



UNICORE VO SUPPORT

UNICORE Team

Document Version:	1.0.1
Component Version:	1.4.0
Date:	09 05 2011

This work is co-funded by the EC EMI project under the FP7 Collaborative Projects Grant Agreement Nr. INFSO-RI-261611.

This work was co-funded by the EC Chemomentum project under the FP6 Grant Agreement Nr. IST-033437.

Contents

1	Overview	1
1.1	The modes	1
1.2	Features	2
1.3	Using UVOS server as rich UADB replacement	2
1.4	Supported VO servers	4
2	Configuration	4
2.1	Configuration in UNICORE Services Environment	4
2.2	Main VO subsystem configuration file	5
2.3	Logging configuration	10
3	Short PULL-mode UVOS HOWTO for Unicore/X	11
3.1	Preparation of UVOS based authorization	11
3.2	Scenario 1: Using UVOS instead of XUADB	13
3.3	Scenario 2: Using non standard, fine grained authorization	13
4	Notes for developers	15
4.1	Pull mode support	15
4.2	Push mode support	16

Note

This library version is used in and applies to **UNICORE Servers release 6.4.0**. It may be applicable also to newer versions but always check for this manual updates if using a more recent version of UNICORE servers.

This documentation is a manual of the VO subsystem which is used by all UNICORE servers based on UNICORE Services Environment container (before the 6.4.0 release those based on WSRFLite). The principal servers using this module are: Unicore/X, Registry, Workflow Service and Service Orchestrator.

Additionally this module can be used by programmers directly to easily add support for SAML-compliant VOs to any XFire based web services.

Please note also the following places for getting more information about UNICORE in general:

UNICORE Website: <http://www.unicore.eu>

UVOS Website: <http://uvos.chemomentum.org>

Support list: unicore-support@lists.sf.net

Developer's list: unicore-devel@lists.sf.net

1 Overview

VO (Virtual Organisation) is a quite broad concept. Here (as for now) we use it in a quite simple way, but still offering great benefits for grid administrators.

VO server software is used to store identities of grid entities along with their attributes. Entities are managed with the usage of groups to help administration. Those attributes can be used e.g. for authorization purposes. This document describes how to take advantage of this approach in any XFire based service along with some additional features as how to use a VO service to obtain XUADB-like data (e.g. UNICORE Xlogin).

1.1 The modes

There are two basic modes of operation supported by this subsystem:

- **PULL mode** In this mode attributes are pulled (fetched) by the module from a VO service specified in a configuration file when a new request arrives. This mode is transparent for Unicore/X client.
- **PUSH mode** In this mode it is user's software responsibility to retrieve attributes from a VO service. The VO service must return the attributes in a digitally signed document which is called an *attribute assertion*. This assertion must be attached to the client's request. The module only verifies if the assertion is correct and if is signed by a trusted VO service.

It is possible to use both modes simultaneously. This package contains also code for supporting PUSH mode on client side: there is possibility to easily configure assertions that should go with the request.

1.2 Features

A typical usage of this module is to make service authorization process more flexible. E.g. Unicore/X XACML policy can require certain attributes in order to grant an access to a resource (e.g. to TSS resource which allows for submitting jobs). Those attributes can be retrieved from a VO service (in any mode).

In a classic situation XUADB service can be seen as a <very> trivial VO service. It is limited to carry only one attribute *role* and additional Xlogin information for any user. The Xlogin is used to map a user's certificate to a local account name. Therefore XUADB stores Xlogin information and role with an associated VSite identifier (so every VSite can have different mappings).

This module can be configured to use attributes retrieved from a VO service to simulate XUADB features: selected attribute can be mapped to XUADB *role* and another selected attribute can be used as a Xlogin. Additionally it is possible to define groups to which user is mapped and even allowed BSS queues. What is also special about VO services are the groups (often hierarchical) to which entities belong. This subsystem takes advantage of groups feature to establish where are defined XUADB-like mappings for the local system. It is also possible to use the fact that a user is (or isn't) a member of a VO or its group as an authorization constraint.

1.3 Using UVOS server as rich UADB replacement

If you want to use UVOS instead of classic XUADB you can simply store mappings of certificates to accounts in a special attribute. Also UNICORE roles can be handled this way. XUADB can be then even not installed.

