University of Kent

Integrating VOMS and PERMIS for Superior Secure Grid Management (VPman)

Deliverable 2.1
VOMS - PERMIS Integration design document

Version <1.0>

Revision History

Date	Version	Description	Author
20/July/2007	0.1	Initial draft	Bassem Nasser
15/March/2008	0.2	Correction and update	Li Su
16 March 2008	0.3	Update	Linying Su
16 Mar. 08	0.4	Minor mods	David Chadwick
28 March 2008	0.5	Finalise section 1 and 2	Linying Su
31 March 2008	0.6	Adding VOMS-PERMIS integration into OMII-UK	Linying Su
10 April 2008	1.0	Minor Modifications	Hani Ragab Hassen

Table of Contents

This document describes how VOMS and PERMIS are integrated into GT4, OMII-UK, LCAS and LCMAPS.

1. VOMS-PERMIS integration into GT4

The GT4 authorization framework allows the users to plug in a series of message interceptors, which process each message before it is being passed to the application. Two types of interceptors are of interest from an authorization perspective: Policy Information Points (PIPs) and Policy Decisions Points (PDPs).

This project intends to integrate VOMS and PERMIS into the Globus Toolkit (GT) so that the user's credentials stored in VOMS databases can be validated against PERMIS policies and then used by either a PERMIS or XACML PDP for GT4 authorisation.

A GT4-VOMS PIP allows GT4 to access and process VOMS attribute certificates. The VOMS PIP parses VOMS attributes and stores them in the GT at runtime. The PERMIS PDP can grant or deny a user request based on the obtained attributes and other information such as the environment (e.g. date and time) and the target DN (if the XACML PDP is used, then target attributes can be used for authorisation as well).

We use a simple example of the HelloWorld service to illustrate how GT4 services are protected by PERMIS. To secure a GT4 service using PERMIS, the service is configured with a set of PERMIS PIPs and a PDP. These PIPs pick up the user's attributes, the requested action along with its parameters, the environmental information and the target DN, and create the various components of an XACML request context, which is then is passed to the PDP. Each PIP or the PDP is defined as a Java module, which is associated with a scope in the format of:

<string1>:<string2>

where *<string1>* is a random identifier specifying the scope and *<string2>* is the name of the java module.

Taking the example

action:uk.ac.kent.dpa.custom.pip.ActionPIP

the random identifier "action" is the scope of the Java module uk.ac.kent.dpa.custom.pip.ActionPIP. A scope is used to to differentiate module instances. It identifies which parameters in the service configuration file (i.e. server-config.wsdd) are passed to the module. Note that a PERMIS PIP can have multiple instances being configured into GT4 (with different scope values) so that each instance can be initialised in a different way. The PERMIS PIP/PDP is configured into a GT4 service by the <authz> statement in the service security file i.e. security-config.xml, e.g.

<authz value=" *subject:uk.ac.kent.dpa.custom.pip.VomsExSubjectPIP*
resource:uk.ac.kent.dpa.custom.pip.ResourcePIP
permis:uk.ac.kent.dpa.custom.pdp.PermisPDP"/>

The following modules are provided for the VOMS-PERMIS-GT4 integration:

uk.ac.kent.dpa.custom.pip.VomsExSubjectPIP,

uk.ac.kent.dpa.custom.pip.ActionPIP,
uk.ac.kent.dpa.custom.pip.EnvPIP,
uk.ac.kent.dpa.custom.pip.ResourcePIP,
uk.ac.kent.dpa.custom.pdp.PermisPDP

As an alternative PDP, we also provide an XACML PDP module, namely *uk.ac.kent.dpa.custom.pdp.XacmlPDP*. (The way of configuring the XACML PDP can be found at http://sec.cs.kent.ac.uk/permis/integrationProjects/GT.shtml)

All these PIPs and PDPs are contained in a single jar file (i.e. JavaProjectForDPA.jar). Users have to copy this jar file and other supporting jar files (in the directory of VOMS-PERMIS-GT4/binary) into $GLOBUS_LOCATION/lib directory. For test purpose, we also provide a GT4 service and a testing client in this release package.

The release package includes three folders:

1. ***binary***: includes the PERMIS GT4 PIPs and a PDP, an XACML PDP, the PERMIS core modules, and the VOMS GT4 module.
2. ***helloWorld***: includes the HelloWorld GT4 service and a client. A sample PERMIS policy, configuration files for both PIP and PDP (init.cfg for the subject/resource PIP and the PERMIS PDP, and myparameter.txt for the action PIP), and a log property file.
3. ***doc***: includes this documentation.

1.1. PERMIS PIPs and PDP

Upon initialisation, GT4 passes configuration parameters to the PERMIS modules as indicated in the protected service's configuration file (*server-config.wsdd*), e.g:

<parameter name="permis-customConfig" value="/home/globus/pip/init.cfg"/>

<parameter name="subject-customConfig" value="/home/globus/pip/init.cfg"/>

Note that an extra "scope" tag (in the above example, the "permis" and "subject" tag) is included in the parameter's name with a dash between the scope and parameter name.

Regarding the above example, the parameter "permis-customConfig" has the value "/home/globus/pip/init.cfg", which denotes a configuration file path. This configuration is for the PERMIS PDP because the scope "permis" is associated with the module *uk.ac.kent.dpa.custom.pdp.PermisPDP* in security-config.xml. This configuration file (e.g. ***/home/globus/pip/init.cfg***. Note that a sample is provided in the release package) stores the required information for the PERMIS PDP such as: policy issuer, policy name, policy location, and user attribute certificates locations. An example of such a file is shown as follows:

```
ini: soa=cn=A Permis Test User,o=permis,c=gb
...: oid=1.2.3.4
...: ac=/home/policy/permis/mypolicyTest.ace
...: url=ldap://129.12.16.148:389/o=Grid
...: init
```

Details of writing a PERMIS PIP/PDP configuration file can be found at http://sec.cs.kent.ac.uk/permis/integrationProjects/GT.shtml

