Installation Guide for UNICORE 6 Server Components

UNICORE Team

October 2011, UNICORE version 6.4.2

Contents

This document is based on UNICORE core server 6.4.2 and workflow services release Version 6.3.2 In case of questions please contact UNICORE Support via unicore-support@lists.sf.net

1 Introduction

1.1 Purpose and Target Audience of this Document

This document is intended for administrators who wish to install the UNICORE servers at their site in a multi-user, production environment for operation with a resource management (batch) system such as Torque, Slurm, etc.

This document is NOT suitable if you just want to get a simple running system for testing quickly, in that case you should simply follow the instructions in the README.txt or download and use the graphical installer `unicore-servers-6.4.2.jar`

This document covers the installation and configuration of the UNICORE 6 server components on UNIX/Linux server systems. It is assumed that the `unicore-servers-6.4.2.tgz` and optionally the `unicore-workflow-6.3.x.tgz` packages have been downloaded, and that Java version 6 is installed on those systems where the UNICORE components will run. We assume that a single firewall exists that protects all the servers within the site.

In this document, not every detail and configuration option of UNICORE can be explored. If the scenarios covered here do not match your needs, please have a look into the following reference manuals

- Gateway http://unicore.eu/documentation/manuals/unicore6/files/gateway/manual.html
- UNICORE/X http://unicore.eu/documentation/manuals/unicore6/files/unicorex/unicorex-manual.html
- TSI http://unicore.eu/documentation/manuals/unicore6/files/tsi/tsi-manual.html
- Registry http://unicore.eu/documentation/manuals/unicore6/files/registry/registry-manual.html
- XUUDB http://unicore.eu/documentation/manuals/unicore6/files/xuudb/manual.html

More background information can be found at the UNICORE website (http://www.unicore.eu).

1.2 Overview of the UNICORE Servers and some Terminology

This section is intended as a very brief overview of the server components in a UNICORE Grid, to give you an idea of the purpose of each component and how they interact.

In the following, a "site" is an administrative domain, protected by a firewall. A "Grid" is a collection of sites that can be accessed in a common way.

The *Gateway* is the main entrance to the site, through which the components can be reached from outside the site firewall. All client connections are with the gateway, which will then forward the requests, and send the replies back to the client. Client is used here in the broad sense, since also services can act as clients, for example when transferring a file between sites. The performs initial user authentication, and will reject requests that do not originate from a trusted client. The gateway port is the only port which needs to be open for https connections from the network outside and the firewall, and you will have to configure your firewall accordingly.

The *UNICORE/X* server is the central component in a UNICORE Grid. It hosts services such as job submission and management, storage access, file transfer and others. The UNICORE/X server accepts jobs, submits them to the local systems via the TSI, allows to upload and download data via several protocols, and thus provides most of the functionality of UNICORE. UNICORE/X handles user authorisation, using (configurable) security policies that take the current user, the requested action and a set of user attributes into account. It also requires one or more "attribute sources", for example an XUUDB, which are queried for user attributes. At minimum, one UNICORE/X per target resource is required. It should to be tuned to the performance requirements, and in general needs ample memory, CPU power and local disk space (for persistent data and log files).

The *XUUDB* is one common option that can be used as an attribute source for UNICORE/X. For each Grid user, it provides local attributes like Unix login, the role (e.g. "user" or "admin"). Since the XUUDB is a web service, it can be used from multiple UNICORE/X servers within a site. Other attribute sources exist, from a simple map file to a VO server (UVOS) that can even be used accross sites. Attribute sources may also be combined.

The *Perl TSI* component (often briefly referred to as *TSI*) interfaces with the local systems (batch scheduler, file systems), thus it must be installed on the target resource. In case of a cluster, this will be one of the frontend (or login) nodes. It is a mandatory component in a multi-user setting, and is the only component that runs as "root". One TSI per target resource is required.

The Registry is very much like a UNICORE/X server, however it runs only a single service (the "shared registry" service) which allows clients to discover available services hosted by multiple UNICORE/X servers. The Registry does not need as much memory and processing power as the UNICORE/X server. In a typical UNICORE Grids, there is at least one Registry. For redundancy in large Grids, more than one Registry may be deployed.