There are three issues with the above approach. The rest of this section explains how to easily solve them.

First of all it must be defined how to recognize account mappings for the VSite, as UVOS service can be used (and usually is) by many VSites concurrently. The solution is that a special group is created for each VSite (e.g /VO1/UADB/SiteA) and only attributes which are assigned in this group scope are used by the VSite. Of course VSite configuration contains a parameter which specifies the group.

The next issue is how to handle a situation when there are multiple Xlogins/roles available for the user, and how to mark the default one? To overcome this for every incarnation attribute it is possible to define two attributes. Base one can possess many values (e.g. in case of Xlogins every value is different Xlogin) while the additional attribute holds the single default value. When there is no need for multiple values then the base attribute can be used alone. When default attribute is defined then its value overrides the value of base one. This is useful in PULL

mode. In PUSH mode user has freedom of choice. E.g. she can ignore default one, select anyone of base values and present only that one to the service.

The last issue is the choice of names for the attributes. You can use any you wish, however there are predefined values, used in examples and as default configuration values. The default ones are predefined in the default configuration file. The following list enumerates the currently known UNICORE incarnation attributes:

UNICORE incarnation attribute	Description
<i>xlogin</i>	This mandatory attribute provides an XLogin for the user, i.e. her local system user id (uid).
<i>role</i>	This attribute is also usually mandatory. It defines a role of the user. The role <i>user</i> is required for normal usage of the UNICORE site. Other possible values are <i>admin</i> (for the super user) and <i>trusted-agent</i> for the trusted server which can have full access to the site.
<i>group</i>	This optional attribute defines user's UNIX group ID (the primary gid).
<i>supplementaryGroups</i>	This optional attribute defines user's UNIX supplementary group IDs.
<i>addDefaultGroups</i>	This optional attribute specifies whether gids which are defined for the xlogin in the operation system should also be added.
<i>queue</i>	This optional attribute specifies range of allowed queues which can be used by the user.

The default configuration is usually fully acceptable however you can modify it to better suit you needs. You can define default values to be used (e.g. if user is allowed to submit to several queues, which one is the default one?), disable an attribute mapping (without deleting it) or even disable a mapping only in one (push or pull) mode. The following example shows an advanced configuration. See [configuration reference](#) Section 2 for a formal syntax definition of all possible entries.

```
# standard settings for the xlogin mapping, however let's ignore ←
  pushed xlogins
vo.unicoreAttribute.xlogin=urn:unicore:attrType:xlogin
vo.unicoreAttribute.xlogin.default=urn:unicore:attrType: ←
  defaultXlogin
vo.unicoreAttribute.xlogin.pushDisabled=

#standard role mapping
vo.unicoreAttribute.role=urn:unicore:attrType:role
vo.unicoreAttribute.role.default=urn:unicore:attrType:defaultRole
```

```
#supplementary groups are stored in a non standard attribute
vo.unicoreAttribute.supplementaryGroups=urn:ourCompany: ←
    secondaryGids

#and group - without default
vo.unicoreAttribute.group=urn:unicore:attrType:primaryGid

#queue mapping is defined, but will be ignored (disabled)
vo.unicoreAttribute.queue=urn:unicore:attrType:queue
vo.unicoreAttribute.queue.default=urn:unicore:attrType:defaultQueue
vo.unicoreAttribute.queue.disable=

# addDefaultGroups - is not defined, so won't be mapped
```

1.4 Supported VO servers

Currently there are several implementations of services that can be used as server side. This module was tested and works well with UVOS system (see <http://uvos.chemomentum.org>). There are other possibilities and you can try to use any SAML (2.0) Attribute service. We are interested in all success/failure stories!

2 Configuration

This sections describes the default configuration file format which is used e.g. in Unicore/X and other USE based servers.

In the following sections some of the configuration options require value of a VO/GROUP type. Whenever it is needed it should be written in the following way:

```
/VO[/group1[/subgroup2[...]]]
```

where elements in square brackets are optional. E.g. `/Math/users` denotes a group *users* of a VO called *Math*.