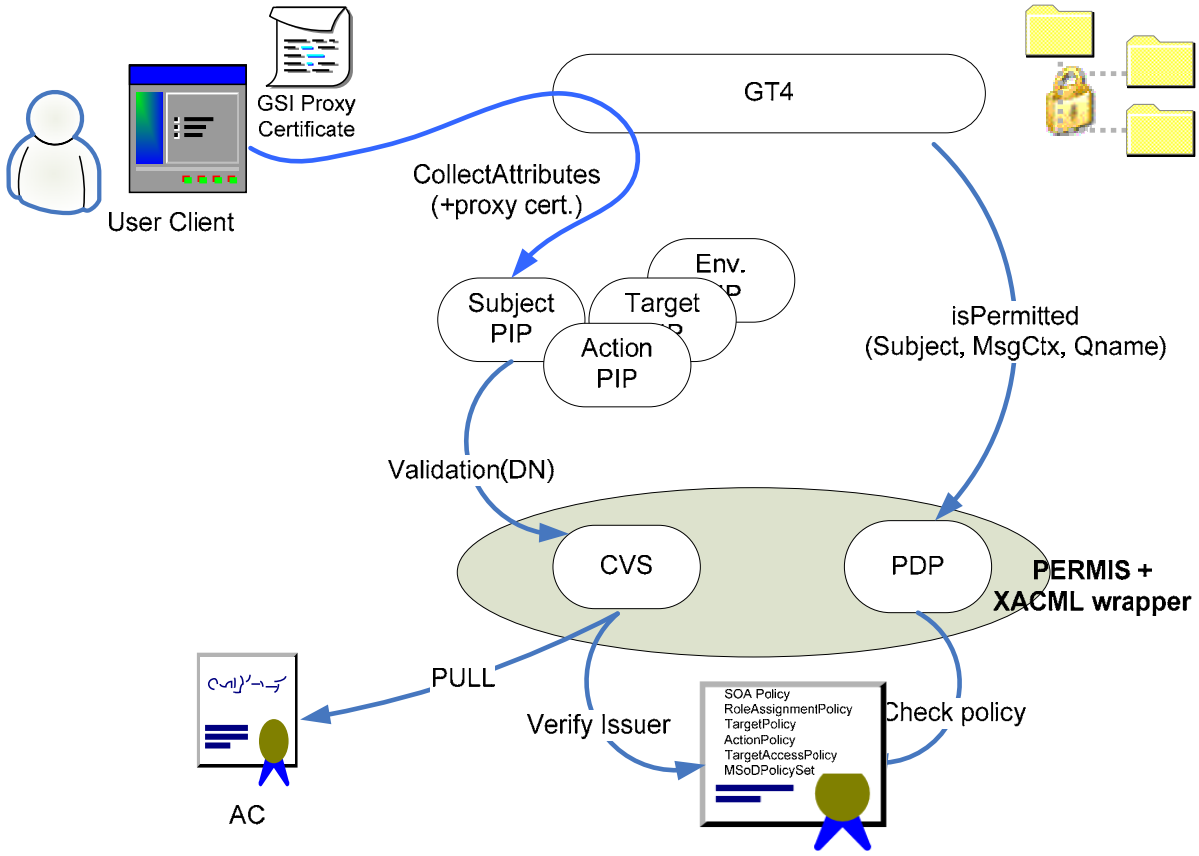


Figure 1: a schematic diagram for PERMIS GT4 integration.

As shown in Figure 1, the PERMIS PDP has an XACML wrapper interface, which enables the PERMIS PDP or XACML PDP to be used for decision making when working with GT4. One of the main functions of the various PIPs is to prepare the appropriate component of the XACML request context, and make it ready for the PDP to make an access control decision.

Each PIP fills in certain data in the request context. The output of the subject PIP is an XACML request context including the subject ID and its valid attributes. The subject PIP defined as uk.ac.kent.dpa.custom.pip.VomsExSubjectPIP is capable of working in both PUSH and PULL modes to get the user's attributes. From the perspective of VOMS-PERMIS integration, the user can push a VOMS proxy certificate and/or pull his attributes from a VOMS service (using the VOMS SAML web service). Then, this subject PIP can validate those attributes and pass them to the PERMIS (or XACML) PDP.

```
<Request xmlns="urn:oasis:names:tc:xacml:1.0:context" xmlns:permis="http://
issrg.cs.kent.ac.uk" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<Subject>
<Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
DataType="http://www.w3.org/2001/XMLSchema#string">
<AttributeValue>
CN=bassem,OU=issrg,O=kent,C=GB
</AttributeValue>
</Attribute>
<Attribute AttributeId="permis:permisRole" DataType="http://www.w3.org/2001/
XMLSchema#string">
<AttributeValue>
researcher
</AttributeValue>
</Attribute>
</Subject>
<Action/>
<Resource/>
<Environment/>
<Request/>
```

Figure 2 An example of request context when multiple PIPs are used.

When other PIPs (Action, Resource and Environment) are called, each one fills in its own part of the request context (these fields are left empty in Figure 2).

Figure 3 shows the details of the authorization procedure. Note, in the push or pull mode, the system pushes or pulls Attribute Certificates (or SAML assertions, which are not shown in Figure 1).

