Deliverable D2.3

New components' integration and deployment report

SEVENTH FRAMEWORK PROGRAMME
Research Infrastructures

INFRA-2007-1.2.2 - Deployment of eInfrastructures for scientific communities

Grant agreement for: Combination of Collaborative projects & Coordination and support actions

Proposal/Contract no.: 213010 – e-nmr

Project full title: Deploying and unifying the NMR e-Infrastructure in System Biology

Project coordinator: Prof. Dr. Harald Schwalbe

Project website: http://www.enmr.eu/

Period covered: from **01-11-2007 to 31-7-2009**

Project funded by the European Community

Table of contents

1. Introduction

1.1. Purpose

This document is the project deliverable D2.3 due by Month 21. It aims at reporting the activity of the task T2.3 which is about the integration into e-NMR grid of software components developed by the Joint Research Activities. The task is assigned to the Work Package 2 (WP2) of the e-NMR project.

1.2. Document organisation

The document is organised as follows:
Section 1 contains the purpose of the document, its references and a glossary of terms and acronyms.
Section 2 summarizes the content of the document.
Section 3 describes the organisation of the joint development platform.
Section 4 and 5 describes the software components developed in the reporting period for the benefit of the e-NMR applications.
Finally, Section 6 tracks the conclusions.

1.3. References

[R1]	Public link not available	*e-NMR Annex I*
[R2]	www.enmr.eu	*e-NMR project web portal*
[R3]	http://www.jspg.org/wiki/VO_Portal_Policy	*JSPG VO Portal Policy wiki page*
[R4]	http://www.biggrid.nl/infra/BiGGrid-VO-Portal-Policy-v1.pdf	*BigGRID VO Portal Policy document*

1.4. Terminology

This subsection provides the definitions of terms, acronyms, and abbreviations required to properly interpret this document.

Term	Definition
BCBR	Bijvoet Centre for Molecular Research, University of Utrecht, The Netherlands
BMRZ	Centre for Biomolecular Magnetic Resonance, Goethe University, Frankfurt, Germany
CA	Certification Authority
CE	Computing Element
CIRMMP	Interuniversity Consortium for Magnetic Resonance on Metalloproteins, Florence, Italy
CSR	Certificate Sign Request
DN	Distinguished Name
DoW	Description of Work
EGEE	Enabling Grids for e-Science
gLite	Codename of the middleware software suite developed by EGEE
GSI	Grid Security Infrastructure
INFN	National Institute of Nuclear Physics, Italy
JSPG	Joint Security Policy Group
LCG	LHC Computing Grid
LFC	LCG File Catalogue
NMR	Nuclear Magnetic Resonance
PM	Project Month
RA	Registration Authority
SE	Storage Element
SL	Scientific Linux
UI	User Interface
VO	Virtual Organization
VOMS	Virtual Organisation Membership Service
WMS	Workload Management System
WN	Worker Node
WP	Work Package

2. Executive summary

As stated in the DoW [**R1**], the main objective of the task T2.3 is

"to test and certify the components developed in the Joint Research Activities. In particular, their integration with the current tools and middleware deployed on the e-NMR infrastructure will be taken care of.
[…]
The whole activity may require a dedicated software repository and a bug tracking system to be setup at the GOC, and is of course strongly linked with the JRA."

In particular, the software components considered here are the ones produced by task T3.2 of WP3, as described in the DoW:

"*Some minor software development to help the gridification of e-NMR applications can also be performed within this task, e.g. code to enable GSI access to e-NMR portals for grid job submission, adaptation of existing tools to manage huge grid workloads, etc. However, neither new grid elements nor enhanced versions of existing gLite grid elements are planned to be released by the project."*

We worked in close cooperation with application developers to identify new components or services, additional to the basic gLite ones, that were required by them to better integrate into grid the e-NMR applications and make easier the user access and interaction.

During the first phase of the project two areas were identified which required the development of new components or adaptations from existing ones:

1. Web portal GSI authentication and user proxy delegation
2. Grid job automation and management

A software repository based on the Trac system (trac.edgewall.org) has been deployed to host the source code and documentation of all the services that have been developed.

3. The joint development platform

During the initial stages of the project software development has been mainly contained within each site, mostly so that different ways of providing grid based user services could be quickly and individually explored. However, now that most of the main technical issues are tackled and working services are online, it is crucial that all sites start to share their expertise, knowledge and code (plus descriptions) in one central resource. This step will help to identify further issues (for example which NMR data formats are in use and need to be interconverted to create a pipeline between resources), and is essential for the project to continue smoothly beyond the current funding period, and when core developers leave.

Several options for this joint development platform were discussed, and the final choice was the Trac system (http://trac.edgewall.org/), which had already been installed by CIRMMP and was in use by them and the INFN. It provides a Wiki, a timeline to track changes, support for project planning and issue tracking, as well as access to a subversion (SVN) source code repository.

This WEB-eNMR Software Development Site is accessible at https://websql-enmr.cerm.unifi.it/trac/WEBeNMR/. For security reasons, access to this site is restricted so only users with a login can participate (for evaluation purposes, please click on the *login* link at the top right of the main menu, then log in as user *eNmrGuest* with password *EU_evaluate!*).

In the initial stage of preparing the Trac site for general e-NMR use, we added a set of pages with discussion points to the Wiki. The ensuing discussions were very useful in preparing a consensus between the partners on how to use the joint development platform. In addition, pages with developer contact information and short descriptions of the software developed so far were introduced.

The next stage was to further discuss the joint development platform in person during an e-NMR developer's meeting at the BCBR (7^{th}-8^{th} July 2009). First, to explore the scope of the tools developed so far, all partners had the opportunity to explain their software in detail. A set of subdivisions for the code was then proposed and agreed upon:

1. **Web layer**: user interaction, HTML form handling, gathering data for program execution, etc. .
2. **Grid layer**: Job submission, job polling, SE data handling, etc. .
3. **'Worker' layer**: Scripts that run on worker nodes (*e.g.* to set up environment to run CNS).

These layers of code then serve to classify the software and simplify navigation.

The directory structure and use of the SVN repository was the next crucial issue. It was agreed that the directory structure in SVN (for sharing and tracking code) does not have to be the same as the directory structure used for deployment (to provide a live web service), but that there has to be some scope for consistently handling and organising generic code that is useful for all sites. The current structure, as shown in Figure 3-1, reflects this: a deployment directory contains directories for specific (stand-alone) web packages (e.g. HADDOCK), each of which contains the files required to deploy the package. The more generic code should ideally be connected to via symbolic links (or similar). A deployment script then creates from

this information a (working) setup. See https://websql-enmr.cerm.unifi.it/trac/WEBeNMR/wiki/eNmrDiscussion/DirectoryStructure.

These shared decisions on the Trac site organisation now allow the partners to deposit their code and initiate or follow up discussions on the Wiki site. We are also creating a resource that describes the available software in terms of the process it performs the input and dependencies it requires, and the output and log files it generates. Discussions about how to deal with software protocols are ongoing.

The shared development platform thus provides much clearer access to the e-NMR software (on the source code level), gives insights into the development process, and provides a single unified software structure that simplifies deployment of all e-NMR at the different sites. It will also help in making the e-NMR software available to external sites (*e.g.* industry).

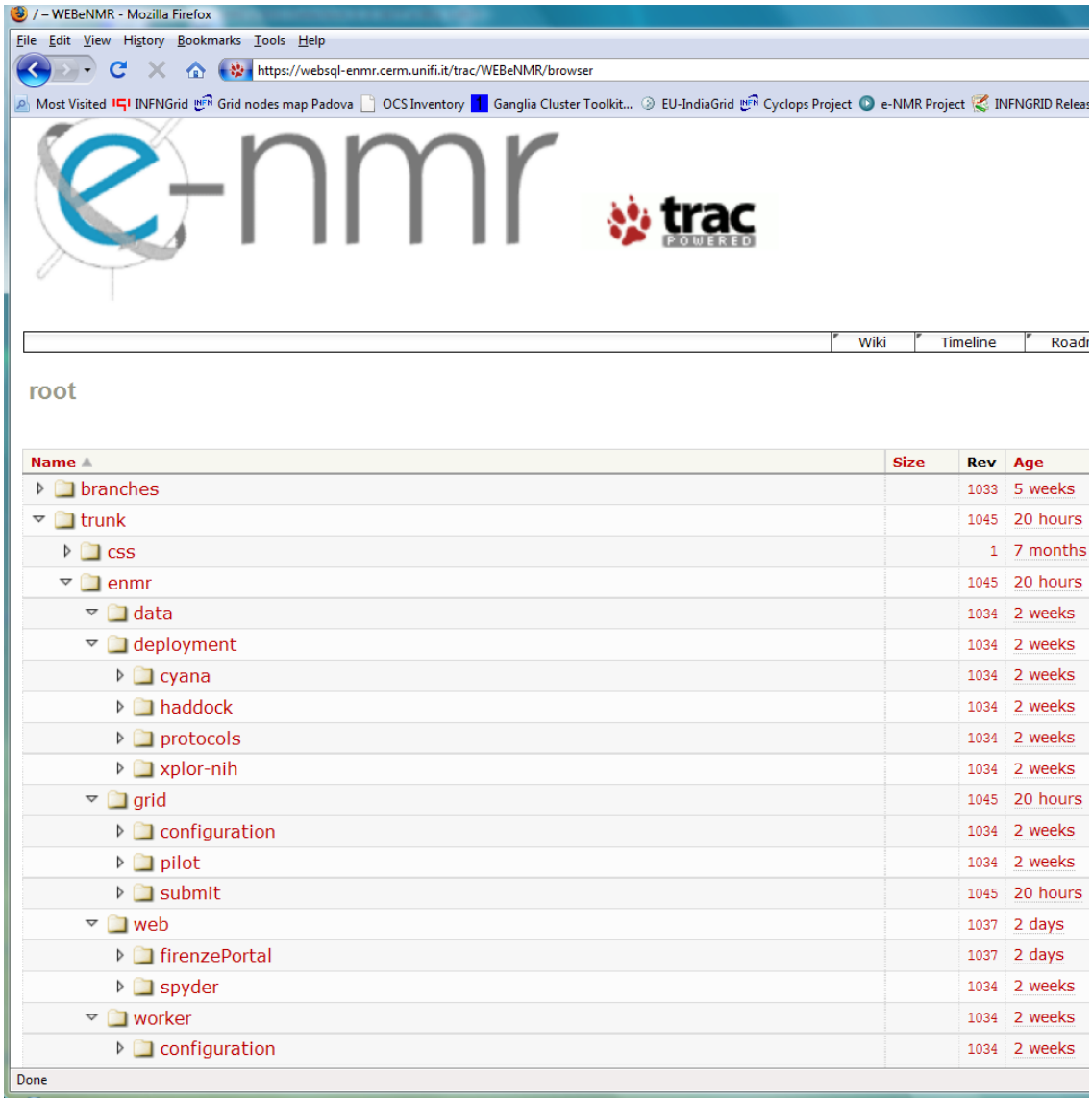

Figure 3-1: Snapshot of the directory structure of e-NMR software repository

4. Web portal GSI authentication and user proxy delegation

The first service developed by the project aimed at providing GSI secured access to the existing e-NMR portals initially developed to run the application on the local computing resources. In case of no use of grid resources, the user access was guaranteed via registration to each portal providing username and password. In case of use of grid resources, the access rights to the portals were instead guaranteed by requiring:

1) a X509 personal certificate issued by a IGTF accredited CA loaded into the users' browser
2) the personal certificate DN had to be registered with the enmr.eu VO

Behind the portal a business logic software component performed the needed operations to allow job submission over the e-NMR grid.
The first grid-enabled version of the HADDOCK, CS-Rosetta and Xplor-NIH portals implemented job submission making use of X509 robot certificate issued per application by NIKHEF and INFN CAs, according with the VO Portal Policy draft documented by the Joint Security Policy Group (JSPG) of EGEE.
In particular, the large number of jobs to be submitted and monitored by the HADDOCK and CS-Rosetta portals calls for automation. The current implementation of these two portals makes use of a robot certificate, qualifying them as a "parametric portal" according to the Joint Security Policy Group (JSPG) [**R3**]
(see http://www.jspg.org/wiki/VO_Portal_Policy#.E2.80.9CParameter.E2.80.9D_Portals).
The portal also obeys to the BigGRID (Dutch e-Science GRID) policies [**R4**].

A custom User Tracking System was implemented by portal developers in order to keep track of applications usage by each user, associating the DN of the users' certificate read at portal access time with the identifiers of grid jobs submitted with the robot on behalf of the user. Of course the usual grid accounting tools will see only the aggregate application usage, while the portal administrator will be always able to see the usage records of each user.

A second version of the Xplor-NIH portal was developed with the goal of replacing robot-based job management with one based on user credentials via the so-called proxy delegation mechanism.
In this version, the user accessing the portal delegates his credential to both the MyProxy server and the portal web server. The latter will generate a proxy certificate to be used for job submission on users' behalf. The interaction between the user and the delegation service is managed by a java applet.
The new architecture is shown in Figure 4-1. The generation of a proxy certificate is as follows. The user does contact the delegation service running on the web server. The service creates a public-private key pair and uses it to generate a Certificate Sign Request (CSR). This is a certificate that has to be signed by the user with his private key. The signed certificate is then sent back to the service. This procedure is similar to the generation of a valid certificate by a CA and, in fact, in this context the user acts like a CA. The certificate generated so far is then combined with the user certificate, thus forming a chain of certificates. The service that examines the proxy certificate can then verify the identity of the user that delegated its credentials by unfolding this chain of certificates. Once the proxy has been created and associated to the user ID in the web server database, it is transferred on the

UI local to the web server LAN and behind a firewall. Decoupling the web server hosting the portal from the UI performing the job submission adds a further level of security. In fact, user proxies created in the web server are immediately destroyed after their transfer to UI. On the other hand, the UI is carefully protected inside the private network, with the minimum set of communication ports left open in order to interact with the e-NMR grid services (VOMS, WMS and MyProxy).
Jobs are submitted to gLite WMS using the short-lived proxy (12 hours of lifetime). Jobs lasting more than 12 hours will have the short-lived proxy automatically renewed before its expiration via the gLite proxy renewal mechanism acting at WMS level using the long-lived credentials (with maximum lifetime of two weeks) stored in the MyProxy server.

The suitability of this new portal implementation with proxy delegation mechanism for implementation in the HADDOCK and CS-Rosetta portals (both requiring a smooth automation with a very large volume of grid jobs to be submitted and monitored) is under study.

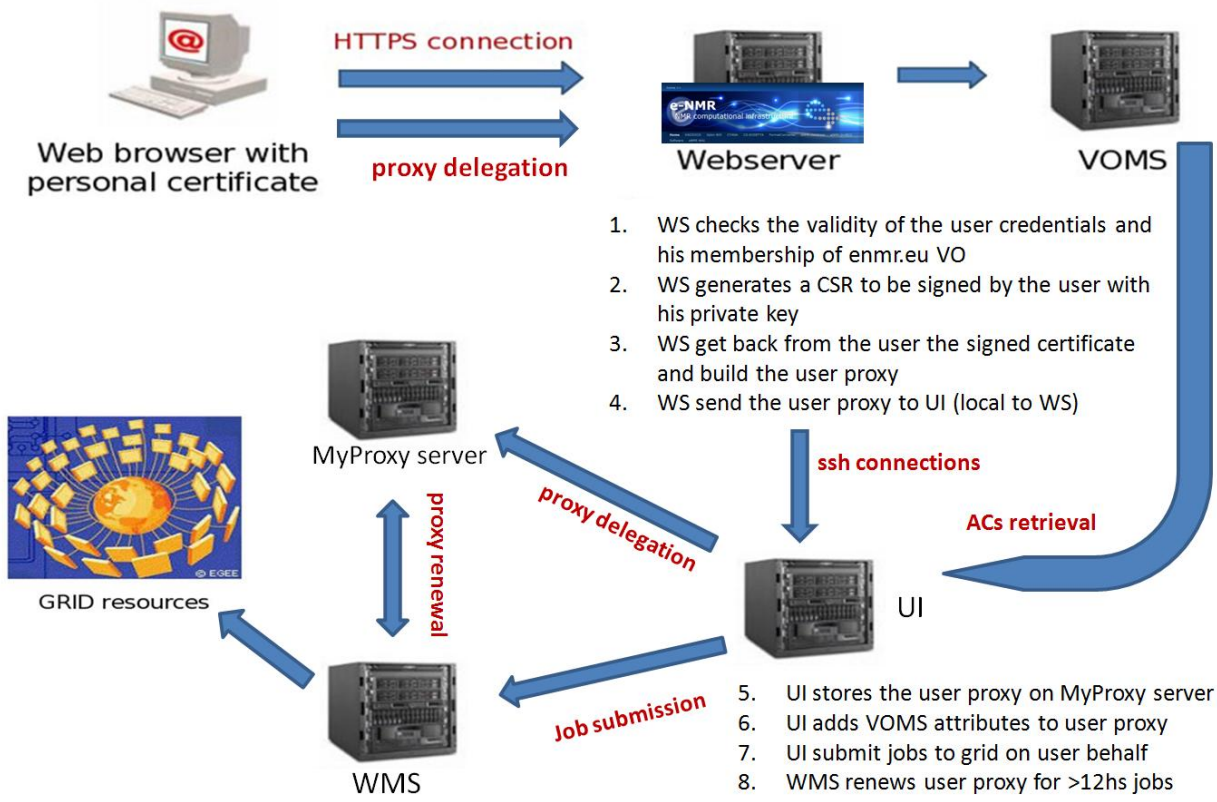

Figure 4-1: Web portal security architecture

The security risks associated with model described above are not higher than the ones normally accepted by users willing to run their applications on the EGEE grid. These risks are actually lower in our case, if we consider that in other communities most of the grid users keep their personal X509 certificate and private key permanently on the $HOME/.globus area of shared UIs provided by their institutes. In our model the private key is used only to sign via applet the CSR generated by the web server, so it can be kept secure in a USB key or in the owner's laptop. Short-lived proxy certificates can of course be stolen by external attackers managing to get fraudulent access to UI, WMS, MyProxy server as well as the distributed Worker Nodes where jobs are running, other than by internal administrators of these services,

but this is a general issue of the EGEE production grid. It is under the responsibility of the grid site managers to enforce strict policy on the network use to avoid the chance of their site being liable for playing a role in any malicious use such as attacks on other connected systems on the Internet.

4.1. Xplor-NIH portal security implementation

The web portal hosted in CIRMMP (web-enmr.cerm.unifi.it) can manage two different types of proxy certificates to access grid resources:

1) robot proxy certificate generated from hardware token
2) user proxy certificate generated from java applet using personal certificate

Proxy robot certificates are created by an eToken PRO USB smart card token that contains a robot certificate issued by the INFN CA. In this device the private key of X509 certificate is stored in the internal memory and cannot be read by the system. It can be used only to create a proxy certificate.
To maintain the grid access control using robot certificate the web site manages an HTTPS connection that checks if the personal certificate installed in the local browser of the user is released by a IGTF recognized CA and if the owner is member of enmr.eu VO.

Web-enmr portal manages also personal certificates to create a proxy, through a java applet that runs on the local machine of the user. The proxy is created on the web portal machine using the gLite delegation service installed on it, with the mechanism described in Figure 4-1, and is used to perform grid job submission on behalf of the user.

The site manages a unique session for every user connected with a valid certificate, and stores in a DB an univocal association between the DN of the certificate and the resources used at the moment (every grid job identifier submitted, user quota).
This permit to maintain a registry of the users that run a job with a robot certificate, that otherwise would be lost.
The site permits to switch between personal and robot certificate in a smooth a user friendly way. For robot and personal certificates is possible to delegate the proxy to a MyProxy server, this permit to extend the proxy duration from 12 hours to 7 days. The site also permits to check the validity of user credential.

The following subsections contain a detailed description of the delegation client applet and of the gLite delegation service. These contain links to the Trac system repositories where the produced code can be found and used to implement the same mechanism on the other portals, following the instructions reported here.

4.1.1. *DelegationClientApplet* applet on the web-enmr portal

The *DelegationClientApplet* applet is a client of the gLite delegation service. In this section, we describe how the *DelegationClientApplet* applet has been integrated in the web-enmr portal and how it can be used in order to perform delegation.

These are the requirements needed to execute the *DelegationClientApplet* applet:

- java runtime environment (1.5 or later) (see http://java.sun.com/javase/downloads/index.jsp)
- java plug-in to enable java applet to run in web browser *(see java plug-in for your browser)*
- installation of the "*Unlimited Strength Jurisdiction Policy Files"* (see section 4.1.1.2)
- modification of the policy file to grant permission to the applet. (see section 4.1.1.2)

4.1.1.1. How to create a proxy

Click on the "*Proxy->Create Proxy From Applet*" on the portal menu and the credential form is shown.