2.1 Configuration in UNICORE Services Environment

In case of Unicore/X and other USE servers there are three configuration files related to a VO setup. The most important one is by default the `vo.config` from the configuration directory of the server (you can change location and name of this file, see below). It holds generic VO configuration which is not dependent to the actual server used - the most of settings is configured there. This file options are described [below](#) Section 2.2.

To enable the VO subsystem certain settings are also required in the main server's configuration file (for Unicore/X this is `uas.config`). You have to define an appropriate Attribute Source.

There are two VO-related Attribute Sources: one for each mode. You can add them both, only one or even use them multiple times. The latter situation occurs when you want to support multiple VOs (from one or multiple VO servers) - then you have to define one attribute source per VO.

You can leave default XUADB Attribute Source if you wish to use classic UNICORE XUADB and VO subsystem only for authorization attributes. It is also possible to use both XUADB Attribute Source and VO Attribute Source - then the order of sources and combining algorithm is important (see Unicore/X documentation for details).

Example with all three attribute sources. Local data from XUADB (if exists) will override incarnation attributes received from VOs (in any mode):

```
uas.security.attributes.order=SAML-PUSH SAML-PULL XUADB
# ... here xuadb configuration

uas.security.attributes.SAML-PUSH.class=eu.unicore.uas.security.vo. ←
    SAMLPushAuthoriser
uas.security.attributes.SAML-PUSH.configurationFile=conf/vo.config

uas.security.attributes.SAML-PULL.class=eu.unicore.uas.security.vo. ←
    SAMPullAuthoriser
uas.security.attributes.SAML-PULL.configurationFile=conf/vo.config
```

The `ZZZ.configurationFile` configuration option is not necessary if you want to use the `conf/vo.config` file for both attribute sources (as `conf/vo.config` is a default configuration file). However this option is very important when you want to use more than one UVOS PULL attribute source, as it allows you to use different configuration files for them.

When using both PUSH and PULL mode together you can disable PULL mode dynamically for those requests that contain pushed assertions only. See `vo.pull.disableIfAttributesWerePushed` configuration option.

The second required file is the so called *VO truststore* file. This is a normal Java keystore file, which should contain **ONLY** the certificates of the trusted VO servers. Note that this file must not contain any CA certificates, only the trusted VO servers' certificates! See [below Section 2.2](#) (the `vo.truststore*` options) how to configure this file location.

Logging configuration is done by means of standard UNICORE logging configuration file. See [logging Section 2.3](#) section for possible settings related to the VO subsystem.

Note that in Unicore/X there is a simple program which allows you to test VO Push setup. The application is called without arguments and is called `testVOPull`.

2.2 Main VO subsystem configuration file

The main configuration file (usually `vo.config`) can be used to configure both PULL and PUSH modes. The following three sections provide complete reference of available options.

2.2.1 General configuration

General configuration section holds settings which are used for both modes, so you should always properly set them.

Property name	Range of values	Default value	Description
<code>vo.unicoreAttribute.NAME</code>	URI		Attribute used as UNICORE internal incarnation attribute NAME. For further explanations see UUDB Support Section 1.3 .
<code>vo.unicoreAttribute.NAME.default</code>	URI		Attribute used as a default for UNICORE internal incarnation attribute NAME. For further explanations see UUDB Support Section 1.3 .
<code>vo.unicoreAttribute.NAME.disabled</code>	ANY, IGNORED		When this attribute is present regardless of its value the NAME attribute won't be mapped. For further explanations see UUDB Support Section 1.3 .
<code>vo.group</code>	VO/GROUP	""	VO or group which is accepted by this attribute source. Server will honour only attributes with exactly this scope or global (i.e. without scope set).

Property name	Range of values	Default value	Description
<code>vo.truststore</code>	Name of keystore file	""	Defines a truststore, with certificates (not corresponding CA's certificates!) of trusted VO services. Never use SSL truststore of UnicoreX for this purpose as it effectively turns off the whole authorization! It is used for push mode (assertions issued by untrusted VO services are ignored) and in pull mode when signature verification is enabled.
<code>vo.truststoreType</code>	PKCS12 or JKS	JKS	Type of VO truststore.
<code>vo.truststorePass</code>	string		Password of VO truststore.
<code>vo.localServerURI</code>	URI	http://example.org:5000/	Should contain this server URI. It is REQUIRED if pull mode is enabled, and is used to identify this server to the VO service. In push mode it is used as this server actor's name (note that assertions in WS security element with no actor set are also used).