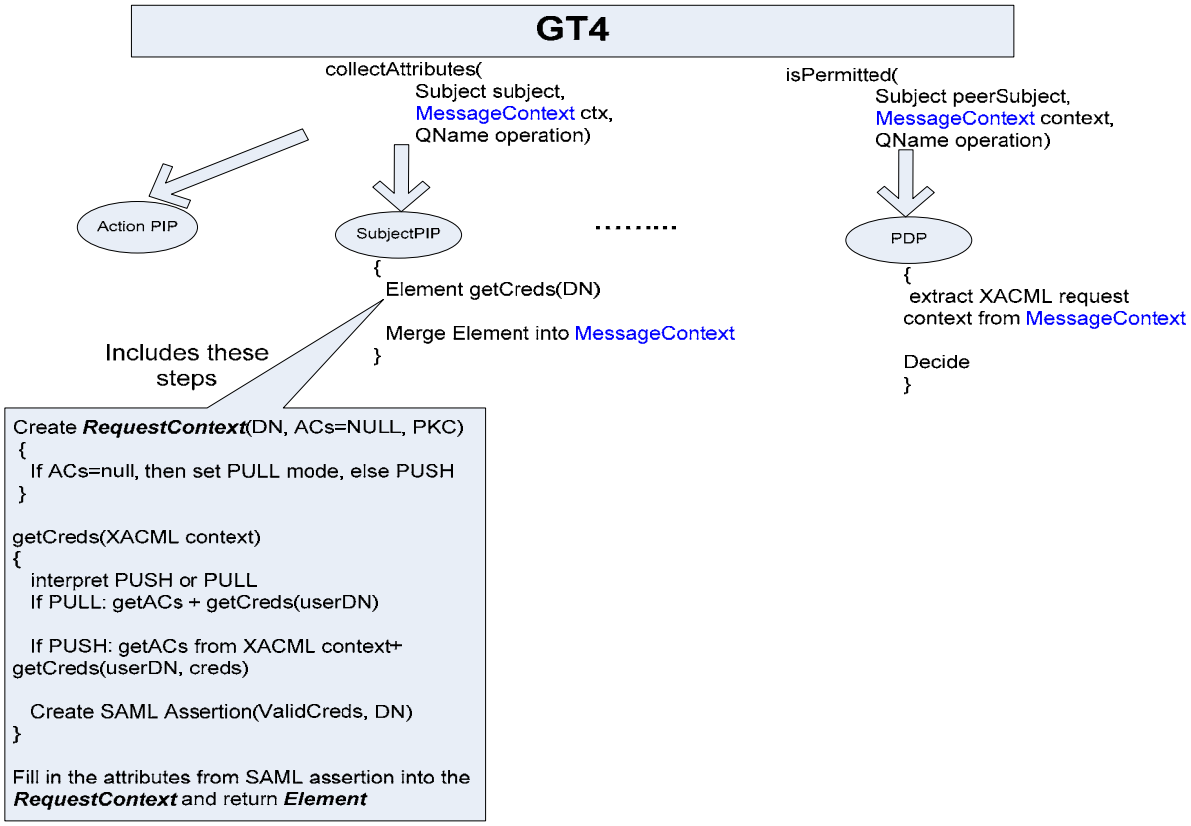


Figure 3: Authorization procedure in both push and pull modes

1.2. Introducing VOMS PIP

When installing the Globus VOMS modules to a particular Globus installation and incorperating org.globus.voms.PIP into the authorisation framework, VomsExSubjectPIP is capable of handling the pushed VOMS attribute certificates. Note that users should not use VOMS plug-ins such as globus-voms-0_2.jar and globus-voms-0_2-gt4.0.jar, which are supplied by Globus (http://dev.globus.org/wiki/VOMS). Instead, our modified version should be used (it is supplied in this release as VOMS-PERMIS-GT4/binary/VOMSplugins.jar). It should be copied to $GLOBUS_LOCATION$/lib, and overwrite the old version. Without installing the Globus VOMS modules, VomsExSubjectPIP can still work as a pure PERMIS subject PIP.

In order to add VOMS PIP into the system, users have to:

- List the VOMS modules in the authorization element of the relevant security descriptor. In this case, the PERMIS subject PIP (i.e. *uk.ac.kent.dpa.custom.pip.VomsExSubjectPIP*) is mandatory and other PIPs are optional. E.g. in the ***security-config.xml***, we may have:

```
<authz value="vomsPIP:org.globus.voms.PIP
subject:uk.ac.kent.dpa.custom.pip.VomsExSubjectPIP
action:uk.ac.kent.dpa.custom.pip.ActionPIP
resource:uk.ac.kent.dpa.custom.pip.ResourcePIP
permis:uk.ac.kent.dpa.custom.pdp.PermisPDP"/>
```

- Include the VOMS module configurations, which provide necessary per-module information. These parameters are passed by GT4 (***server-config.wsdd***) during initialisation:

  - *vomsTrustStore* : indicates to the PIP which public key certificates are trustwhorthy when validating the signatures on the VOMS ACs. It also defines a directory where the trusted VOMS servers PKCs are stored, e.g.

```
<parameter name="vomsPIP-vomsTrustStore"
value="/etc/gridSecurity/vomsdir/*"/>
```

  - *vomsValidate*: If this variable is set to *true*, it triggers path validation on the attribute certificate, e.g.

```
<parameter name="vomsPIP-vomsValidate " value="true">
```

Figure 4 illustrates how the functionality of the PERMIS Subject PIP is modified to use VOMS credentials.

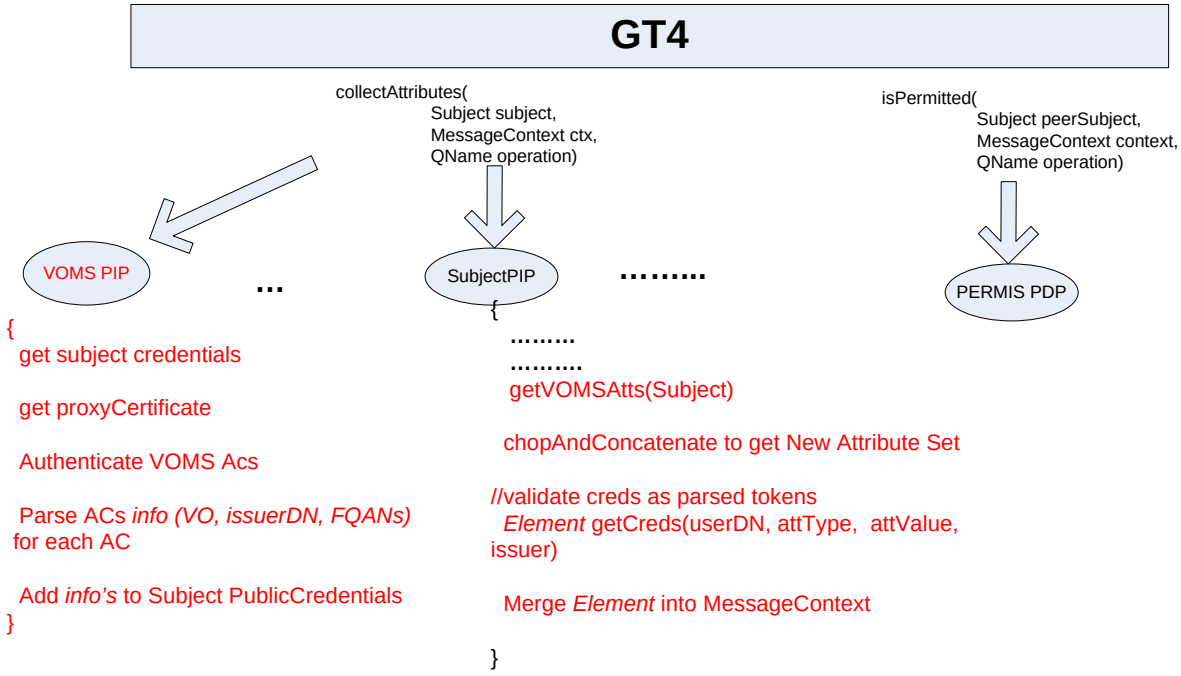


Figure 4: a modified version of PERMIS Subject PIP which uses VOMS credentials.

Note: Subject PIP *(Element* ***getCreds(userDN, attType, attValue, issuer))*** checks whether a VOMS server (passed as issuer) is authorized by the policy to issue such attributes (attType, attValue) to a particular user. This is the AC validation step.

2. HelloWorld Service deployment and security

2.1. Introduction

In order to test VOMS-PERMIS-GT4 integration, we implement a sample GLOBUS GT4's HelloWorld service and a client secured by PERMIS (tested on GT4.0.0 and GT4.0.4). The client submits a proxy certificate including VOMS credentials. Then, PERMIS is used to decide whether the request should be granted or denied according to the user's attributes, e.g. vo_name, vo_group or vo_role, which are validated against the specified policy.

Users should have a PKI such as openssl or keytool to issue and sign certificates. Globus requires a container certificate, host certificate, service certificate and user certificate to be specified. Usually, a container certificate and a host certificate should be automatically created after Globus installation and configuration. Only a service and a user certificate should be added by the user.