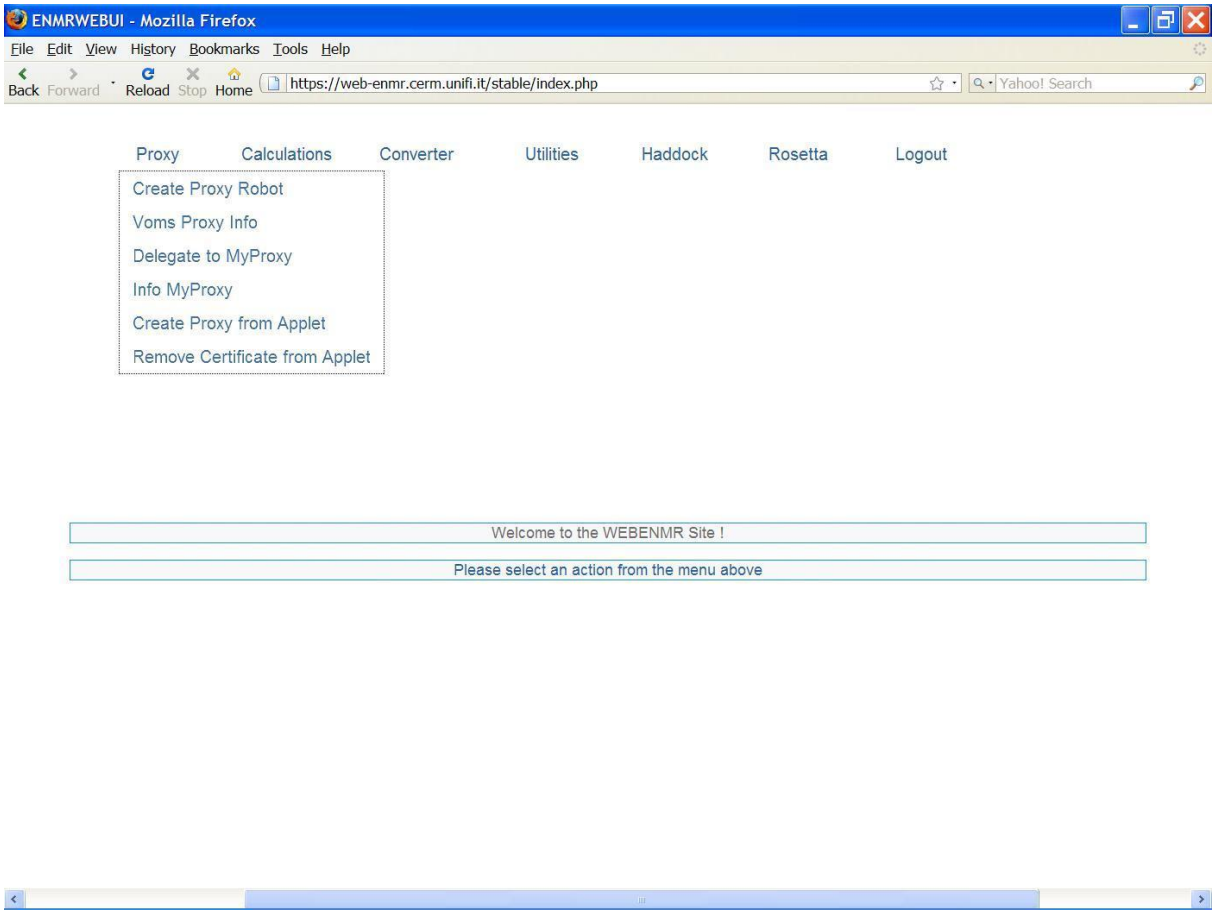

Figure 4-2: Web-enmr portal proxy menu

Fill in the form with the file path of the certificate (p12/pfx format) and the corresponding passphrase.

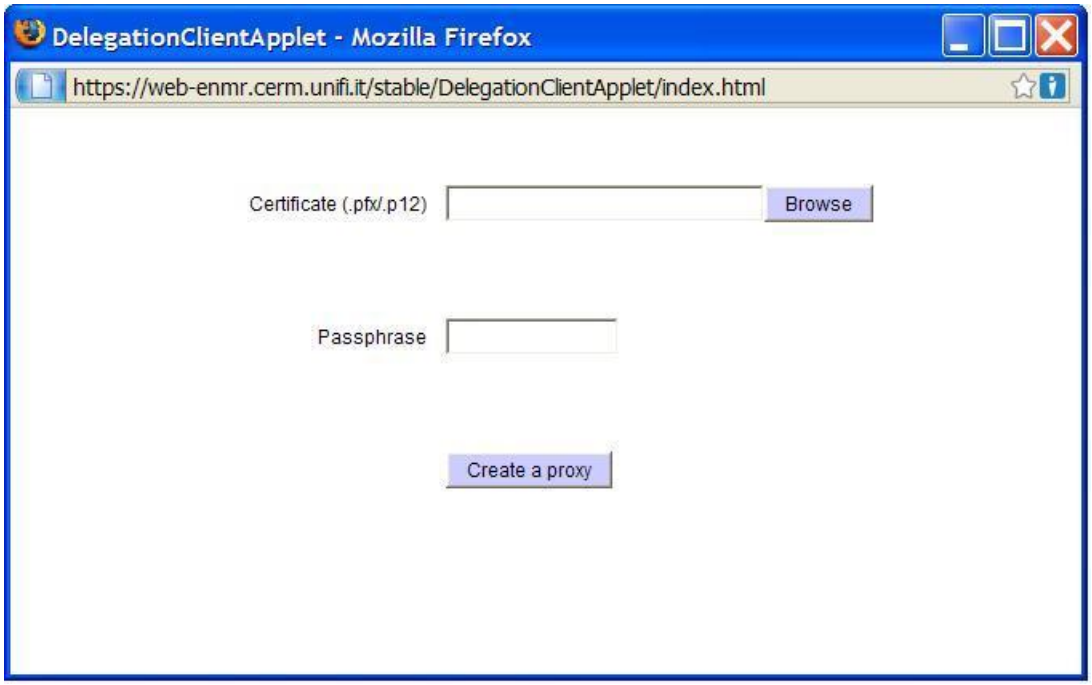

Figure 4-3: Credential form

Now, click the "*Create proxy*" button. After a while a box appears to communicate that your proxy has been created successfully!

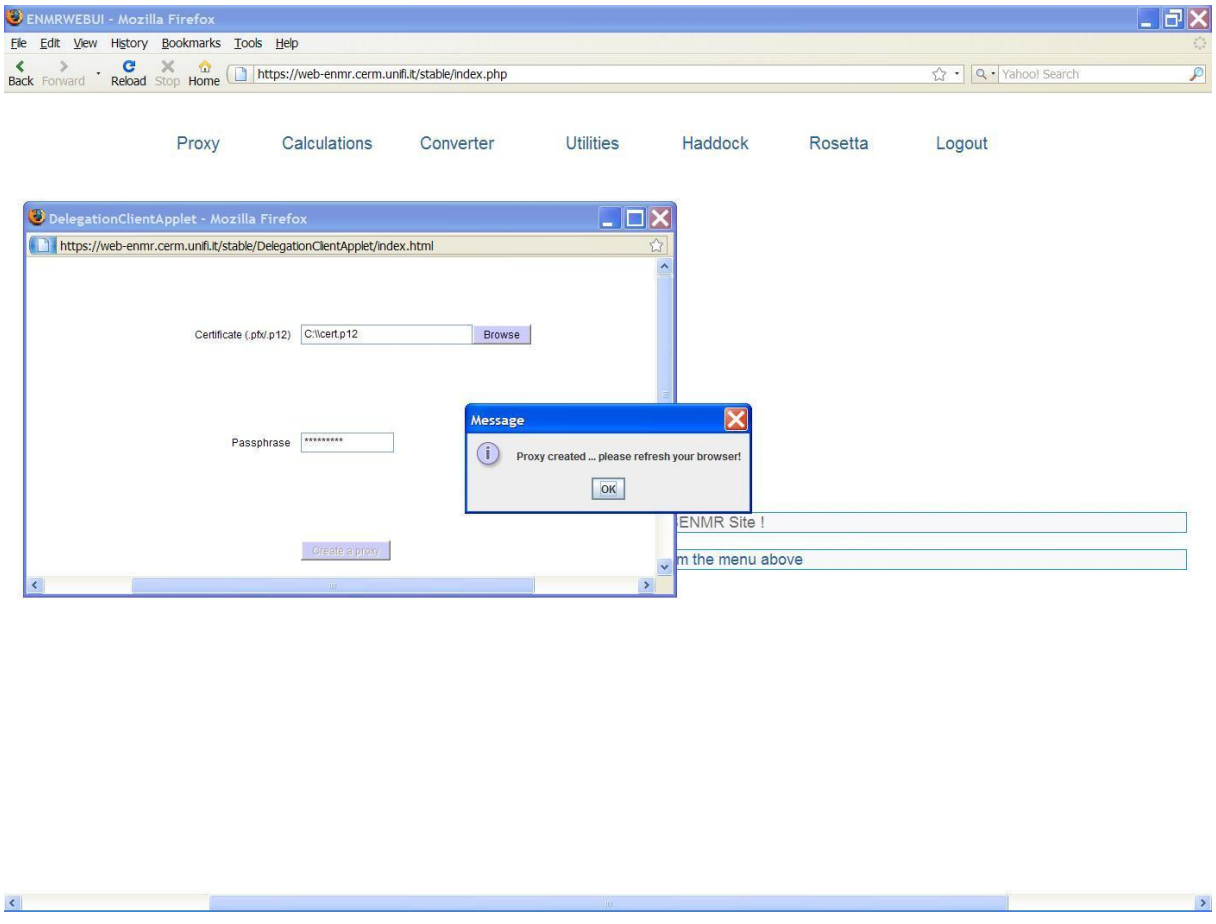

Figure 4-4: Proxy created successfully

Then the portal confirms the creation of the proxy and shows the related lifetime: "*You have a personal certificate present in database that will expire in 170 hours*".