The components mentioned (Gateway, UNICORE/X, XUUDB, TSI, Registry) are denoted as "core" services, and can be downloaded together in the "core server" package mentioned above.

The UNICORE workflow system introduces two more components, which again are based on UNICORE/X, but host other types of services. We will introduce them later in the workflow section.

All UNICORE servers are implemented in Java, except the Perl TSI. Thus Java version 6 is required to run the UNICORE servers.

1.3 Overview over this Document

There are different ways for setting up a UNICORE installation. Depending on the number of servers you have and the type of resource you wish to access (e.g. a multinode cluster or just a single server) the optimal setup can vary.

To be concrete, we cover the following common scenarios:

1. Single UNICORE Site for accessing a multi-node cluster. This consists of one Gateway, UNICORE/X, XUUDB, and Registry installed on two servers, with the Perl TSI installed on the cluster frontend. This setup can be seen as a foundation for further extensions, since it already includes a Registry, which is useful in those cases where more than a single UNICORE resource shall be available to clients.
2. Adding another UNICORE/X server, e.g. for accessing another cluster, to an existing installation
3. Multi-site UNICORE installations crossing administrative domains
4. Setting up the UNICORE workflow system

2 Installation of Core Services for a Single Site

This section covers the installation of services at one UNICORE site.

2.1 Basic Scenarios

The basic layout of a UNICORE site is depicted in Figure Figure 1. Each component (Gateway, UNICORE/X, Registry, Perl TSI, XUUDB) has an associated hostname and port setting.

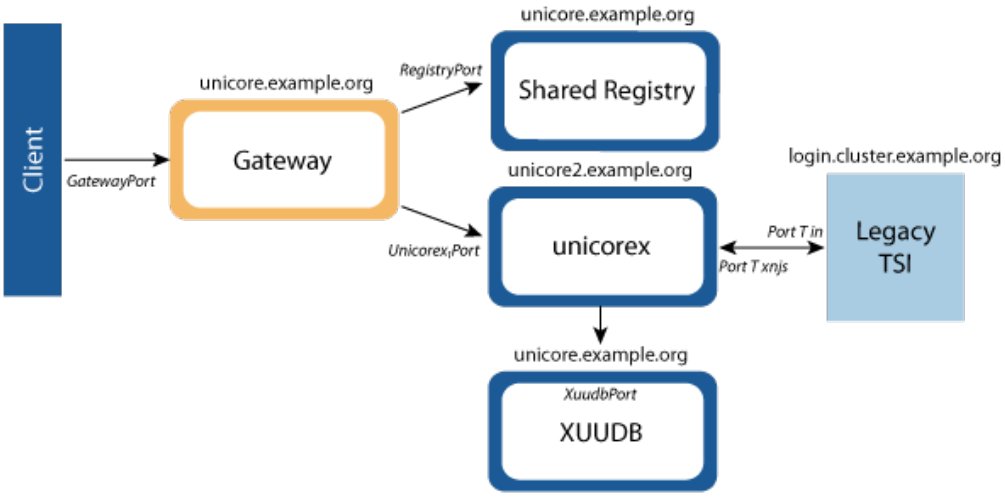

Figure 1: basic single-site configuration with shared registry

Of course all components can be installed on a single machine, but in general you should distribute the components. Even in this simple case, several deployment options exist. To choose the right one, consider the following:

The Gateway port is the only one which needs to be open for https connections from the network outside the firewall. It needs to be able to accept and initiate connections from/to the UNICORE/X and Registry servers.

The UNICORE/X component requires most of the processing power, and memory, and is thus best installed on its own machine.

The Perl TSI component communicates with the local systems (batch scheduler or file systems), thus it must be installed on the target resource. In case of a cluster, this will be one of the login nodes.

In the following two layout options are given for the case where there are two server machines available:

1. Option "load distributed" This option allows to provide more processing power to the UNICORE/X server.

   - Machine 1: Gateway, XUUDB, Registry
   - Machine 2: UNICORE/X

2. Option "security focussed" In a more security oriented environment, the Gateway would run on a separate server, residing in a demilitarised network zone (DMZ), and the other components would run in the intranet. In Figure Figure 1 this option is visualised by the different background colours of the services.