A sample policy (specified in testUserpolicy.ace and testUserpolicy.xml) can be found in this release. Note that one of the SOAs in the policy, which issues the VOMS proxy certificate, must be the VOMS server, which is identified by its host certificate. In order to write PERMIS policy, users may need the PERMIS policy editor (download from http://sec.cs.kent.ac.uk/permis/downloads/Level1/policy.shtml). If they want to use a policy attribute certificate, then they may need the PERMIS ACM tool (download from http://sec.cs.kent.ac.uk/permis/downloads/Level1/acm.shtml).

2.2. VOMS server and credentials

In order to configure the VOMS server, you may need to:

- Add a user to a VO
- Assign an attribute to the user within the VO

These tasks can be done using a VOMS web application. Alternatively, they can also be done using VOMS commands.

- Get a proxy certificate including VOMS credentials

This should be done using the following command: e.g.

voms-proxy-init –cert userCert.pem –key user.key –debug - voms nanocmos:all

where *nanocmos* is a VO name. The VO's configurations file (i.e. vomses) should be placed at ~/.glite or $VOMS_LOCATION/etc directory. Users can also point to this file by using the –userconf option in the above command.

Note that, in PERMIS, if a user has a FQAN of *"/VOname/VOgroup/VOrole*", , he/she is considered to have the following attributes:

vo_name = /VOname
vo_group = /VOname/VOgroup
vo_role = /VOname/VOgroup/VOrole

These attributes can be used to write a PERMIS policy (see section 4).

2.3. HelloWorld service

This is a simple service with only two methods:

- *void setHello(String name):* a method with a single parameter. It sets the name to the value of the parameter.
- *String getValueRP():* a method that returns "Hello" followed by the name set by the previous method.

To deploy the HelloWorld service, users can simply copy the directory *VOMS-PERMIS-GT4/helloWorld/service/org*_globus_services_core_HelloWorldService into *$GLOBUS_LOCATION$/etc*; *VOMS-PERMIS-GT4/helloWorld/service/HelloWorldService* into *$GLOBUS_LOCATION$/share/schema*; and all files in *VOMS-PERMIS-GT4/helloWorld/service/binary* into *$GLOBUS_LOCATION$/lib*. Note that users have to replace the service certificate specification in the service's security configuration (i.e. *security-config.xml*) with their own specification. The service certificate (i.e. PKC and private key certificate) can be placed at anywhere. However, the signing PKC (must be self-signed) of the service certificate must be placed at */etc/grid-security/certificates*

Users also need to copy the files `init.cfg` and my`parameter.txt` to their local file system and modify the file `server-config.wsdd` to include paths to these configuration files.

The files "`init.cfg`" and "my`parameter.txt`" contain configuration information for PERMIS PIPs and PDP (see Appendix A). You should modify the init.cfg file to indicate the path to your PERMIS policy (see section 4). The myparameter.txt is used for an action PIP, which specifies what action and parameter will be picked up when the user submits a request. Details of writing the action PIP configuration can be found in GT4 documents or from http://sec.cs.kent.ac.uk/permis/integrationProjects/GT.shtml

The VOMS PIP needs to be initialised by those two parameters: *vomsPIP-vomsTrustStore* and *vomsPIP-vomsValidate.* The former indicates the path to the *vomsTrustStore.* This store contains the trusted VOMS servers. Please make sure that a copy of your VOMS server host certificate *hostcert.pem* is in that folder. The latter triggers the validation of the VOMS certificate chain. An example of the VOMS PIP configuration is shown as follows:

```
<parameter name="vomsPIP-vomsTrustStore" value="/etc/grid-
security/vomsdir/*"/>
<parameter name="vomsPIP-vomsValidate" value="false"/>
```

2.4. PERMIS policy

PERMIS policy should include the new attribute types vo_name, vo_group and vo_role. Each of them contains a specific OID. An example for defining an attribute "/nanocmos" of type "vo_name" and another vo_role, "/nanocmos/Role=modeller" is shown as follows:

```
<RoleSpec OID="1.2.826.0.1.3344810.1.1.109" Type="vo_name">
 <SupRole Value="/nanocmos" />
 </RoleSpec>

<RoleSpec OID="1.2.826.0.1.3344810.1.1.110 " Type="vo_group">
```

```
    <SupRole Value=" /nanocmos/glasgow" />
  </RoleSpec>

  <RoleSpec OID="1.2.826.0.1.3344810.1.1.111" Type="vo_role">
    <SupRole Value=" nanocmos/Role=modeller" />
  </RoleSpec>
```

The VOMS server should be one of the SOAs (Source of Authority)

```
<SOASpec ID="SOA2"
LDAPDN="CN=dhcp10b1.kent.ac.uk,OU=ISSRG,O=University of Kent,C=GB" />
```

The above policy indicates the attributes that a VOMS server may issue for that user.
The policy can be set to either fine or coarse granularity level. For example, it may indicate that:

- Users that have the attribute “vo_name=/nanocmos” are authorised to invoke method setHello.
- The policy may also indicate that users with attribute vo_role= /nanocmos/Role=modeller can be authorised to invoke method setHello.
- It is also possible for group users that have attribute vo_group = /nanocmos/glasgow to be authorised to invoke method getValueRP

You can find an example policy in the release package.

2.5. Globus container

To run the globus container, use the following command:

```
globus-start-container -debug
```

Then, you should see

```
VOMS PIP is being initialised
initialising VomsCredentialPIP
```

followed by a list of services including the HelloWorld service.

To enable logging, copy the supplied “container-log4j.properties” and replace the old one in GLOBUS_LOCATION.

2.6. Hello service Client

- Setup the environment variables X509_VOMS_DIR to indicate the path to the VOMS server certificate (e.g. /etc/grid-security/vomsdir), and TRUSTED_CA to indicate the path of trusted CAs (e.g. /etc/grid-security/certificates).

- Make sure that you start the VOMS server from a different account as you run a VOMS client if using the same machine.

- Create a proxy certificate using the client *voms-proxy-init*, e.g.

  voms-proxy-init -cert usercert.pem -key userkey.pem -voms vpman:all

As an example, the following paragraph presents a vomses file, which should be placed at either ~/.glite or $VOMS_LOCATION/etc. This configuration file specifies the VOs contained by the VOMS server. Each line specifies a VO, which includes five parameters to indicate the local VO name, the server host name, the server port, the server Distinguished Name (DN) and the VO name on the server. Note that the common name of the server's DN should be the same as the server's host name.

"eso" "dhcp10b1.kent.ac.uk" "15000" "/C=GB/O=University of Kent/OU=ISSRG/CN=dhcp10b1.kent.ac.uk" "eso"
"vpman" "dhcp10b1.kent.ac.uk" "15001" "/C=GB/O=University of Kent/OU=ISSRG/CN=dhcp1081.kent.ac.uk" "vpman"
"votesdiabetes" "dhcp1081.kent.ac.uk" "15002" "/C=GB/O=kent/OU=issrg/CN=dhcp1081.kent.ac.uk" "votesdiabetes"]

The created proxy certificate is saved as a text file e.g. /tmp/x509up_u500. The file name may be different from the above, depending on account the user logs in.