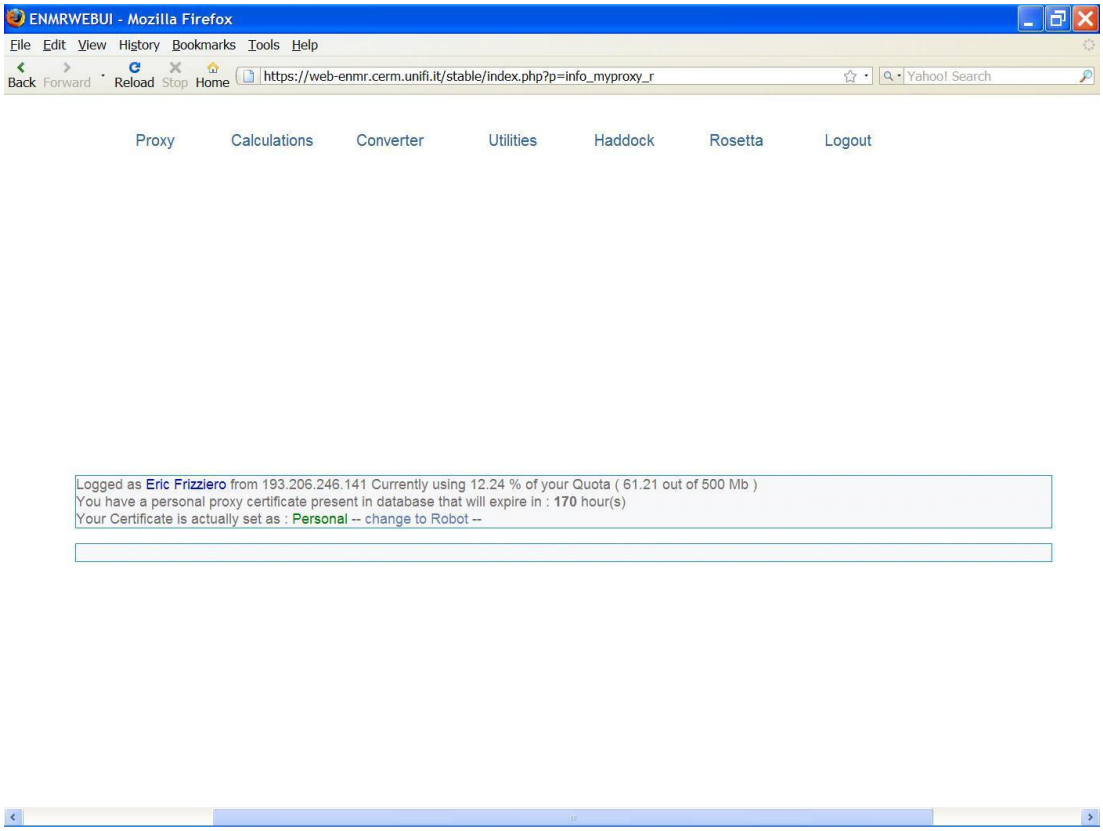

Figure 4-5: Information about user credential

4.1.1.2. DelegationClientApplet applet requirements

In this paragraph, we see how to install the "Unlimited Strength Jurisdiction policy files" and how to modify the policy file (java.policy file) to grant the required permission.

Installation of the Unlimited Strength Jurisdiction Policy Files

The normal JDK installation has a set of policy files that places certain restrictions on the key sizes that can be used. The easiest way to deal with this restriction if it need not apply to you is to download the unrestricted policy files.

You can find the unrestricted policy files on the same page as the JCE/JDK downloads are found.

Normally there is a link at the bottom of the download page named as "*Unlimited Strength Jurisdiction Policy Files*".

Other Downloads

You should download the ZIP file and install the two JAR files it contains according to the instructions in the README file contained in the ZIP file.

You need to make sure you install the policy files in the Java runtime you are using, and you will probably need root access or the assistance of a root user to do so, unless you have installed a personal runtime of your own.

If the unrestricted policy files are not installed, the *DelegationClientApplet* applet requires that updating.

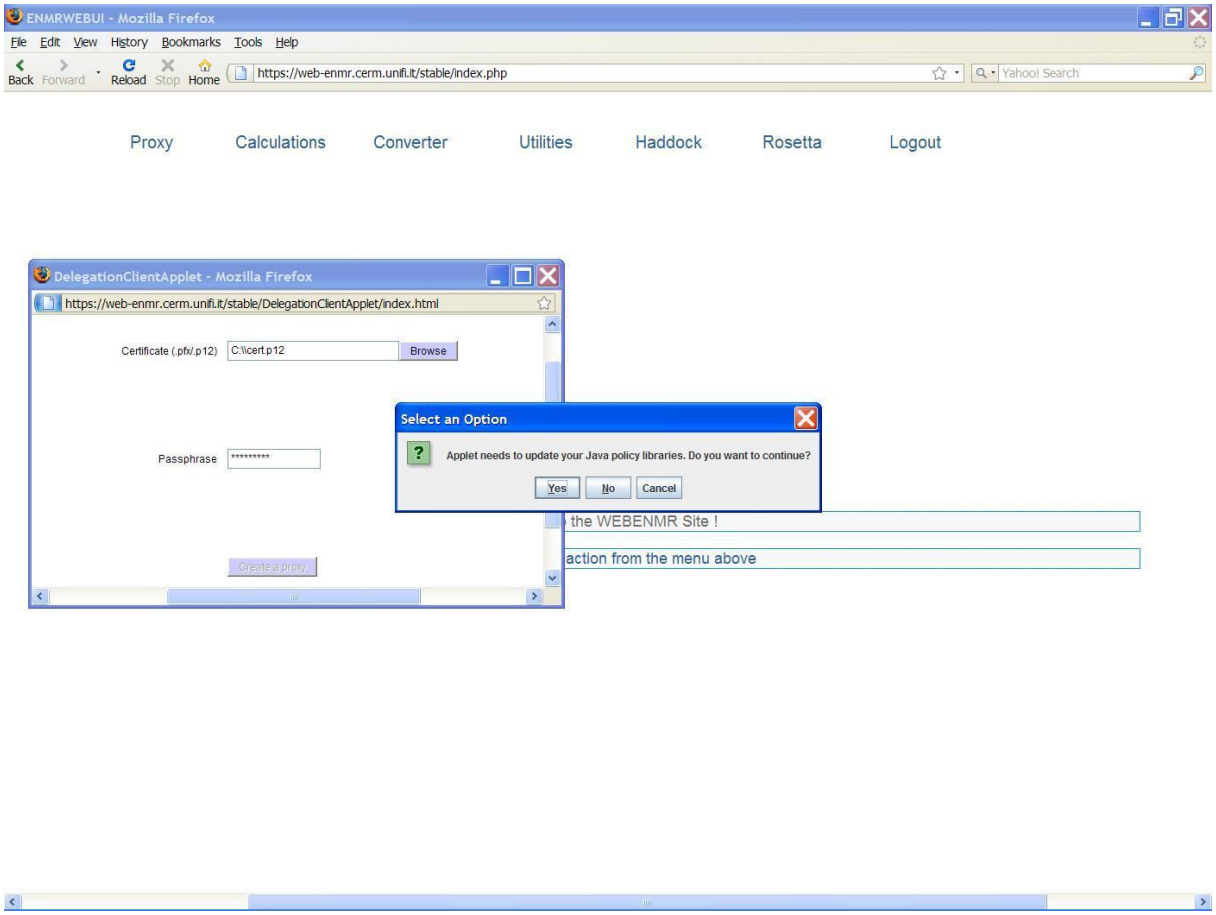

Figure 4-6: Applet needs to update your java policy libraries

Also in this case, you will probably need root access or the assistance of a root user to do so, unless you have installed a personal runtime of your own.

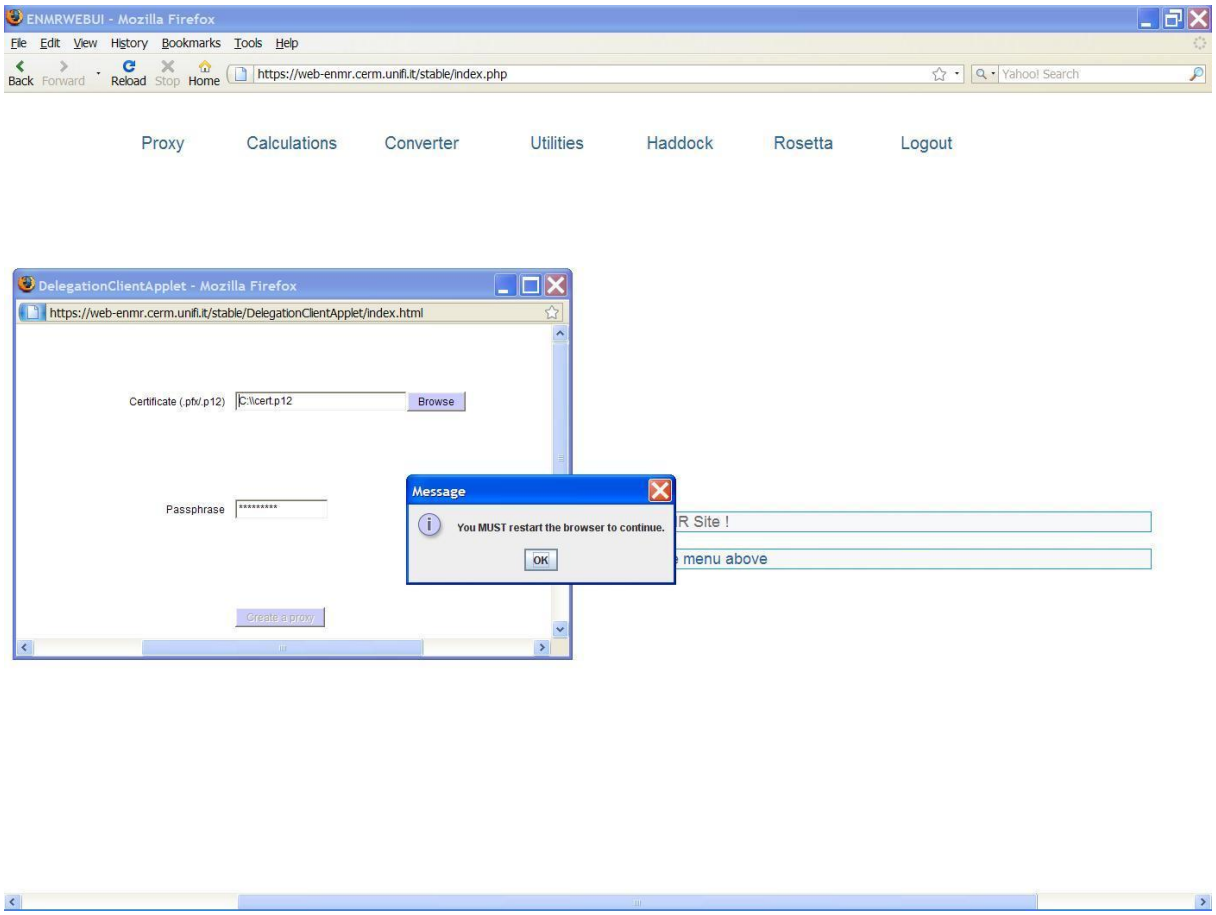

Figure 4-7: Update executed successfully

Modification of the Policy File to grant the required permission

It is necessary to grant permission to the *DelegationClientApplet* applet in order to be executed correctly. By default an applet runs in a secure sandbox that prevents it from interacting with the users system, unless authorized.
To grant permission to the applet, you need to modify the *java.policy* file of the java runtime environment you are using.
A policy file is an ASCII text file and can be modified via a text editor or the graphical Policy Tool utility.