- Machine 1: Gateway
- Machine 2: XUUDB, Registry, UNICORE/X

In any case, you should make a deployment plan, i.e. decide how you wish to distribute the components, and note all the hostnames and ports. The following table provides a template for this deployment plan:

	Hostname	Port usage	Port value	Notes
Gateway		HTTPS (mandatory)		open in firewall
UNICORE/X		HTTPS (mandatory)		
UNICORE/X		TSI Reply (TCP) (mandatory)		
UNICORE/X		JMX management (optional)		to monitor running java application
Registry		HTTPS (mandatory)		
Registry		JMX Management (optional)		to monitor running java application
XUUDB		HTTPS (mandatory)		only if XUUDB is used
TSI		TCP (mandatory)		Cluster frontend node. SSL is optional

2.2 Preparation

For security reasons, you should create a **dedicated user** (say, *unicore*) who will run the server components (Gateway, XUUDB, UNICORE/X, Registry). In a multi-user setting, the Perl TSI component needs to run as root.

Note
For testing, you can run the Perl TSI as a non-privileged user. All requests will then be executed under that user.

The core server components `gateway`, `unicorex`, `registry`, `xuudb`, and `tsi` are available in the bundle `unicore-servers-6.4.2.tgz` at https://sourceforge.net/projects/unicore/files/1%20UNICORE%206%20Servers%20and%20Services/1.1%20Core%20Server/6.4.2/ .

If you haven't already done so, download the .tgz file, and un-tar it into a directory such as `/opt/unicore`.

You have several options to configure and install the components.

- (recommended) You may use the `configure.properties` file and set up the local configuration parameters there before you run the `configure.py` script, which is available on top-level of the `unicore-servers-6.4.2.tgz`.
- You may go through the configuration files and adapt them to your local setup, e.g. hostnames and ports, local paths etc. which should be straight forward. Specific configuration options are covered in the next chapters.

We definitely prefer the first approach, since it will give you a working configuration, which you then can customise later.

So here is the recommended procedure

1. Edit the `configure.properties` file, and customize it according to your deployment plan
2. Install and configure Gateway, XUUDB, Registry, and UNICORE/X
3. Configure security (keystores, truststores) for the components
4. Install the correct Perl TSI for your batch system and configure UNICORE/X for the chosen TSI and local systems
5. Start the servers

2.3 Installation

Here is a full example. It assumes we have two servers, `unicore.example.org` and `unicore2.example.org`, and the TSI will be installed on `login.cluster.example.org`. We'll choose the Torque TSI (linux_torque).

We'll put UNICORE/X on `unicore2` and Gateway, Registry and XUUDB on `unicore.example.org`, corresponding to "option 1" above. The UNICORE/X server will be named *CLUSTER* and the Registry will be named *REGISTRY*.

Note
We will also enable the StorageFactory service, which allows clients to create directories in a base directory on the cluster filesystem, and use them at their discretion. This is very useful if later the Workflow services are added.

You should copy and untar the `unicore-servers-6.4.2.tgz` on both servers and on the cluster login node, say into `/opt/unicore/unicore-servers-6.4.2`

After untarring, you'll have the following basic layout

```
unicore-servers-6.4.2
- gateway
   * bin
   * conf
   * lib
   * logs
- registry
   * bin
   * ...
- unicorex
   * bin
   * ...
- xuudb
   * bin
   * ...
- tsi
   * bin
   * ...
- configure.properties
- configure.py
- install.py
- certs
- docs
- README
- CHANGES.txt
- start.sh
- stop.sh
```

Now you should edit the `configure.properties` file and enter all the hostnames and ports that you have decided on.