2.2.2 PULL mode configuration

Property name	Range of values	Default value	Description
<code>vo.pull.enable</code>	true, false	false	Defines if pull mode should be enabled.
<code>vo.pull.enableGenericAttributes</code>	true, false	true	If you want to use VO service ONLY as a replacement of XUADB (i.e. to store xlogin mappings and roles) and NOT to use it as a source of other authorization attributes set it to false.
<code>vo.pull.voHost</code>	host name or IP address	localhost	Address of SAML VO service host. Note that this server's CA cert must be present in Unicore/X truststore.
<code>vo.pull.voPort</code>	1-65535	2443	Port of SAML VO service.
<code>vo.pull.voPath</code>	URI path	""	If using UVOS as a service you can leave this field empty. Otherwise specify correct path part of VO service URI.
<code>vo.pull.cacheTtl</code>	integer	600	Controls pulled attributes cache. Set to negative integer to disable the caching or to positive number - lifetime in seconds of cached entries.

Property name	Range of values	Default value	Description
<code>vo.pull.verifySignatures</code>	true,false	true	Additional security (except transport level which is always on) can be achieved by verification of signatures. The key which is used for verification must be present in <code>vo.truststore</code> .
<code>vo.pull.verifySignatures.alias</code>	string	""	Alias used for signature verification (only if <code>vo.pull.verifySignatures</code> is true).
<code>vo.unicoreAttribute.NAME.pullDisabled</code>	ANY, IGNORED		When this attribute is present regardless of its value the pulled NAME attributes won't be mapped. For further explanations see UADB Support Section 1.3 .
<code>vo.pull.enableTrustDelegation</code>	true, false	true	If you enable trust delegation support then attributes will be pulled on behalf of the caller (of course if he/she delegated his/her trust to this service). This is useful as then there is no need to add this server's identity to the VO service and assign read permissions to it (by default everybody can read her/his own data).

Property name	Range of values	Default value	Description
<code>vo.pull.disableIfAttributesWerePushed</code>	true, false	true	Whether pull mode should be skipped if user sent (or pushed) some attributes with the request. Note that to make this feature work PUSH mode must be enabled AND PULL authorization must be invoked AFTER PUSH authorization.

2.2.3 PUSH mode configuration

Property name	Range of values	Default value	Description
<code>vo.push.enable</code>	true, false	false	Defines if push mode should be enabled.
<code>vo.unicoreAttribute.NAME.pushDisabled</code>	ANY, IGNORED		When this attribute is present regardless of its value the pushed NAME attributes won't be mapped. For further explanations see UADB Support Section 1.3 .

2.3 Logging configuration

All components use log4j logging mechanism. All events are logged with `unicore.security.vo` prefix. Further logging category is either `pull`, `push` or `common`. Finally reporting class name is appended.

Example snippet of log4j configuration for logging all events for VO subsystem but only *INFO* and higher events for PUSH mode can be specified as follows:

```
log4j.logger.unicore.security.vo=TRACE
log4j.logger.unicore.security.vo.push=INFO
```

3 Short PULL-mode UVOS HOWTO for Unicore/X

This section shows all the basic steps which are required to setup a Unicore/X and UVOS to work in PULL mode. Important fact to note here (and in case of PUSH mode too) is how the user's group membership is encoded as an XACML attribute. By default it is an attribute of string type (so XACML *DataType*="http://www.w3.org/2001/XMLSchema#string") with its name (*AttributeId*) equal to *urn:SAML:voprofile:group*. The example policy below uses this attribute.

There are two scenarios described. Most often the simpler scenario number one is used.

3.1 Preparation of UVOS based authorization

This section provides an overview of preparation steps which are mostly common for both scenarios and PULL style usage of UVOS in general. Both scenarios refer to the necessary steps from the list below.