Using text editor

Open the *java.policy* file of the java runtime environment you are using and create the following policy entry granting permission:

```
grant codeBase "https://web-enmr.cerm.unifi.it/-" {

permission java.security.AllPermission;

};
```

N.B: you will probably need root access or the assistance of a root user to do so, unless you have installed a personal runtime of your own.

Using Policy Tool

The Policy Tool relieves you of the hassle of typing and knowing eliminates the required syntax of policy files, thus reducing errors.

Follow these steps to modify your policy file:

1. Start policy tool
2. Select the policy file and grant the required permission
3. Save the policy file

1) Start Policy Tool

To start Policy Tool, simply type the following at the command line:

```
policytool
```

This brings up the Policy Tool window.

2) Select the policy file and grant the required permission

Use the Policy Tool menu, open the *java.policy* file of the java runtime enviroment you are using.

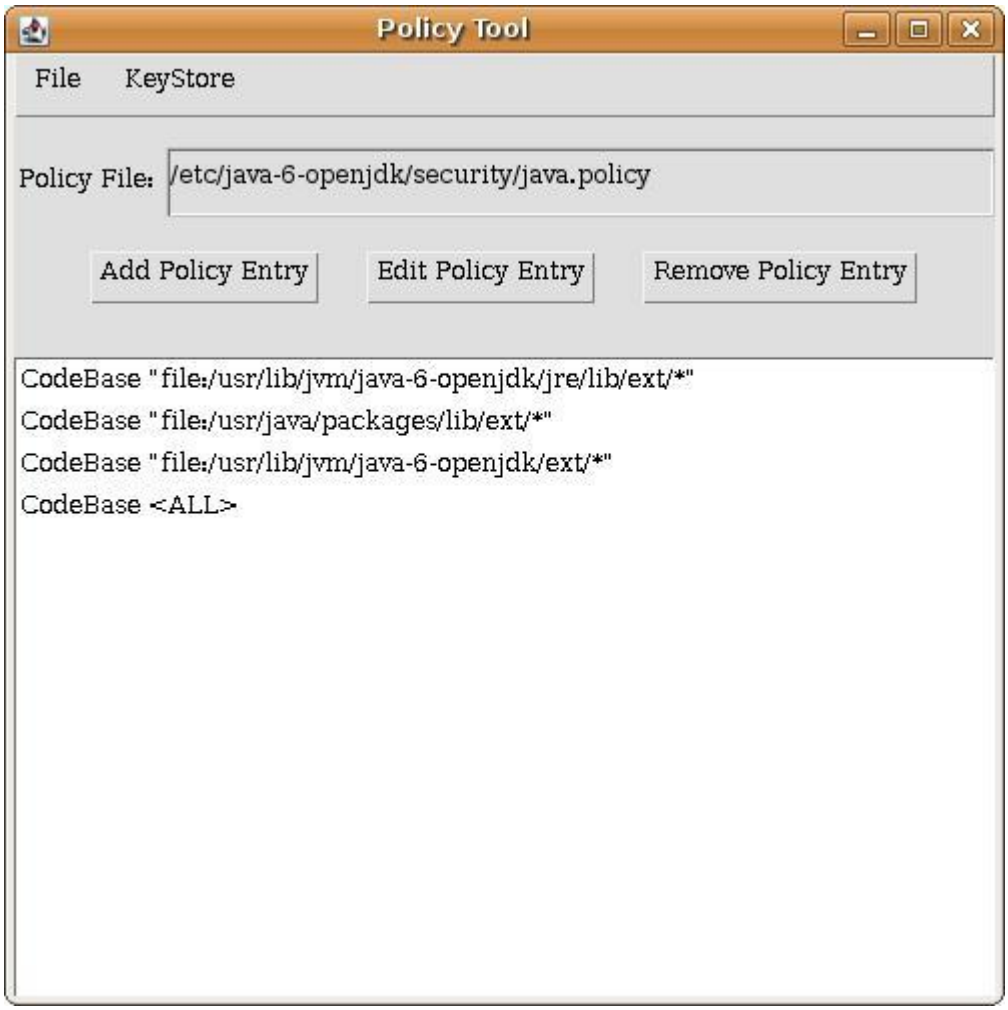

Figure 4-8: Selection of the policy file

After that, to grant the *DelegationClientApplet* applet permission, you must create a policy entry. To create a new entry, click on the **Add Policy Entry** button in the main Policy Tool window. This displays the policy entry dialog box as shown in the following figure.

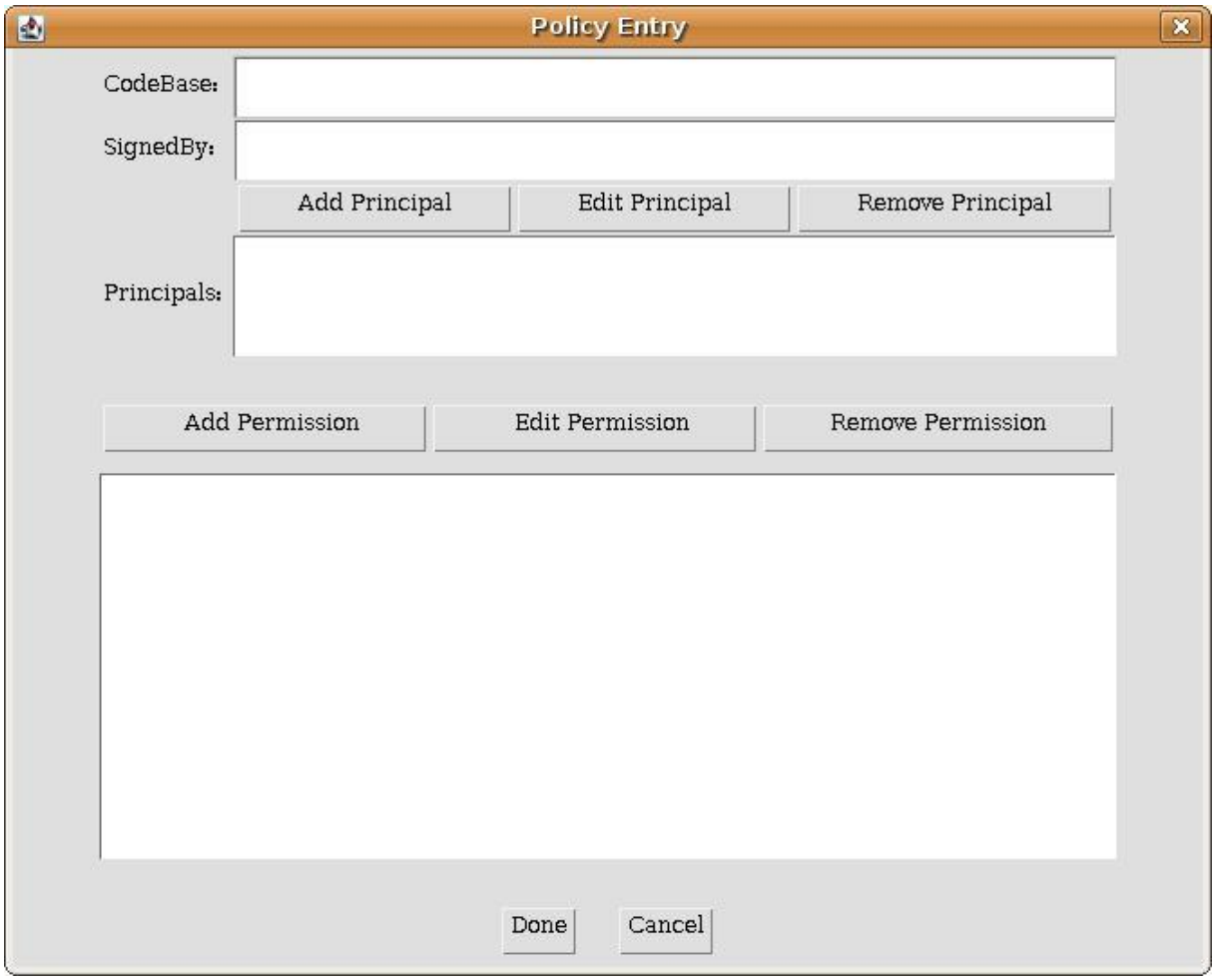

Figure 4-9: Policy entry dialog box

A *CodeBase* value indicates the code source location; you grant the permission(s) to code from that location.

Type the following URL into the *CodeBase* text box of the Policy Entry dialog box:

`https://web-enmr.cerm.unifi.it/-`

web-enmr.cerm.unifi.it is the host where the *DelegationClientApplet* applet comes from.