- Copy VOMS-PERMIS-GT4/helloWorld/client into usre's local directory

User's security configuration (i.e. security-config.xml in VOMS-PERMIS-GT4/helloWorld/client) has a reference to the file that contains the proxy certificate issued by VOMS server (in fact, it contains a certificate and a key) e.g.:

```
<securityConfig xmlns="http://www.globus.org">
     <credential>
          <key-file value="/tmp/x509up_u500"/>
          <cert-file value="/tmp/x509up_u500"/>
     </credential>
     <authz value="none"/>
</securityConfig>
```

- Navigate to the folder and run the client using the following command, e.g..

```
java -jar secClientAlone.jar
https://129.12.16.133:8443/wsrf/services/org/globus/services/co
re/HelloWorldService bassem ./security-config.xml
```

The client will try to invoke the service at the specified address, which may be different from the above example. It will call the method setHello("bassem"), and then call the method getValueRP.

If the client is authorized to invoke both methods, the user will get the following output:

```
[main] DEBUG Client helloworld  - Client initialized with
user's security descriptor = ./security-config.xml
[main] DEBUG Client helloworld  - Endpointreference created
[main] DEBUG Client helloworld  - Endpointreference setaddress
[main] DEBUG Client helloworld  - portType created
[main] DEBUG Client helloworld  - security properties set, OK
[main] DEBUG Client helloworld  - trying to invoke setHello
```

```
[main] DEBUG Client helloworld  - setHello called, OK
Helloo Hello bassem!
[main] DEBUG Client helloworld  - getValueRP output: Helloo
Hello bassem!
```

On the GT4 server side, All of the VOMS attributes contained in the proxy certificate are displayed. E.g.

In PIP /vpman/Role=NULL/Capability=NULL
In PIP /vpman/Role=VO-Admin/Capability=NULL
In PIP /vpman/Role=Engineer/Capability=NULL
In PIP /vpman/kent/Role=NULL/Capability=NULL
In PIP /vpman/kent/issrg/Role=NULL/Capability=NULL

Users can also check the log file "day.log", which presents the detailed output from each PIP and the PDP. The log file will be generated in the same folder as you run "globus-start-container –debug".

If the client is denied to invoke the methods, the user will get some Java exceptions.

3. VOMS-PERMIS integration into OMII-UK

PERMIS has a SAML authorisation service, which can be run as a stand alone server that will accept incoming SAML Authorisation Decision Requests and will respond with SAML Authorisation Decision Responses. OMII-UK has the SAML callout authorisation mechanism; therefore it will be able to call the PERMIS SAML authorisation service. Once the PERMIS is configured with the VOMS SAML repository, which contacts a VOMS SAML web service, OMII services can be protected by PERMIS using VOMS attributes.

Alternatively, PERMIS can also implement the OMII_UK's authorization interface, (http://www.omii.ac.uk/downloads/project.jsp?projectid=68) so that it can be used to protect any OMII data resource and activity on the resource.

http://www.omii.ac.uk/downloads/project.jsp?projectid=68

4. VOMS-PERMIS integration with LCAS

4.1. LCAS functioning and configurations

The authorization decision of LCAS is based on the user's certificate and the job specification in the RSL (JDL) format. When making a decision, a certificate and a RSL are sent to (plugin) authorization modules, which then grant or deny the access to the fabric.

The LCAS obtains its configuration, including the plugins, from the file *lcas.db*. An example of this file is shown as follows. The arguments required by the plugin are specified as *pluginargs="<arguments>"*.

```
# LCAS database/plugin list
#
# Format of each line:
# pluginname="<name/path of plugin>", pluginargs="<arguments>"
#
#
pluginname="lcas_userallow.mod",pluginargs="allowed_users.db"
pluginname="lcas_userban.mod",pluginargs="ban_users.db"
pluginname="lcas_timeslots.mod",pluginargs="timeslots.db"
pluginname="lcas_plugin_example.mod",pluginargs="Some bogus arguments"
pluginname="lcas_voms.mod",pluginargs="-vomsdir /etc/grid-security -
certdir /etc/grid-security/certificates/ -authfile /etc/grid-
security/lcasvoms-mapfile -authformat simple
```

Three standard authorization modules are provided by default. Each has its own configuration as specified in those files as below:

- allowed users.db: It contains a list of LDAP distinguished names (DN) of users that are allowed on the fabric.
- ban users.db: It contains a list of DNs of users that should be banned from the fabric.
- timeslots.db: It contains the 'opening hours' of the fabric.

In addition to these standard authorization plugins, new plugins may be written. The plugins have to be provided as shared objects. When LCAS receives an authorization request,

it uses *dlopen* to open the plugin shared object. The interface between the plugins and the LCAS has three forms, which are used to call a *dlsym* by LCAS. In the next section, we will explain each in turn.

4.2. LCAS plugins

4.2.1. The API used by LCAS plugins

The LCAS authorization plugins/modules, should "include" this file lcas_modules.h to use the offered utilities API via these header files (included in lcas_modules.h):

lcas_defines.h: *Public header file with common definitions for the LCAS (authorization modules):*
lcas_log.h : *Logging API for the LCAS plugins and LCAS itself.*
lcas_modules.h : *The LCAS authorization plugins/modules should "include" this file*
lcas_types.h : *Public header file with typedefs for LCAS.*
lcas_utils.h : *API for the utilities for the LCAS.*

The header file lcas_vo_data.h (not included in lcas_modules.h) should be included by the plugins that manipulate VOMS attributes. It enables creating and accessing VO data structures.

4.2.2. LCAS plugins interface:

The interface that the between the plugin and LCAS contains the following methods:

1. *int plugin_initialize(int argc, char ** argv)*

This initializes the plugin with arguments argc and argv. *argc* denotes the number of arguments and *argv* is a list of arguments. By default, argv[0] contains the name of the plugin.

The above methods may return the following values:
LCAS_MOD_SUCCESS indicates a successful initialization
LCAS_MOD_FAIL indicates a failure in the plugin initialization
LCAS_MOD_NOFILE indicates a private plugin database could not be found (resulting in the same effect as LCAS_MOD_FAIL)

2. *int plugin_terminate()*

Question: when is the terminate called? Is it called after each call to the module? Then if LCAS calls the module then LCMAPS, will each one call initialize and terminate? Are they called on each request or upon lcas/lcmaps initialisation??

Whatever is needed to terminate the plugin module goes in here.

LCAS_MOD_SUCCESS indicates a successful execution
LCAS_MOD_FAIL indicates an unsuccessful execution (which results in an authorization failure)

3. *int plugin_confirm_authorization(lcas_request_t request, lcas_cred_id_t lcas_cred)*

This method requests for authorization by providing the RSL (or JDL) and the user credential. The user credential contains information describing the role of the user. RSL (JDL) specifies the resources that a user intends to use. Then, an authorization decision can be made using this information.