The required steps are:

1. Add UVOS server's CA certificate to the Unicore/X's main truststore.
2. Add Unicore/X's CA certificate to the UVOS server's truststore.
3. *Often this step is optional. If you enable trust delegation support in the VO configuration file (as in the example configuration below) you can jump to the next point. However note that (very rarely) workflow system services do not forward trust delegation assertions so it is better to perform this step to be 100% sure everything will work always correctly.*

Add Unicore/X's DN (from its certificate) as a member to the UVOS service. You don't have to make it a member of any particular VO (or group). However it must have the **read** permission to all groups where its users will be placed. UVOS documentation contains details.

4. Create a VO (possibly with subgroups). Add grid users to the VO and assign them appropriate attributes. Scenarios below describe this step more precisely.
5. Update `uas.config` file to enable VO subsystem. Here we will configure UVOS as the primary source and leave XUADB to provide local mappings (which can override data fetched from UVOS). You should have the following entries:

```
uas.security.attributes.order=SAML-PULL XUADB
uas.security.attributes.combiningPolicy=MERGE_LAST_OVERRIDES
# ... here goes xuadb configuration

uas.security.attributes.SAML-PULL.class=eu.unicore.uas. ←
security.vo.SAMLPullAuthoriser
```

6. Configure `vo.config` file as described in the [Configuration Section 2](#) section. `vo.config` may look like this:

```
vo.group=/Math-VO/UUDB/SiteA

vo.truststore=conf/votruststore.jks
vo.truststoreType=JKS
vo.truststorePass=the!server

vo.localServerURI=http://example.org:7777

# #####
# PULL mode configuration
# #####

vo.pull.enable=true

vo.pull.cacheTtl=20

vo.pull.voHost=uvos.example.org
vo.pull.voPort=2443
vo.pull.voPath=

vo.pull.verifySignatures=false

vo.pull.enableTrustDelegation=true

# Mapping of UVOS attributes (right side) to the special, ←
#   recognized by UNICORE
# incarnation attributes (left)
vo.unicoreAttribute.xlogin=urn:unicore:attrType:xlogin
vo.unicoreAttribute.xlogin.default=urn:unicore:attrType: ←
#   defaultXlogin
vo.unicoreAttribute.role=urn:unicore:attrType:role
vo.unicoreAttribute.role.default=urn:unicore:attrType: ←
#   defaultRole
vo.unicoreAttribute.group=urn:unicore:attrType:primaryGid
vo.unicoreAttribute.group.default=urn:unicore:attrType: ←
#   defaultPrimaryGid
vo.unicoreAttribute.supplementaryGroups=urn:unicore:attrType: ←
#   defaultSupplementaryGids
vo.unicoreAttribute.supplementaryGroups.default=urn:unicore: ←
#   attrType:supplementaryGids
vo.unicoreAttribute.addDefaultGroups=urn:unicore:attrType: ←
#   addDefaultGroups
vo.unicoreAttribute.queue=urn:unicore:attrType:queue
vo.unicoreAttribute.queue.default=urn:unicore:attrType: ←
#   defaultQueue
vo.unicoreAttribute.virtualOrganisations=urn:SAML:voprofile: ←
#   group
```

3.2 Scenario 1: Using UVOS instead of XUUDB

In this scenario we will use UVOS to store at a central point mappings of certificates to UNIX logins (Xlogins) and roles of our users. To do so:

1. Perform steps 1-3 from the generic list above.
2. Choose a group name for storing mappings for your site (actually it may be shared by many sites which use the same mapping). Let's assume it is `/Math-VO/UUDB/SiteA`. Add grid users to this group. Next assign them *in the scope of the group* attribute `urn:unicore:attrType:xlogin` with the value of Xlogin for the user, and attribute `urn:unicore:attrType:role` with the value of the user's role (usually its just `user`). Note that if you want to assign the same Xlogin/role to multiple users then you can define UVOS *group attributes* and set them for the whole `/Math-VO/UUDB/SiteA` group.
3. You can use the same configuration file as in the above scenario so perform steps 5 and 6 of the above section. You don't have to modify default authorization policy.

It is good to test a configuration with a VO Push test utility available in Unicore/X.

3.3 Scenario 2: Using non standard, fine grained authorization

You can configure Unicore/X security policy to require attributes defined and maintained by the UVOS. To do so ensure that you have this setting in `vo.config` file: `vo.pull.enableGenericAttributes=true`. Then you can modify XACML policy to require certain VO attributes.