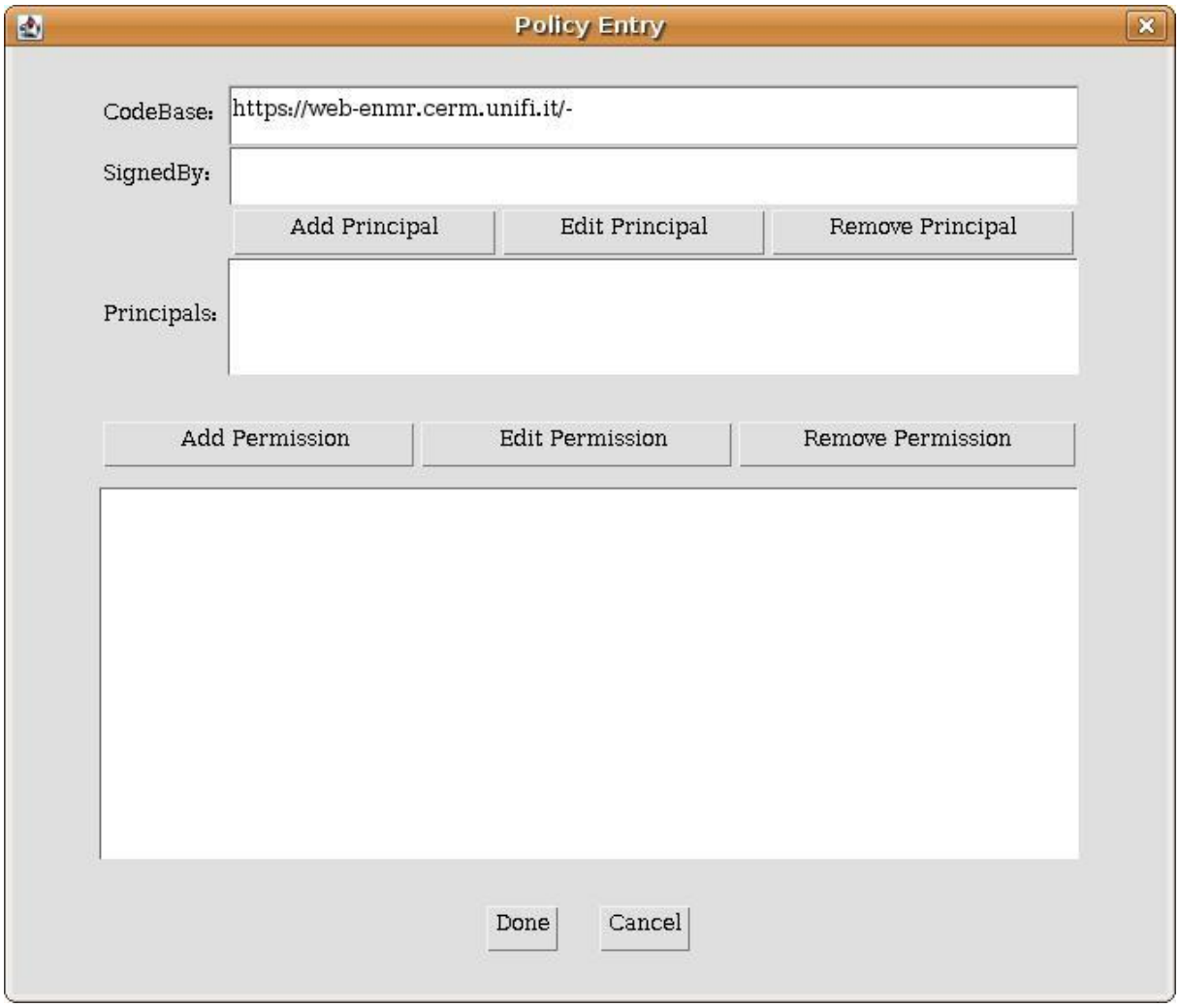

Figure 4-10: Inserting the codebase value

Click on the **Add Permission** button to display the Permissions dialog box. Choose **AllPermission** from the Permission drop-down list. The permission dialog box now looks like the following:

Figure 4-11: Permission dialog box

Click on the **OK** button. You have now specified this policy entry, so click on the **Done** button in the Policy Entry dialog. The Policy Tool window now contains a line representing the policy entry, showing the CodeBase value.

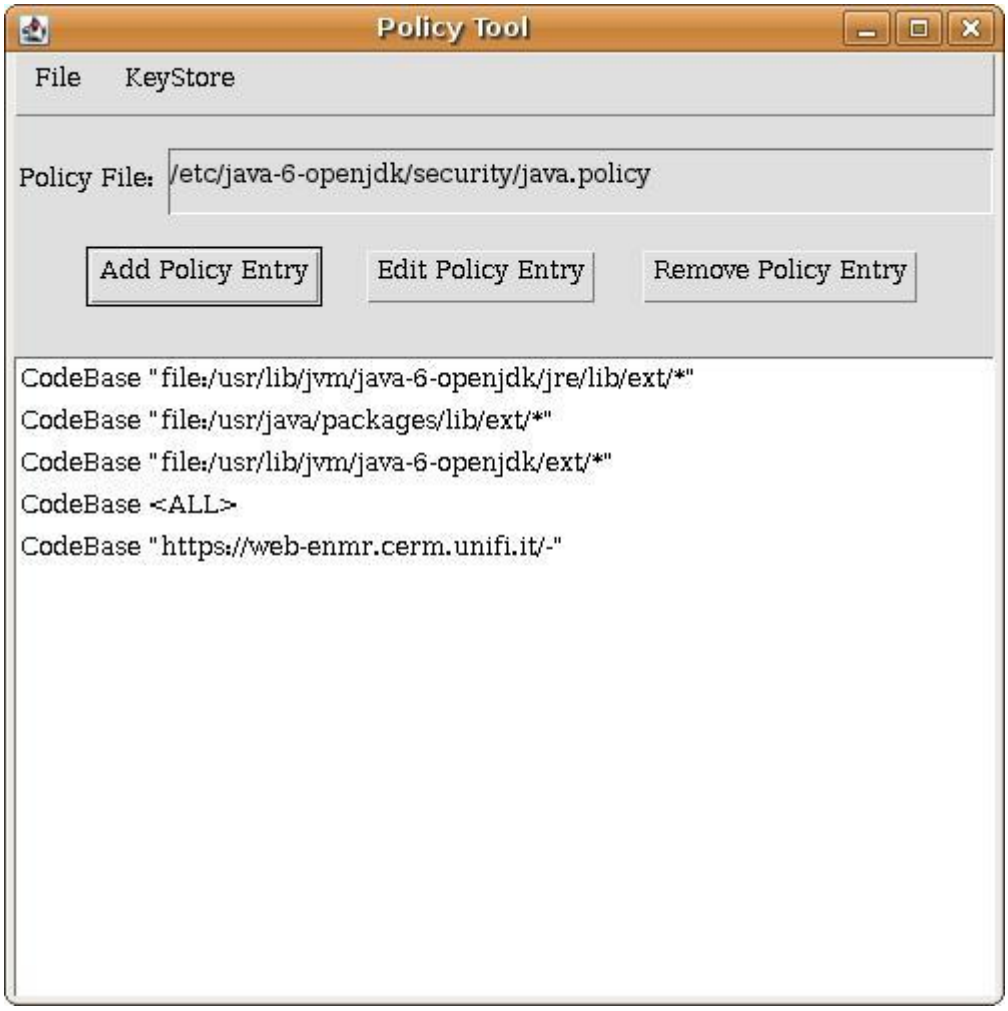

Figure 4-12: Policy tool window updated

3) Save the Policy File

To save the policy file modified, choose the **Save** command from the **File** menu.

N.B: you will probably need root access or the assistance of a root user to do so, unless you have installed a personal runtime of your own.

After that, exit Policy Tool by choosing **Exit** from the **File** menu.

4.1.1.3. Download DelegationClientApplet software

The DelegationClientApplet software is stored in a Subversion repository (SVN) at the following URL:

https://websql-enmr.cerm.unifi.it/trac/WEBeNMR/browser/trunk/enmr/web/firenzePortal/java/delegationClientApplet

In order to install and configure the *DelegationClientApplet* applet, you can use the archive file available at:

https://websql-enmr.cerm.unifi.it/trac/WEBeNMR/browser/trunk/enmr/web/firenzePortal/java/delegationClientApplet/dist/delegationClientApplet.tgz

After having uncompressed the file in your HTTP web server:

```
tar xzvf delegationClientApplet.tgz
```

you have to modify the applet parameter *delegationService* in the *index.html* file indicating the URL of the delegation service that the applet has to contact to perform delegation.

Example:

```
<PARAM NAME="delegationService" VALUE="https://web-enmr.cerm.unifi.it:8443">
```

4.1.2. Installation of standalone gLite delegation service

In this section, we describe how to install and deploy the gLite delegation service in an Apache Tomcat.

These are the requirements needed to install the delegation service:

- Apache Tomcat (5.5 or later) already installed and consequently the correct Java environment. Just make sure to set the CATALINA_HOME environment variable to point to the Apache Tomcat installation location (see http://tomcat.apache.org).
- A Mysql database server (4.1 or later) already installed (see http://www.mysql.com).

Moreover the delegation service requires the gLite trustmanager component to be installed in the Apache Tomcat (see the next section 4.2.2.1).

4.1.2.1. Download delegation-service software

In order to install and configure the delegation service easily, you can use the archive file available at:

http://etics-repository.cern.ch:8080/repository/download/registered/org.glite/org.glite.security.delegation-service-java/1.3.0/noarch/glite-security-delegation-service-java-1.3.0-1.tar.gz

After having uncompressed the file:

```
tar xzvf glite-security-delegation-service-java-1.3.0-1.tar.gz
```

you have to copy the *share/java/glite-security-delegation-service-java.war* file in

$CATALINA_HOME/webapps/delegation-service.war file and start the Apache Tomcat in order the *$CATALINA_HOME/webapps/delegation-service* directory to be created. Then you must stop Apache Tomcat for continuing the configuration of the delegation service.

N.B: if the *$CATALINA_HOME/webapps/delegation-service* directory is not created, then there are some installation or configuration problems in your Apache Tomcat (see http://tomcat.apache.org).

4.1.2.2. Delegation service configuration

The proxy created by the delegation service can be stored in the file system or in a database. Now we see how to configure the service to store proxy on a database named ***enmrdlgDB.***

In the following examples *database_name* must be replaced with *enmrdlgDB.*

dlgee.properties file

To allow to the delegation service to store the proxy on a mysql database some changes should be done. Edit and modify the *WEB-INF/classes/dlgee.properties* file:

```
#dlgeeStorageFactory=org.glite.security.delegation.storage.GrDPStorageFilesystemFactory
#delegationStorage=/tmp/dlgee_delegationStorage
dlgeeStorageFactory=org.glite.security.delegation.storage.GrDPStorageDatabaseFactory
dlgeeStorageDbPool=jdbc/database_name
dlgeeKeySize=512
```

delegation-service.xml file

You have to create the */usr/share/tomcat5/conf/Catalina/localhost/delegation-service.xml* file:

```
<Context path="/delegation-service" docBase="delegation-service"
privileged="true" antiResourceLocking="false" antiJARLocking="false">
<Resource name="jdbc/database_name"
auth="Container"
type="javax.sql.DataSource"
factory="factory"
driverClassName="org.gjt.mm.mysql.Driver"
username="database_user"
password="database_password"
maxActive="500"
maxIdle="30"
maxWait="10000"
logAbandoned="false" removeAbandoned="true" removeAbandonedTimeout="30"
url="jdbc:mysql://localhost:3306/database_name?autoReconnect=true"
```

```
validationQuery="SELECT 1"
testOnBorrow="true"
testWhileIdle="true"
timeBetweenEvictionRunsMillis="20000"
minEvictableIdleTimeMillis="90000"
/>
</Context>
```

where:

database_name is the database name: *enmrdlgDB;*

database_user is the database user;

database_password is the corresponding password of the database user;

factory is the name of the factory for the jdbc resource.