configure.properties

```
#
# Configuration properties
#
# this file is read by the configure.py script.
#
# Note:: the special parameter value "hostname" will be replaced
```

```
# by the hostname specified on the command line or
# found by lookup.
#

[parameters]

#
# target base directory, use "currentdir" to leave as-is
# (otherwise use an absolute path, such as "/opt/unicore")
#
INSTALL_PATH=currentdir


# which components to configure

#
# the Gateway
#
gateway=true

#
# the UNICORE/X server
#
unicorex=true

#
# the Perl TSI (check also the xnjsConfigFile property below)
#
tsi=true

#
# the shared registry server
# (if you want UNICORE/X to use this or another external registry,
#  make sure to edit the relevant settings below)
#
registry=true

#
# the XUUDB user database
#
xuudb=true

#
# shall the demo user cert be added to the xuudb and the map file?
#
installdemouser=false

#
# shall the docs be copied by install.py?
#
```

```
docs=true

#
# Java command to use
#
JAVA_CMD=java


#
# Gateway host and port
#
gwHost=unicore.example.org
gwPort=8080

#
# enable auto-registration of the UNICORE/X server with the gateway ↩
    ?
#
gwAutoRegistration=false

#
# add a line to connection.properties for the registry
# defaults to the value of the "registry" parameter
#
gwAddRegistryEntry=${registry}

#
# UNICORE/X server host and port
#
uxHost=unicore2.example.org
uxPort=7777
# Port used to connect to the UNICORE/X server via JMX
# (leave empty to disable JMX)
uxJmxPort=9128

#
# VSite name
#
uxName=CLUSTER

#
# XNJS configuration file
#
# xnjs.xml         : uses the embedded Java TSI
# xnjs_legacy.xml  : uses the Perl TSI
#
uxXnjsConfigFile=xnjs_legacy.xml

#
# settings for defining the external registry
```

```
#

# register with an external Registry?
uxUseExternalRegistry=true

# if yes, auto-discover?
uxFindExternalRegistry=false

#
# Alternatively, give fixed URL to the registry
# This URL is
uxUrlExternalRegistry=https://${gwHost}:${gwPort}/${registryName}/ ↩
    services/Registry?res=default_registry


#
# Enable the StorageFactory service for this UNICORE/X server?
# A StorageFactory should ideally be running on sites with a  ↩
    powerful,
# high capacity filesystem
#
uxEnableStorageFactory=true

#
# For the default storage factory, where on the cluster file system ↩
     should the files be created
# This has to be world executable (like the filespace directory)
# I.e. root should execute a 'chmod 1777' on this directory
#
uxDefaultStorageFactoryPath=${INSTALL_PATH}${FILE_SEPARATOR} ↩
    unicorex${FILE_SEPARATOR}storage-factory

#
# Enable the OGSA BES interface
# (leave empty to disable BES)
#
uxEnableBES=de.fzj.unicore.bes.util.BESOnStartup

#
# UNICORE/X GCID: the ID used by the UNICORE/X server for querying  ↩
    the XUUDB
#
uxGCID=CLUSTER

#
# VO Server configuration
#

# The following settings are enough for an UVOS server. For others  ↩
    you
```

```
# may need to adapt the generated configuration

# Which group is used to store Xlogins and Roles for this site?
voGroup=/demo-vo/DEMO-SITE
# whether to query the VO server (aka PULL mode)?
# set to "VO-PULL" to enable, leave empty to disable
voEnablePull=
#voEnablePull=VO-PULL
voPullServerHost=hostname
voPullServerPort=2443

# whether to process SAML assertions pushed by clients
# set to "VO-PUSH" to enable, leave empty to disable
voEnablePush=
#voEnablePush=VO-PUSH

#
# TSI host and port
# (i.e. port on which the TSI process will listen for XNJS requests ↩
    )
tsiHost=login.cluster.example.org
tsiMyPort=4433

# The port on which the XNJS is listening for TSI worker  ↩
    connections
tsiNjsPort=7654

# The user id for querying job info from the batch system (cannot  ↩
    be root!)
tsiPrivedUser=unicore

#
# The directory on the target system where the job directories will ↩
     be created
# (On a cluster, this should be a shared filesystem!)
# The directory must be accessible for every user id,
# I.e. root should execute a 'chmod 1777' on this directory
#
tsiFilespace=${INSTALL_PATH}/FILESPACE

#
# The selected TSI (i.e. directory containing platform-specific
# TSI files)
#
tsiSelected=tsi/linux_torque

#
# Where to install the TSI
# (absolute path)
#
```

```
tsiInstallDirectory=${INSTALL_PATH}/tsi-torque


#
# Common information provider (CIP) settings
# This data is used to provide information about this site to
# the Common Information Service (CIS)
#
cipLatitude=52.0
cipLongitude=4.888573
cipCPUClockSpeed=3000
cipTotalCapacity=204800


#
# XUUDB server host and port
#
xuudbHost=unicore.example.org
xuudbPort=34463

# XUUDB type (dn or normal)
xuudbType=normal


#
# Shared registry UNICORE/X server host and port
#
registryHost=unicore.example.org
registryPort=7778
# Port used to connect to the UNICORE/X server via JMX
# (leave empty to disable JMX)
registryJmxPort=9129

#
# VSite name
#
registryName=REGISTRY

#
# registry GCID: the ID used by the registry UNICORE/X server for ↩
    querying the XUUDB
# this is only relevant if access control on the registry is ↩
    enabled
# (see also the registry/conf/uas.config file)
#
registryGCID=CLUSTER
```