The following XACML fragment allows to reach TargetSystemFactory service only for the users which are both members of VO `Example-VO` and a VO group `/Math-VO/UUDB/SiteA`. Moreover those users also must have a standard Unicore/X attribute `role` with a value `user`. It means that in UVOS service grid user must have `urn:unicore:attrType:role` attribute defined (it is the standard setting) with a value `user`.

```
<Rule RuleId="AcceptTSF" Effect="Permit">
  <Description>
    Accept selected users to reach TSF
  </Description>
  <Target>
    <Subjects>
      <AnySubject />
    </Subjects>
    <Resources>
      <Resource>
        <ResourceMatch
          MatchId="urn:oasis:names:tc:
            :xacml:1.0:function:
            anyURI-equal">
```

```

        <AttributeValue DataType=" http://www.w3.org/2001/
            XMLSchema#anyURI "
        >TargetSystemFactoryService
        </AttributeValue>
    <
        ResourceAttributeDesignator
            DataType="http://
                www.w3.org
                /2001/XMLSchema
                #anyURI "
            AttributeId="urn:
                oasis:names:tc:
                xacml:1.0:
                resource:
                resource-id" />
        </ResourceMatch>
    </Resource>
</Resources>
<Actions>
    <AnyAction />
</Actions>
</Target>
<Condition FunctionId="urn:oasis:names:tc:xacml:1.0:
    function:and">
    <Apply FunctionId="urn:oasis:names:tc:xacml:
        :1.0:function:string-equal">
        <Apply FunctionId="urn:oasis:names:
            tc:xacml:1.0:function:string-
            one-and-only">
            <SubjectAttributeDesignator
                DataType="http://
                    www.w3.org
                    /2001/XMLSchema
                    #string"
                AttributeId="role"
            />
        </Apply>
        <AttributeValue
            DataType="http://www.w3.org
                /2001/XMLSchema#string
            ">user</AttributeValue>
        </Apply>
    <Apply FunctionId="urn:oasis:names:tc:xacml:
        :1.0:function:any-of-all">
        <Function FunctionId="urn:oasis:
            names:tc:xacml:1.0:function:
            string-equal"/>
        <SubjectAttributeDesignator

```

```

        DataType="http:// ↵
            www.w3.org ↵
            /2001/XMLSchema ↵
            #string"
        AttributeId="urn: ↵
            SAML:vopprofile: ↵
            group" />
    <Apply FunctionId="urn:oasis:names: ↵
        tc:xacml:1.0:function:string- ↵
        bag">
        <AttributeValue DataType=" ↵
            http://www.w3.org/2001/ ↵
           /XMLSchema#string"/> ↵
            Example-VO</ ↵
            AttributeValue>
        <AttributeValue DataType=" ↵
            http://www.w3.org/2001/ ↵
           /XMLSchema#string"/>/Math ↵
            -VO/UUDB/SiteA</ ↵
            AttributeValue>
    </Apply>
</Apply>
</Condition>
</Rule>

```

4 Notes for developers

The API is documented in JavaDocs so you should browse it in most cases. Here is presented a short, general introduction only to show you good starting points. Package names were removed, but in most cases it is *pl.edu.icm.unicore.voutils*.

The classes in this package require quite complex configuration. You can provide it as a custom implementation of an proper interface: either *IPullConfiguration* or *IPushConfiguration*. There is a default implementation of both, which reads data from Java properties file: the *PropertiesBasedConfiguration* class. See [Configuration Section 2](#) for the properties defined.

4.1 Pull mode support

- *VOAttributeFetcher* this class allows for simple fetching of attributes for the effective user from the configured VO service. It can be used in custom authorisers or attribute resolvers.
- *SAMLAttributePullInHandler* is a trivial wrapper around *VOAttributeFetcher* which invokes it as a handler. After adding it, fetched attributes are populated in standard UNICORE security context for every coming request.

4.2 Push mode support

- `ClientVOUtil` allows for configuring `SAMLAttributePushOutHandler` to push previously obtained attributes to the service being used.
- `SAMLAttributePushInHandler` extracts, parses and verifies assertions pushed in a request. Valid attributes are then injected into UNICORE security tokens.