In order to indentify the factory to be used, you should execute the following command:

```
jar tvf $CATALINA_HOME/common/lib/*commons-dbcp*.jar | grep BasicDataSourceFactory
```

Some outputs of that command should be like these:

1) *7459 Fri Aug 18 15:05:36 CEST 2006 org/apache/commons/dbcp/BasicDataSourceFactory.class*

2) *7400 Tue Jul 15 17:11:40 CEST 2008 org/apache/tomcat/dbcp/dbcp/BasicDataSourceFactory.class*

In the former case:

factory="org/apache/commons/dbcp/BasicDataSourceFactory.class"

in the latter case:

factory="org/apache/tomcat/dbcp/dbcp/BasicDataSourceFactory.class"

web.xml file

Check the presence of the "*jdbc/enmrdlgDB*" resource definition in the *webapps/delegation-service/WEB-INF/web.xml* file:

```
<!-- Database holding the request store. -->
<resource-ref>
<description>Delegation storage database</description>
```

```
<res-ref-name>jdbc/database_name</res-ref-name>
<res-type>javax.sql.DataSource</res-type>
<res-auth>Container</res-auth>
</resource-ref>
```

where *database_name* must be replaced with ***enmrdlgDB***.

delegation service database

Finally, you need to create the delegation service database by using the following SQL script:

```
/************ Create: Database ***************/
CREATE DATABASE enmrdlgDB;

/************ Use: Database ***************/
USE enmrdlgDB;
/********************* Add Tables *************************/

CREATE TABLE t_credential_cache (
dlg_id VARCHAR(100),
dn VARCHAR(255),
cert_request TEXT NOT NULL,
priv_key TEXT NOT NULL,
voms_attrs TEXT,
PRIMARY KEY (dlg_id, dn)
) ENGINE=InnoDB;

CREATE TABLE t_credential (
dlg_id VARCHAR(100),
dn VARCHAR(255),
proxy TEXT NOT NULL,
voms_attrs TEXT,
termination_time DATETIME,
PRIMARY KEY (dlg_id, dn)
) ENGINE=InnoDB;

CREATE TABLE t_credential_vers (
major INT,
minor INT,
patch INT
) ENGINE=InnoDB;

commit;
```

```
/************** Insert row in t_credential_vers table ***************/

insert into t_credential_vers(major, minor, patch) values ('1','2','0');

commit;
```

4.1.2.3. Installation of the gLite trustmanager component

The delegation service works fine if the trustmanager component is installed and configured correctly in the Apache Tomcat.

Download trustmanager component

In order to install and configure the trustmanager component easily, you can use the archive file available at:

http://etics-repository.cern.ch:8080/repository/download/registered/org.glite/org.glite.security.trustmanager/1.8.16/noarch/glite-security-trustmanager-1.8.16-3.tar.gz

Apache Tomcat configuration for trustmanager component

After having uncompressed the file:

```
tar xzvf glite-security-trustmanager-1.8.16-3.tar.gz
```

you need to configure the Apache Tomcat in order the trustmanager component is used.

Host credentials and CA certificates

First of all make sure you have the host credentials (*hostcert.pem* and *hostkey.pem* in */etc/grid-security* directory). Also make sure you have all the necessary CA certificates and CRL description files in */etc/grid-security/certificates* directory.

Configure Apache Tomcat with SSL

Apache Tomcat comes with a built-in SSL implementation to allow you to connect securely via "https". However, it is disabled by default. You need to edit the *server.xml* configuration file to enable SSL.

In *share/glite-security-trustmanager* directory we find the *server.xml.template* file:

```
<Connector port="@PORT@"

maxThreads="150" minSpareThreads="25" maxSpareThreads="75"

enableLookups="false" disableUploadTimeout="true"

acceptCount="100" debug="0" scheme="https" secure="true"
```

```
sSLImplementation="org.glite.security.trustmanager.tomcat.TMSSLImplementation"
sslCAFiles="@CAFILES@"
crlFiles="@CRLFILES@"
sslCertFile="@SSLCERTFILE@"
sslKey="@SSLKEY@"
log4jConfFile="@LOG4JCONF@"
clientAuth="true" sslProtocol="TLS"
crlEnabled="true" crlRequired="true"/>
```

So go to the *$CATALINA_HOME/conf* directory and edit the *server.xml* file. Look for an XML block that starts with:

```
<-- Define a SSL HTTP/1.1 Connector on port 8443 -->
<--
<Connector port="8443" ...
... />
-->
```

Remove the comments (<-- -->) from around this <Connector.../> xml block to enable the SSL Connector and insert the correct information.

Example:

```
<Connector port="8443"
maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
enableLookups="false" disableUploadTimeout="true"
acceptCount="100" debug="0" scheme="https" secure="true"
SSLImplementation="org.glite.security.trustmanager.tomcat.TMSSLImplementation"
sslCAFiles="/etc/grid-security/certificates/*.0"
crlFiles="/etc/grid-security/certificates/*.r0"
sslCertFile="/usr/share/tomcat5/hostcert.pem"
sslKey="/usr/share/tomcat5/hostkey.pem"
log4jConfFile="/usr/share/tomcat5/conf/log4j-trustmanager.properties"
clientAuth="true" sslProtocol="TLS"
crlEnabled="true" crlRequired="true"/>
```

Install libraries

The following libraries have to be added to the *$CATALINA_HOME/server/lib* directory:

- *glite-security-util-java.jar*

  you can find it at the following link:

  http://etics-repository.cern.ch:8080/repository/download/registered/org.glite/org.glite.security.util-java/1.4.0/noarch/glite-security-util-java-1.4.0-1.tar.gz

- *glite-security-trustmanager.jar*

  you can find it at */share/java* directory (*glite-security-trustmanager.jar* file).

- *log4j jar*

  you can find it at */share/glite-security-trustmanager* directory (*log4j-1.2.14.jar* file).

- *bouncycastle jar*

  you can find it at */share/glite-security-trustmanager* directory (*bcprov-1.37.jar* file).

Configure log4j for trustmanager

Setup the log4j file defined in *server.xml* file. You can use the *etc/glite-security-trustmanager/log4j-trustmanager.properties* file and copy it in the file path specified in the *log4jConfFile* tag.

4.1.2.4. Check the delegation service

Now, everything should be in place and configured correctly. So, you can start Apache Tomcat and the delegation service should be available at the link:

https://<host>:8443/delegation-service/services

<*host*>: the host where the delegation service has been installed.

Example:

https://web-enmr.cerm.unifi.it:8443/delegation-service/services

If the delegation service is installed, deployed and configured correctly, you should display:

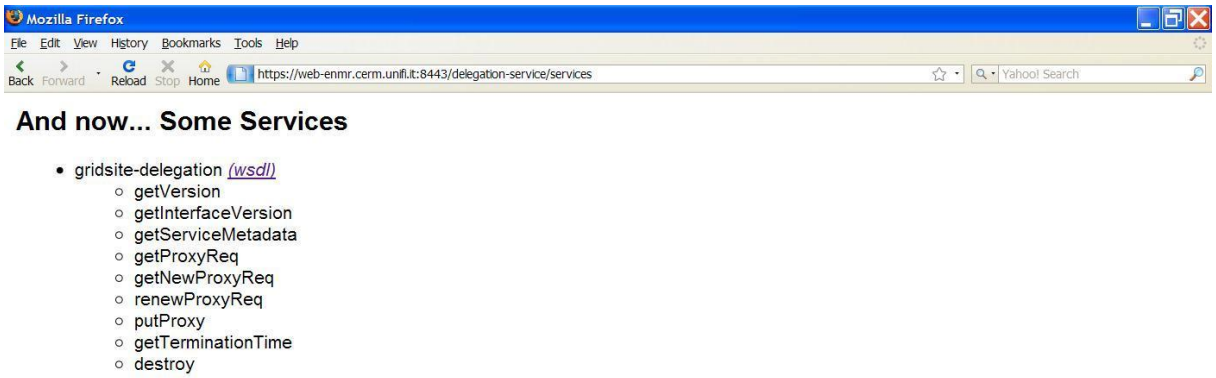

Figure 4-13: Delegation service deployed

Now, a delegation client can be run to perform delegation with the running service.

5. Grid job automation and management

To streamline the processes involved in handling of user requests and performing grid calculations, two novel approaches were developed by the consortium to manage job submission, monitoring and retrieval of results. These are highly modular and therefore independent of the service or even to the nature of the job being submitted. The first of these is a server side job pooling mechanism, whereas the second makes use of pilot jobs and the Storage Elements. Both are explained in more detail below.

5.1. Server side job pooling

The underlying principle of the pooling mechanisms is a strict separation of the stages involved in job preparation, submission/monitoring and running. These stages roughly correspond to the web server, the grid User Interface (UI) and the Computing Element (CE) c.q. Worker Node (WN). On the server side, the end user issues a request through a web interface, providing the data required. The user input/data is subsequently validated by the CGI script at the backend of the web portal. In the server side job pooling scheme, illustrated in Figure 5-1, the data is then packaged into a self-containing job, which is dropped in a service specific directory: the job pool.
Whereas in most schemes processes in a pipeline invoke each other, the pooling mechanisms are characterized by the absence of communication between processes at the different stages. Rather than the CGI script calling a submission process, on the UI a daemon is running, scanning the job pool for packaged jobs. When these are present they are submitted to the grid (CE/WN). This can be done directly, or through a wrapper process that allows pre-processing of data and dividing a job in smaller jobs for optimized distributed computing. Jobs that are submitted are monitored from the UI and the results are retrieved after finishing. If a job fails to complete within a certain amount of time, the job will be cancelled and resubmitted.
The BCBR partner has used this approach for the HADDOCK and the csRosetta server.