Copy your `configure.properties` somewhere safe, e.g. to your home directory, for later use and reference.

Now you're ready to configure the components. In our example scenario, this would be as follows

```
ssh unicore@unicore.example.org
cd /opt/unicore/unicore-servers-6.4.2
cp ~/configure.properties .
./configure.py [installation userid] [hostname]
```

and the same for unicore2.example.org and the cluster login node.

This will insert your configuration values into the relevant files. In principle your components are now ready to run, however still the demo certificates are used. In a production setting, you'll need to setup keystores and truststores with your production certificates, which is covered in the next section.

2.4 Security Settings

This section addresses the security setup of UNICORE 6 server components. Public Key Infrastructure (PKI) is used for setting up SSL connections between the components. Each component has its own credentials (private and public key of an x509 certificate), signed by a trusted Certificate Authority.

Furthermore the Gateway and UNICORE/X servers need to be able to ascertain the validity of user certificates, for which they require a truststore containing all relevant CA certificates. The Registry and XUUDB need to trust only the CA that issued the Gateway and UNICORE/X certificates.

All server components support the PKCS12 and JKS (Java Key Store) keystore formats. PKCS12 is commonly used for holding the credentials (private keys) whereas JKS can be used for both holding the private keys and also the trusted certificates. This makes the truststore (a keystore with only trusted certificates entries) exchangeable between the server components. It is good practice to use two distinct keystores for each server component: One for the server's credential (PKCS12 or JKS) and one for the truststore (JKS). Tools for creating a truststore from certificates (pem files) are e.g. keytool from the Java runtime and Portecle.

2.4.1 Gateway

The configuration for key- and truststore is done in `gateway/conf/security.properties` The truststore must contain every CA certificate signing user certificates of those who are allowed to access the site. The `gateway` also has to trust connections from the other services. Thus the CA certificate signing the services' certificates must also be added to the truststore.

configure the credentials

```
# keystore.p12 contains the private key and the certificate
# (public key signed by the CA)
keystore=/path/to/your/gateway/keystore.p12
keystorepassword=******
```

configure the truststore

```
truststore=/path/to/your/gateway/truststore.jks
truststorepassword=******
truststoretype=jks
```

The gateway supports the truststore type "directory", where all pem files in a given directory are trusted. This is configured as follows

directory truststore

```
truststore=/etc/unicore/trusted-certs
truststorepassword=unused
truststoretype=directory
```

2.4.2 UNICORE/X

The configuration for key- and truststore is done in `unicorex/conf/wsrflite.xml`. UNICORE/X has to trust connections from the Gateway, and needs to be able to validate user certificates. Thus you have to add all the relevant CA certificates to the truststore.

configure SSL properties

```
<!-- SSL -->
<property name="unicore.wsrflite.ssl" value="true"/>
<property name="unicore.wsrflite.ssl.clientauth" value="true"/>
```

`Set both values to "true".`

configure the credentials

```
<!-- UNICORE/X server identity -->
<property name="unicore.wsrflite.ssl.keystore"
          value="/path/to/your/unicorex/keystore.p12"/>
<property name="unicore.wsrflite.ssl.keypass" value="*******"/>
<property name="unicore.wsrflite.ssl.keytype" value="PKCS12"/>
<property name="unicore.wsrflite.ssl.keyalias" value="xnjs_id"/>
```

configure the truststore

```
<!-- UNICORE/X truststore -->
<property name="unicore.wsrflite.ssl.truststore"
          value="/path/to/your/unicorex/truststore.jks"/>
<property name="unicore.wsrflite.ssl.truststorepass" value ↩
    ="*******"/>
<property name="unicore.wsrflite.ssl.truststoretype" value="JKS ↩
    "/>
```