The possible return values of this method are:
LCAS_MOD_SUCCESS indicates that the authorization was successful
LCAS_MOD_FAIL indicates that the authorization has failed
LCAS_MOD_NOFILE indicates a private plugin database could not be found (this results in LCAS authorization being denied)

4.2.3. LCAS VOMS plugin

This plugin acts as an interface between the VOMS data created in the user grid credential (X509 certificate) and the LCAS system. It retrieves the VOMS data using the VOMS API. Then, the VOMS data can be checked against either a (simple) gridmap style file, a GACL-file or an XACML-file in order for the user's request to be authorized.

LCAS may load this module with the following parameters:

lcas_voms.mod -vomsdir <vomsdir> -certdir <certdir> -authfile <authorization file> [-authformat <format of the authorization file>]

- *vomsdir <vomsdir>*: This is the directory that contains the certificates of the VOMS servers
- *certdir <certdir>*: This is the directory that contains the CA certificates
- *authfile <authorization file>* : This file specifies the authorization/access control based on VOMS information. Its format can be 'simple' (gridmap style), 'gacl' or 'xacml', which is specified explicitly with the option –authformat (see the next parameter).
- *authformat <format of the authorization file>* : it sets the format of the authorization file. It can take values of: `gacl/GACL`, `xacml/XACML` or `simple`.
- *gacl_use_voms_dn [yes|no|always]*: This option specifies whether the voms DN, which is found in the user certificate, should be included in the user gacl credential. It can have the following values:

- yes : For each VO-GROUP-ROLE combination found in the user certificate, two gacl credentials are created. (One *with* and one *without* the voms DN.) In this way, the user is authorized if no voms DN is included in the gacl authorization file. The default value is 'yes'.
- always : For each VO-GROUP-ROLE combination found in the user certificate, only one gacl credential can be created *with* a voms DN.
- no : For each VO-GROUP-ROLE combination found in the user certificate, a gacl credential can be created *without* a voms DN.

- *use_user_dn:* If this option is set, user proxies will be processed without voms information. If the user DN of the proxy is present in the gacl or gridmapfile, the user can be authorized by this plugin.

The return value can be following:

- LCAS_MOD_SUCCESS : indecates a successful execution
- LCAS_MOD_FAIL : indecates a failure

5. VOMS-PERMIS integration with LCMAPS

5.1. LCMAPS plugins interface

The interface between the plugin and the LCMAPS contains the following methods:

1. int plugin_initialize (int argc, char ** argv)

This method initializes the plugin with the arguments as read from the LCMAPS database (argc, argv).

Its possible return values are:
LCMAPS_MOD_SUCCESS indicates a successful initialization
LCMAPS_MOD_FAIL indicates a failure in the plugin initialization
LCMAPS_MOD_NOFILE indicates that a private plugin database could not be found (same effect as LCMAPS_MOD_FAIL)

2. int plugin_introspect(int * *argc*, lcmaps_argument_t ** *argv*)

Plugin asks for required arguments [is this unfinished?]

Its possible return values are:
LCMAPS_MOD_SUCCESS indicates that the process is successful
LCMAPS_MOD_FAIL indicates the failure of the method call (which results in a lcmaps failure)

3. int plugin_run(int *argc*, lcmaps_argument_t * *argv*)

This method gathers credentials for users from its arguments (like JDL, globus DN, VOMS roles etc.), which were ordered by the plugin_introspect() function in advance.

The possible return values are:
LCMAPS_MOD_SUCCESS indicates that the authorization is successful
LCMAPS_MOD_FAIL indicates that the authorization has failed

4. int plugin_terminate()

This methods terminates the plugin module.

The possible return values are:
LCMAPS_MOD_SUCCESS indicates a successful termination

LCMAPS_MOD_FAIL indicates a termination failure (which results in an authorization failure)

5.2. The API to be used by the LCMAPS plugins

lcmaps_modules.h should be included in the LCMAPS authorization plugins/modules . The API available for including the header contains:

lcmaps_arguments.h *Public header file to be used by plugins.*
lcmaps_cred_data.h *Public header file to be used by plugins.*
lcmaps_defines.h *Public header file with common definitions for the LCMAPS (authorization modules).*
lcmaps_log.h *Logging API for the LCMAPS plugins and LCMAPS itself.*
lcmaps_modules.h *The LCMAPS authorization plugins/modules should "include" this file*
lcmaps_types.h *Public header file with typedefs for LCMAPS.*
lcmaps_utils.h *API for the utilities for the LCMAPS.*
lcmaps_vo_data.h *LCMAPS module for creating and accessing VO data structures*

5.3. LCMAPS configuration

The LCMAPS reads its configuration, in particular the plugins and the local site policy from the file lcmaps.db. An example of such file is shown as follows:

```
# LCMAPS policy file/plugin definition
#
# default path
path = /opt/edg/lib/lcmaps/modules

# Plugin definitions:
example           = "lcmaps_plugin_example.mod"
                        "Some bogus arguments"
localaccount      = "lcmaps_localaccount.mod"
                        "-gridmapfile /etc/grid-security/grid-mapfile"
poolaccount       = "lcmaps_poolaccount.mod"
                        "-gridmapfile /etc/grid-security/grid-mapfile"
                        "-gridmapdir /etc/grid-security/gridmapdir"
                        "-override_inconsistency"
posix_enf         = "lcmaps_posix_enf.mod"
                        "-maxuid 1"
                        "-maxpgid 1"
                        "-maxsgid 32"
vomsextract       = "lcmaps_voms.mod"
                        "-vomsdir /etc/grid-security/vomsdir"
                        "-certdir /etc/grid-security/certificates"
vomslocalgroup    = "lcmaps_voms_localgroup.mod"
                        "-groupmapfile /etc/grid-security/groupmapfile"
                        "-mapmin 0"
vomspoolgroup     = "lcmaps_voms_poolgroup.mod"
                        "-groupmapfile /etc/grid-security/groupmapfile"
                        "-groupmapdir /etc/grid-security/groupmapdir"
                        "-override_inconsistency"
                        "-mapmin 0"
vomspoolaccount   = "lcmaps_voms_poolaccount.mod"
                        "-gridmapfile /etc/grid-security/grid-mapfile"
                        "-gridmapdir /etc/grid-security/gridmapdir"
ldap_enf          = "lcmaps_ldap_enf.mod"
                        "-maxuid  1"
                        "-maxpgid 1"
                        "-maxsgid 32"
```

```
                          "-hostname ldap.example.org"
                          "-port 389"
                          "-require_all_groups yes"
                          "-dn_manager \"cn=Manager,dc=root\""
                          "-ldap_pw /opt/edg/etc/lcmaps/test_pw"
                          "-sb_groups \"ou=LocalGroups,dc=foobar,dc=ough\""
                          "-sb_user \"ou=LocalUsers,dc=foobar,dc=ough\""
                          "-timeout 5"

# Policy:
# Rule1:
voms:

# if localaccount succeeds, enforce with posix_enf, else vomsextract
localaccount -> posix_enf | vomsextract
vomsextract -> vomspoolgroup
vomspoolgroup -> vomspoolaccount
vomspoolaccount -> ldap_enf
ldap_enf -> posix_enf

# Rule2:
standard:
localaccount -> posix_enf | poolaccount
poolaccount -> posix_enf
```

The default path to the LCMAPS plugins is specified in the line starting with “path =”. In the following lines, aliases are defined for the complete plugin names and their options.
The policy rules define a sequence of modules to evaluate the rule. The first rule concerns the user’s VOMS credentials verification. The localaccount module has to be invoked first. When it has been successfully invoked, the posix_enf is called for enforcement, otherise vomsextract should be called. Successful invocations chain the modules vomspoolgroup, vomspoolaccount, ldap_enf and posix_enf for enforcement.

