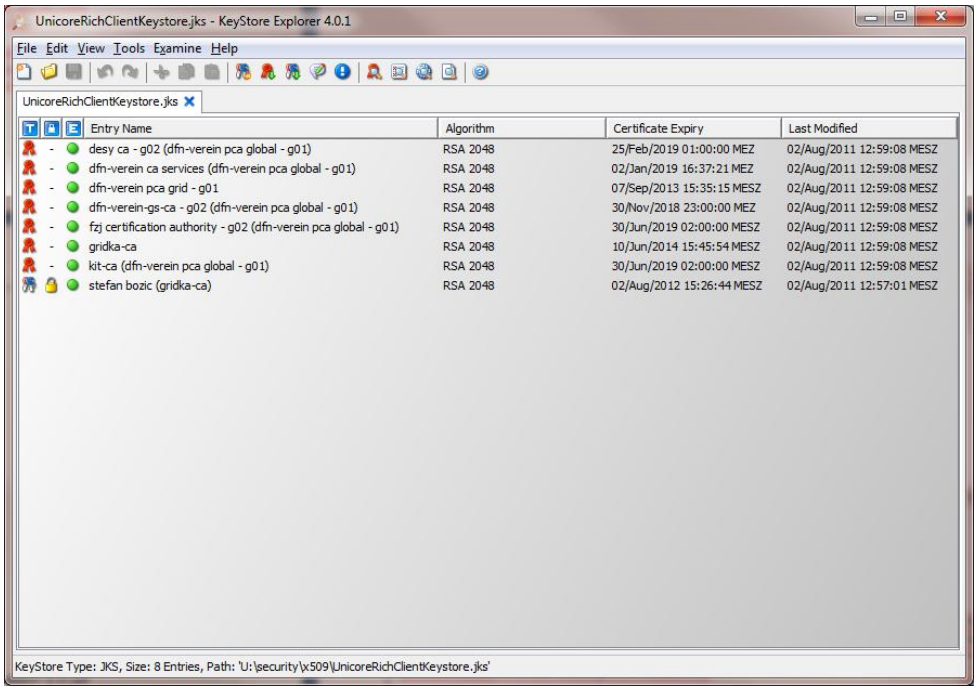



**Figure 5 Trusted certificates view in the Keystore Explorer**

Now that your keystore is set you are ready to proceed to the installation and launching of the UNICORE Rich Client.