The gateway supports the truststore type "directory", where all pem files in a given directory are trusted. This is configured as follows

directory truststore

```
<!-- UNICORE/X truststore -->
<property name="unicore.wsrflite.ssl.truststore"
          value="/etc/unicore/trusted-certs"/>
<property name="unicore.wsrflite.ssl.truststorepass" value=" ↩
    unused"/>
<property name="unicore.wsrflite.ssl.truststoretype" value=" ↩
    directory"/>
```

2.4.3 XUUDB

The configuration for key- and truststore is done in `xuudb/conf/xuudb_server.conf`. The XUUDB needs to trust the UNICORE/X server(s) and the administrator. Usually the administrator uses the XUUDB certificate.

activate SSL

```
xuudb_use_ssl=true
```

configure the credentials

```
xuudb_keystore_file=/path/yo/your/xuudb/keystore.p12
xuudb_keystore_type=PKCS12
xuudb_keystore_password=******
```

configure the truststore

```
xuudb_truststore_file=/path/to/your/xuudb/truststore.jks
xuudb_truststore_type=JKS
xuudb_truststore_password=******
```

configure xuudb.acl The access control list has to contain the DNs (distinguished names) of the XUUDB administrators, i.e. those certificates which are allowed to modify the content of the XUUDB. This is in particular the DN of the credentials (.p12 file) defined in `xuudb/conf/xuudb_client.conf`, to enable the administrator logged on to the system `unicore.example.org` to list or manipulate the content of the XUUDB.

example xuudb access control list

```
#
# example XUUDB ACL file
#
CN=uudbadmin,OU=example lt,O=some ca,C=eu
```

2.5 Installation of the Perl TSI and TSI-related configuration of the UNICORE/X server

During the previous installation step, the files were copied to the target system, in our example to `login.cluster.example.org`, into the directory `/opt/unicore/unicore-6.4.2/tsi-torque`.

You can repeat this step if necessary (for example for choosing a different batch system), by doing

```
cd /opt/unicore/unicore-6.4.2/tsi
./Install.sh
```

Note
Several TSI implementations are available in the sub-directory `./tsi`. The specific TSIs with batch sub-system are composed of all common files from `./tsi/SHARED` plus the files for the specific operation system and/or batch sub-system, e.g. `./tsi/linux_torque`. The required files together with config files and start/stop scripts are copied into a new directory by the `Install.sh` script

Please review the `<tsi_installation_directory>/perl/tsi` file. Hostnames and ports have already been set by the `configure.py` step, but it might be necessary to adapt the path settings, and the location of the batch system commands (`qsub` etc).

Also review the basic configuration of the TSI `conf/tsi.properties` file where host name and ports are set as well as the logging directory.

When you are done you should execute (as root) `./Install_permissions.sh <tsi_installation_directo` to set restricted file permissions for the TSI files and directory. In our example, the `tsi_installation_directory` is `/opt/unicore/unicore-6.4.2/tsi-torque`.

The TSI uses the auxiliary script `tsi_ls` to list files and `tsi_df` to report the free disk space on a file system. Make sure that both scripts are world readable because they have to be read from any user id when executing a ListDirectory request. The `unicorex` configuration (`unicorex/conf/xnjs_legacy.xml`) needs to be changed so that the `tsi_ls` and `tsi_df` files in your TSI installation can be found:

xnjs_legacy.xml

```
<property name="CLASSICTSI.TSI_LS" value="/my_full_tsi_path/perl/ ↩
    tsi_ls"/>
<property name="CLASSICTSI.TSI_DF" value="/my_full_tsi_path/perl/ ↩
    tsi_df"/>
```

unicorex/conf/simpleidb Here available resources, applications, and environments on the target system have to be specified to make them available to the user's client. More detailed information is provided in the IDB documentation.

All of the predefined specifications are optional, so you can either adapt the values to match your system or delete the specification.