5.4. Existing Acquisition plugins

5.4.1. lcmaps_localaccount.mod plugin

This plugin is an *Acquisition Plugin* that provides the LCMAPS system with Local Account credential information. To do this, it lookd up the Distinghuished Name (DN) from a user's certificate in the grid-mapfile. If this DN is found in the grid-mapfile, the plugin knows the mapped local (system) account username. Using the username of the local account, the plugin can then gather additional information about this account.

The plugin resolves the UID, GID and all the secondary GIDs. When finishes and the plugin has been detected, it will add this information to the Plugin Manager. The plugin returns a LCMAPS_MOD_SUCCESS. This result will be sent to the Plugin Manager, which invokes the plugin. Then, it will forward the result to the Evaluation Manager, which takes appropriate actions for the next plugin. Normally, this plugin is followed by an enforcement plugin that applies these gathered credentials as required by a system administrator.

5.4.2. lcmaps_voms.mod

This plugin is an interface between the VOMS data in a certificate and the LCMAPS system. It acquires VOMS data via the VOMS API. The API specifies a retrieve function that builds a voms data structure in the plug-in. The retrieve function requires a (chain of) OpenSSL x.509 certificate(s). The plug-in will transfer the required VOMS data to the Plugin Manager, where this 'raw' credential data is stored and made reachable like the other known

data (e.g. gathered uid(s), Primary GID(s) and Secondary GIDs). Using this plugin other voms-aware plugins can transparently use the VOMS data without explicit extract such data.

5.4.3. lcmaps_voms_localgroup.mod plugin

The localgroup acquisition plugin is a voms-aware plugin. The plugin's main purpose is to gather credential information from a given Voms Acquisition plugin. This plugin gathers primary and secondary GIDs.

, All VO-GROUP-ROLE(-CAPABILITY) values are stored in the credential data structure in the Plugin Manager. This plugin obtains this data and lookups in a “groupmapfile” for all VO-GROUP-ROLE values. Wildcards can be used in the groupmapfile to match multiple VO-GROUP-ROLE combinations.

An example groupmapfile:
```
/VO=atlas/GROUP=mcprod atmcprod
/VO=atlas/GROUP=* atlasgrps
```

/VO=atlas/GROUP=mcprod as a VO-GROUP combination from the gained credential data can be matched with /VO=atlas/GROUP=mcprod, and mapped to the GID of the 'atmcprod' group. All the other groups within the 'atlas' VO will be mapped to 'atlasgrps'. If there is a user with /VO=cms, that user cannot be mapped to any local system group unless there is an extra information in the groupmapfile such as '/VO=* allothers', which maps from any other VO-GROUP-ROLE combinations to 'allothers'.

For every value in the Plugin Manager there will be a search in the groupmapfile. [I don’t understand it.] The first extracted and gathered VO-GROUP-ROLE combination is set to the primary group.
This plugin may be configured to map all voms data entries to (system) groups in order to find their GID, or just map a subset of entries. If the number of entries in that subset is set to zero or unspecified, the plugin returns a success even if no local groups were found.

5.4.4. lcmaps_voms_poolaccount.mod plugin

lcmaps_poolaccount.mod [-gridmapfile|-GRIDMAPFILE|-gridmap|-GRIDMAP <location grid-mapfile>] [-gridmapdir|-GRIDMAPDIR <location gridmapdir>] [-do_not_use_secondary_gids] [-do_not_require_primary_gid]

The poolaccount acquisition plugin is a voms-'aware' plugin. Its main purpose is to gather credential information from a given VOMS acquisition plugin. This plugin then gathers a UID. The values of VO-GROUP-ROLE(-CAPABILITY) are stored in the credential in the Plugin Manager. This plugin can obtain this data and compare it with the first known VO-GROUP-ROLE combination that has been extracted from the certificate. Entries in the certificate are the same 'grid-mapfile' as the localaccount and poolaccount plugin. In that file, VO-GROUP-ROLE combinations are stored with each entry mapping to a poolaccount, for example:
```
"/VO=wilma/GROUP=*" .test
"/VO=fred/GROUP=*" .test
```

When a user submits its certificate and the first known VO is 'wilma', the plugin s reads poolaccount from the '.test' pool. This returns 'test001' as a poolaccount for this user. A

linking between '/VO=wilma/GROUP=*', this user and a poolaccount must be made. [I don't understand it.]

NOTE: This plugin will only be successful after running both localgroup and poolgroup.

5.4.5. voms poolgroup plugin

lcmaps_voms_poolgroup.mod -GROUPMAPFILE|-groupmapfile|-GROUPMAP|-groupmap <groupmapfile> [-mapall] -GROUPMAPDIR|-groupmapdir <groupmapdir>

The poolgroup acquisition plugin is a voms-'aware' plugin. Its main purpose is to gather credential information from a given Voms acquisition plugin. This plugin gathers both primary and secondary GIDs. Then, it compares all the VO-GROUP-ROLE values with the entries in the file 'groupmapfile'. The plugin lookups each value (a VO-GROUP-ROLE combination) and searches in the groupmapfile for a match. Wildcards can be used in the groupmapfile to match multiple VO-GROUP-ROLE combinations.

Examples of 'groupmapfile' are shown as follows:

```
/VO=atlas/GROUP=mcprod mcprod
/VO=atlas/GROUP=mcprod .atlas
/VO=atlas/GROUP=dev .atlas
/VO=atlas/GROUP=* .atlas
```

Note, /VO=atlas/GROUP=mcprod as VO-GROUP combination starts with an alfanumeric character (not a point or a dot) . It indicates a localgroup entry in the groupmapfile.

The /VO=atlas/GROUP=* as a VO-GROUP combination indicates that all users from the Atlas VO with every other groups (except for 'mcprod') will be mapped to the '.atlas' pool of (system) groups. Similar to the poolaccount plugin, this plugin links an entry (in this case a VO-GROUP-ROLE combination) to a locally known group (from this 'atlas'-pool there for a.k.a. pool group).