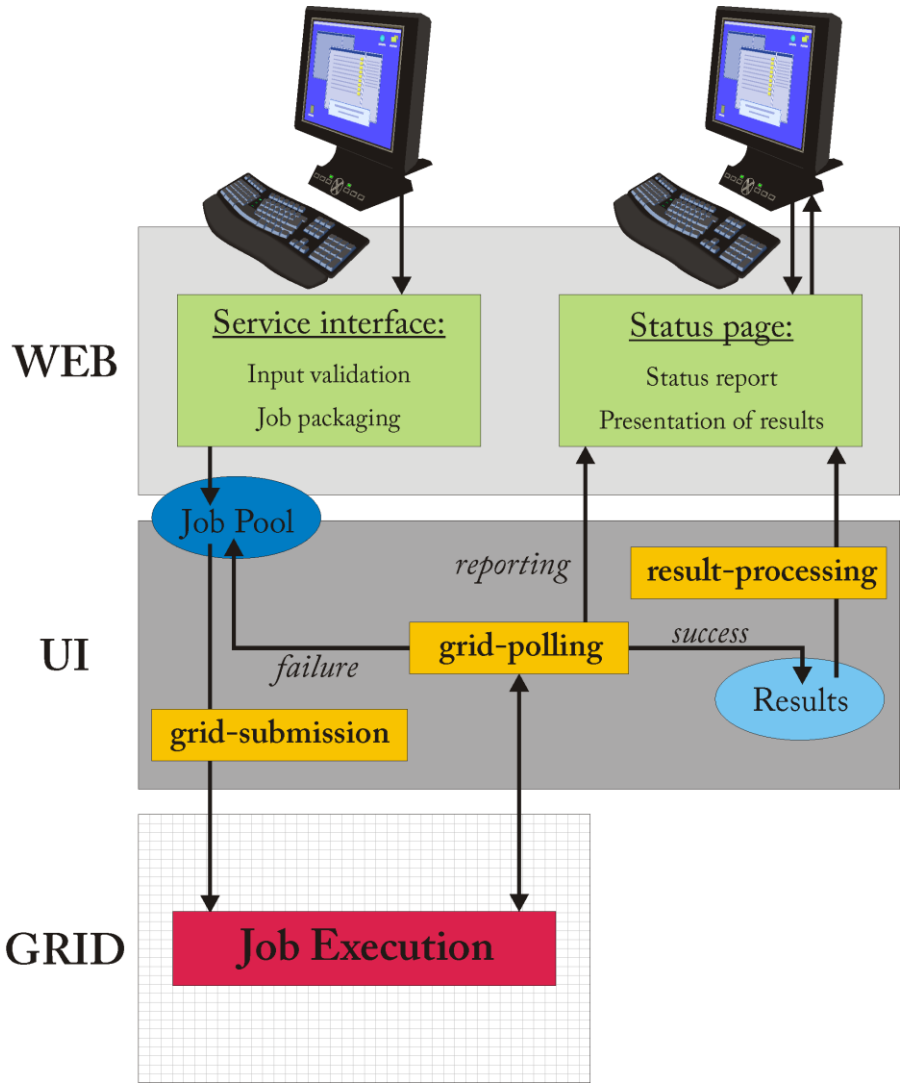

Figure 5-1: Server side job pooling scheme: green boxes indicate user interaction, whereas yellow boxes indicate jobs that are running periodically as daemon jobs and that use a robot certificate for generating a grid proxy. The blue ellipses represent 'pools', which are used for storage of job or result packages. User service requests are processed on the server, up to the point of generating a job package that is stored on disk. On the grid UI a daemon job (grid-submission) is running on a scheduled base scanning the 'job pool' for job packages and submitting these to the grid when found. Another daemon job (grid-polling) is also running periodically checking jobs that are running for their status, retrieving the results when ready and placing these in a result pool. Finally, results are presented back to the user, possibly after post-processing (results-processing).

5.2. Storage Element based pooling using pilot jobs

The server side pooling approach sketched above uses a local directory for storage of self-containing job packages and a daemon running on the UI to handle processing of the jobs. An alternative approach is to use the Storage Element (SE) for pooling of jobs and use pilot jobs running on the CE/WN for processing them, as illustrated in Figure 5-2.

Pilot jobs are generic jobs that can be submitted to the grid and, when running, request jobs from a database for execution. Several implementations of such pilot jobs have been developed, most of which use external database software like MySQL. To relieve dependency on external software, the BCBR partner has developed a pilot job mechanism that uses only the SE as the job database and standard gLite and bash commands for communication and handling. The pilot mechanism itself consists of two components: the UI pilot daemon that controls submission of pilot jobs and allows tuning the number of pilots running to the work load present, and the pilot script itself that is run on the grid. Both of these jobs are fully generic, although they can be tuned to meet the requirements of the processes to be run.
After a user has issued a request to a portal and the input has been validated, the job is prepared, packaged and transferred to a storage element. When a pilot finds a job, it locks it for processing, keeping other pilot jobs from selecting the same job. The selected job is pulled onto the worker node and executed, after which the output files are returned to the job pool on the SE. These output files can be results or can be new jobs, allowing building workflows.
The use of pilot jobs has some advantages over regular submission. The most broadly recognized advantage is that jobs run through a pilot job are less likely to stall, since only pilot jobs that are successfully submitted to the grid start pulling in jobs. Another advantage is the possibility to better tune the workload, as the number of pilot jobs may be chosen smaller than the number of jobs to be executed. A third advantage, which to our knowledge has not previously been exploited, is that a pilot job allows execution of a lot of jobs against a set of data that is common for all jobs. As an example, projects in structural biology commonly involve gathering statistics by doing calculations on a fixed set of data using random seeds for processing. This means that it is sufficient for a pilot job to copy the static data only once and then pulling in small ‘joblets’ to be executed with these data.
The BCBR partner is currently implementing this approach for RECOORD.

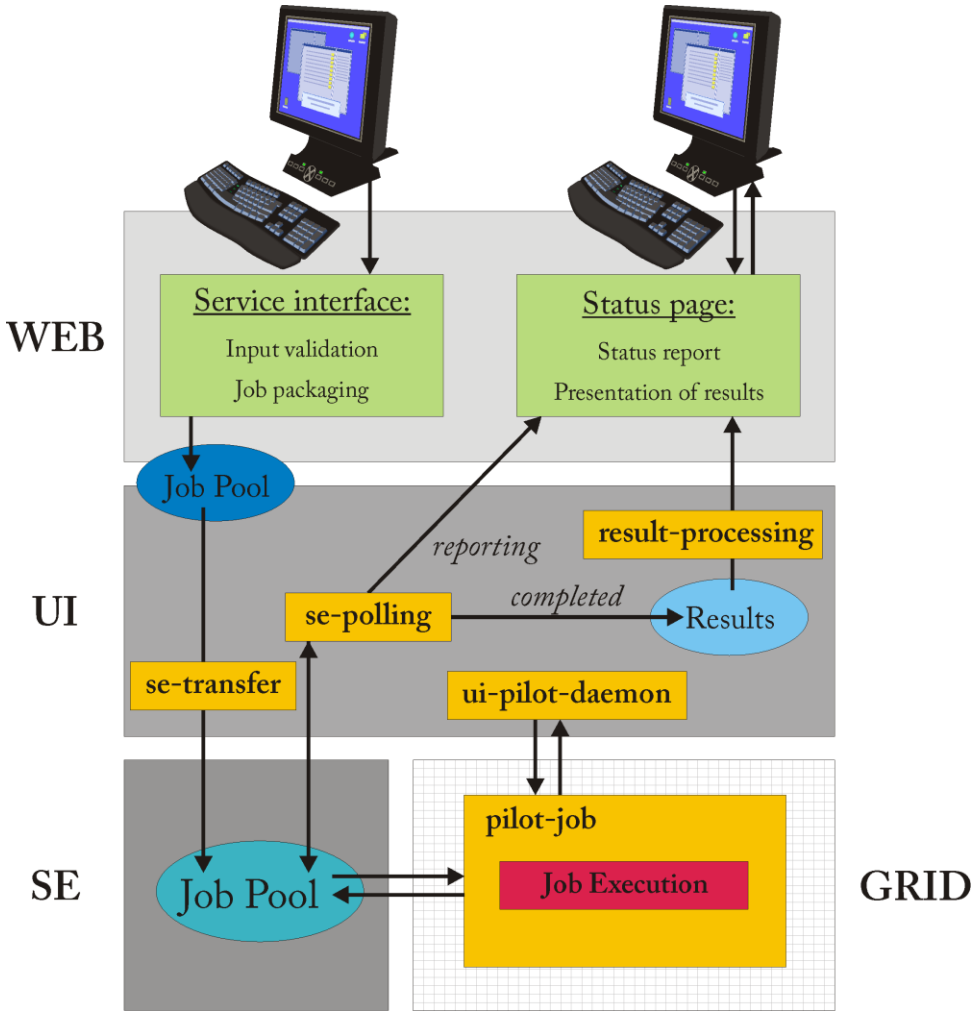

Figure 5-2: Storage element based pooling: rather than being submitted to the grid, as in server side pooling, jobs are pooled on the SE. Job execution is handled through pilot jobs that are managed through a daemon running on the grid UI.

6. Conclusions

During the reporting period software development has been mainly contained within each site, mostly so that different ways of providing grid based user services could be quickly and individually explored.
However, now that most of the main technical issues are tackled and working services are online, it is crucial that all sites start to share their expertise, knowledge and code (plus descriptions) in one central resource.
The adoption of a shared development platform and its structure has been agreed within the consortium. It aims to be the common code repository, and is also essential for the project to continue smoothly beyond the current funding period, and when core developers leave.

The software development activity concentrated in two main areas: web portal grid security; grid job automation and management.
In the first area, the work has led to a solution implemented initially in the Xplor-NIH portal operated at CIRMMP. The solution allows the user to access grid resources through his credential instead of application tied robot certificates. Following the documentation included in this report (section 4.1), the proxy delegation service has been deployed also on the BCBR testing instance of the portal hosting HADDOCK and csRosetta servers in July 2009.
In the second area, two mechanisms of job pooling have been developed to face with the huge amount of grid jobs generated mainly by HADDOCK and especially by csRosetta applications. In particular, a pilot job approach base on the use of LFC and SEs was investigated and implemented, in order to increase the effectiveness of the workload management. The software components developed for this are however of general purpose and can be applied also in different environments.