Resources section of the simpleidb

```
<jsdl:Exact>
<!-- Resources -->
  <idb:TargetSystemProperties>
    <jsdl:Resources xmlns:jsdl="http://schemas.ggf.org/jsdl ↩
        /2005/11/jsdl">
<!-- CPU architecture, must be one of x86, x86_64, ia64, powerpc,  ↩
    sparc or other -->
      <jsdl:CPUArchitecture>
        <jsdl:CPUArchitectureName>x86_64</jsdl:CPUArchitectureName>
      </jsdl:CPUArchitecture>
<!-- Operating system, must be one of those defined by JSDL, e.g.  ↩
    LINUX, MACOS, WINNT, AIX, -->
      <jsdl:OperatingSystem>
       <jsdl:OperatingSystemType>
         <jsdl:OperatingSystemName>LINUX</jsdl:OperatingSystemName>
       </jsdl:OperatingSystemType>
       <jsdl:OperatingSystemVersion>2.6.38</jsdl: ↩
           OperatingSystemVersion>
       <jsdl:Description>Ubuntu GNU/Linux</jsdl:Description>
      </jsdl:OperatingSystem>
<!-- cpu time (per cpu) in seconds -->
      <jsdl:IndividualCPUTime>
        <jsdl:Exact>3600</jsdl:Exact>
        <!-- Exact denotes the default value -->
        <jsdl:Range>
          <jsdl:LowerBound>1</jsdl:LowerBound>
          <jsdl:UpperBound>86400</jsdl:UpperBound>
        </jsdl:Range>
      </jsdl:IndividualCPUTime>
<!-- Nodes -->
      <jsdl:TotalResourceCount>
        <jsdl:Exact>1.0</jsdl:Exact>
        <!-- Exact denotes the default value -->
        <jsdl:Range>
          <jsdl:LowerBound>1.0</jsdl:LowerBound>
          <jsdl:UpperBound>1.0</jsdl:UpperBound>
        </jsdl:Range>
      </jsdl:TotalResourceCount>
    </jsdl:Resources>
<!-- CPUs per node -->
      <jsdl:IndividualCPUCount>
        <jsdl:Exact>1.0</jsdl:Exact>
        <!-- Exact denotes the default value -->
        <jsdl:Range>
          <jsdl:LowerBound>1.0</jsdl:LowerBound>
          <jsdl:UpperBound>1.0</jsdl:UpperBound>
```

```
        </jsdl:Range>
      </jsdl:IndividualCPUCount>
<!-- either Nodes together with CPUs per node have to be specified  ↩
    or total CPUs, not all of them -->
<!-- total CPUs -->
      <jsdl:TotalCPUCount>
        <jsdl:Exact>1.0</jsdl:Exact>
        <!-- Exact denotes the default value -->
        <jsdl:Range>
          <jsdl:LowerBound>1.0</jsdl:LowerBound>
          <jsdl:UpperBound>1.0</jsdl:UpperBound>
        </jsdl:Range>
      </jsdl:TotalCPUCount>
<!-- Memory per node (bytes) -->
      <jsdl:IndividualPhysicalMemory>
        <jsdl:Exact>268435456</jsdl:Exact>
        <!-- Exact denotes the default value -->
        <jsdl:Range>
          <jsdl:LowerBound>1048576</jsdl:LowerBound>
          <jsdl:UpperBound>1073741824</jsdl:UpperBound>
        </jsdl:Range>
      </jsdl:IndividualPhysicalMemory>
```

2.6 Summary

This section shows in detail which configuration file entries control the connections between components, and where hostnames and ports can be set for each component.

2.6.1 The Connections Between the UNICORE Components

By default a UNICORE site would consist of the Gateway, a UNICORE/X server, a Registry, the Perl TSI, and the XUUDB. For defining a virtual site (vsite, includes the components needed to access a target system through UNICORE) named *SITE*, its host and port the following entries have to be made: - in `gateway/conf/connections.properties`

```
      SITE = https://VsiteHost:VsitePort
```

- `uas.config` in `unicorex/conf`

```
       uas.targetsystem.sitename=SITE
```

- `wsrflite.xml` in `unicorex/conf`

```
<property name="unicore.wsrflite.baseurl"
 value="https://gatewayHost:gatewayPort/SITE/services"/>
<!-- physical hostname (or IP address) and port -->
<property name="unicore.wsrflite.host" value="VsiteHost"/>
<property name="unicore.wsrflite.port" value="VsitePort"/>
```