Using the poolgroup plugin, '/VO=atlas/GROUP=mcprod' can be mapped to the group 'atlas001' (based on the .atlas pool). The '/VO=atlas/GROUP=dev' entry can also be assigned to a group from this '.atlas' pool, but it will be mapped to a different group, e.g. 'atlas002'.

For every value in the Plugin Manager, there is a search in the groupmapfile. [Again, I don't understand it.] The first VO-GROUP-ROLE combination will set to the primary group, the others will be set to secondary groups. As the primary GID being used for auditing and accounting purposes, it is important that the user keeps the correct order of VO-GROUP-ROLE combinations in the grid credential (X509 certificate).

5.5. Proposed Architecture

There are two scenarios:

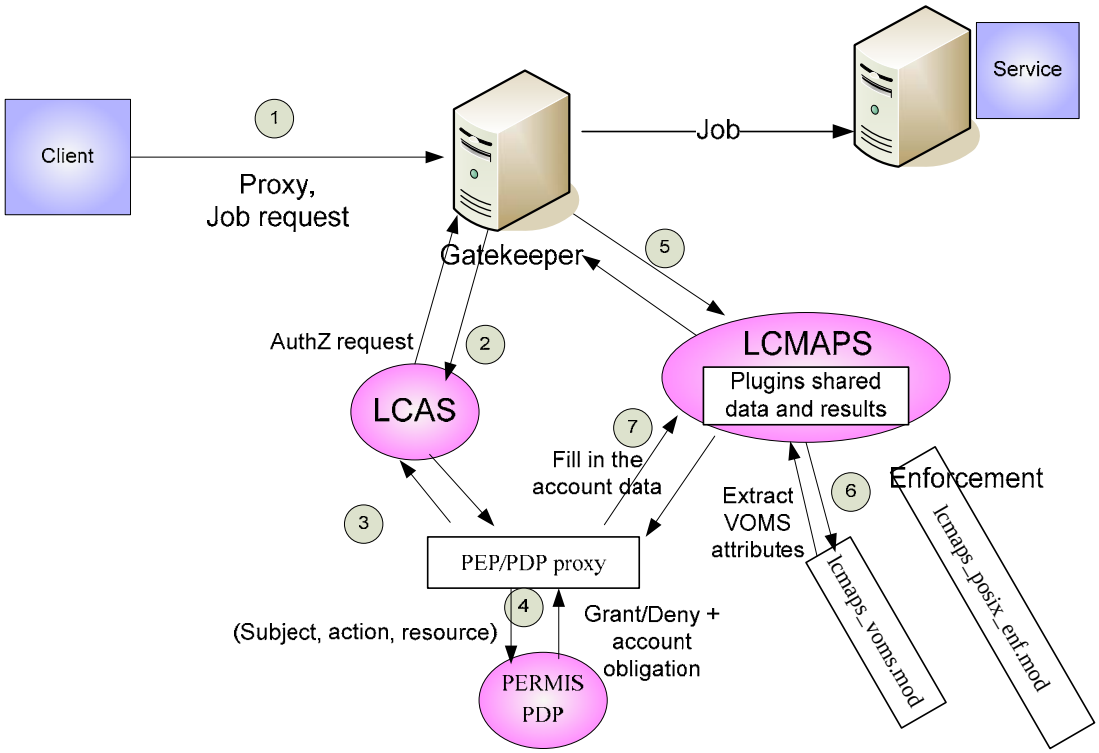
- Replace LCAS and LCMAPS by PERMIS, which should, in addition to authorisation, supply and enforce account obligations. PERMIS can easily replace LCAS for authorisation. However; replacing LCMAPS requires at least supplying the LCMAPS

plugins functionalities, which are looking for a local user account, getting a pool account, primary, secondary and pool groups.

- When plug PERMIS into LCAS and LCMAPS, PERMIS can be easily plugged into LCAS for authorization. The architecture becomes more complicated when PERMIS is plugged in LCMAPS within the same architecture.

We should also decide what features PERMIS should supply to LCMAPS. The LCMAPS plugins help the policy writer by avoiding to specify uid/gid(s). The plugins gather information about uid, primary and secondary gids in the followoing ways.

1- PERMIS obligation policy indicates the uid/gid according to the user voms attributes. This requires the policy writer to know the necessary uid and gid(s). The obligation policy may need to remember in a certain way, the currently allocated accounts and groups in order to avoid for instance accounting problems.
2- PERMIS obligation policy indicates a user account and group as it is working with different plugins. This means that the obligation service should search for uid, gid(s) using an account or group name. It acts as plugins when mapping an account to a user.



The PEP/PDP proxy provides the obligation data to LCMAPS shared data handles.

Enforcement plugins or any other required plugins may be called later. These plugins may even change returned data from PERMIS, e.g. getting and modifying secondary gids.

The PEP/PDP proxy implements two interfaces, one for LCAS plugin and the other for LCMAPS plugin. Considering that middleware that calls LCAS and LCMAPS in different orders, the initialization within these two interfaces should be the same.

6. Appendix A

The following is an example of init.cfg for PERMIS configuration, where policy is local:

```
ini: soa=cn=testuser,ou=issrg,o=kent,c=gb
...: oid=1.2.3.4
 ...: ac=/home/globus/pip/testUserpolicy.ace
...: pkc=/home/globus/pip/testusercert.cer
...: rootca=/home/globus/pip/myCA.pem
...: init
```

The “soa” parameter indicates the issuer of the policy that PERMIS should use. PERMIS will then look for a policy of this issuer in every indicated policy location.

Please note that this “soa” parameter is different from “SOA” entity inside the policy.

In case of the policy being embedded in a signed attribute certificate, the “soa” should be the issuer and the holder of the certificate too.

The “oid” parameter indicates that the oid of the policy should be used. Evidently, this policy should be signed by “soa”.

“ac” parameter is used to indicate the location (local file system) of a policy certificate signed by “soa”.

The “pkc” parameter defines a path to the public key certificate of “soa” in order to validate the certificate chain.

The “rootca” indicates a path to the root CA certificate.

The following is an example of init.cfg for PERMIS configuration, where policy is in an ldap repository:

```
ini: soa=cn=root,ou=GlobusTest,o=Grid
...: oid=1.2.3.4
...: rootca=/home/ac/grid-cacert.cer
...: pkcattribute=userCertificate;binary
...: acattribute=attributeCertificateAttribute
...: url=ldap://dhcp1094.ukc.ac.uk/
...: init
```

The “pkcattribute” indicates which attribute in ldap is used to look for a PKC, which may be used to sign attribute certificates. In this example, it is in the userCertificate parameter.

The “acattribute” indicates which attribute in ldap is used to look for the policy certificate. In this example, it is `attributeCertificateAttribute`.

The “url” parameter indicates the url of the ldap repository. url parameters can also be used to indicate webDAV root entries, file stores and SAML web services.