CONFIGURATION OF AN XUUDB NEEDS SETTINGS IN

- `uas.config` in `unicorex/conf` (for using the `xuudb`)

```
xuudb_http_host=https://XuudbHost
xuudb_http_port=XuudbPort
xuudb_gcid=VsiteName
```

- `xuudb_client_conf` and `xuudb_server_conf` in `xuudb/conf` (for defining the `xuudb`)

```
xuudb_http_host=https://XuudbHost
xuudb_http_port=XuudbPort
```

CONFIGURATION OF LEGACY TSI NEEDS SETTINGS IN

- `tsi.properties` in `<TSI_Name>/conf`

```
tsi.njs_machine = VsiteHost
tsi.njs_port = Port-T_njs
tsi.my_port = Port-T_in
```

where `Port-T_njs` is the port unicorex is listening for connections from the tsi; `Port-T_in` is the port tsi is listening for connections from unicorex.

- `xnjs_legacy.xml` in `unicorex/conf`

```
<eng:Property name="CLASSICTSI.machine" value="TargetHost"/>
<eng:Property name="CLASSICTSI.port" value="Port-T_in"/>
<eng:Property name="CLASSICTSI.replyport" value="Port-T_njs ↩
    "/>
<eng:Property name="CLASSICTSI.TSI_LS"
              value="/opt/unicore/TSI_Name/tsi_ls"/>
```

3 Operation of a UNICORE Installation

Once the components have been installed and configured they can be started, tested, monitored and stopped.

3.1 Starting

Each of the services has a `bin/start.sh` script to start it. One exception is the tsi, which is started with `bin/start_tsi`. To start all components installed on one machine with just one command use the top-level `start.sh`. It will start all components whose directories are present in the installation directory.

3.2 Stopping

Analogously all components can be either stopped by their respective `bin/stop.sh` command or altogether by the top-level `stop.sh` command. The tsi can be stopped with `bin/kill_tsi`.

3.3 Monitoring

The Gateway, UNICORE/X, Registry and XUUDB servers have a `bin/status.sh` script which checks whether the processes are running. The TSI has an analogous `bin/find_pids` command. To check whether your services have started properly, check the `log/startup.log` of each service. In addition you can inspect the component's log file, e.g. `unicorex/logs/uas.log` or `/xuudb/logs/server.log`, for more detailed information.

Note
The Java based servers (i.e. all except the TSI) can be inspected throgh the standard Java management extensions (JMX), which allows to check memory usage and other relevant information through a program called "jconsole" which is part of the Java SDK. If the Java process is running on the same host as the jconsole, no further configuration is required. To enable remote access via JMX, you'll have to adapt `unicorex/conf/startup.properties` respectively `registry/conf/startup.propertes`. There, several system properties are defined that enable and configure remote access via JMX.

3.4 User management

To allow users access, they will need to be added to the XUUDB (or any other attribute source that you have configured).

For the XUUDB in our example scenario, assume we want to add a user with Unix login "hpc123"

adding a user to the XUUDB

```
ssh unicore@unicore.example.org
cd /opt/unicore/unicore6.4.2/xuudb
bin/admin.sh add CLUSTER /path/to/users/pemfile hpc123 user
```

The parameters to the admin "add" command are - the GCID (grid component ID) which was chosen as the parameter `uxGCID` in `configure.properties` - the path to the users certificate in pem format - the desired Unix login for the user (which has to exist on the target resource) - the role, in this case "user".

Note
It is not possible to use the Unix login "root", this is forbidden by the TSI for obvious security reasons.

3.5 Testing your installation

For testing the your installation, the simplest way is to use the UNICORE Command Line Client (ucc). Please refer to the UCC manual at http://unicore.eu/documentation/manuals/unicore6/-files/ucc/ucc-manual.html for further installation, configuration and usage instructions. For functionality testing you should use a user certificate and try the following command, which lists many of the capabilities of an installation

```
ucc system-info -l
```

To further test your installation, try

```
ucc connect
ucc run -v somejob.u
```

etc.

Note
UCC is very powerful if used with a certificate that has role "admin". See the chapter "Admin use of UCC" in the UCC manual.

4 Integration of another Target System

In case there is a second cluster or other compute system which should be made accessible through UNICORE it can be included by adding another unicorex service and a tsi. The xuudb, registry and the gateway can be shared as shown in Figure Figure 2.

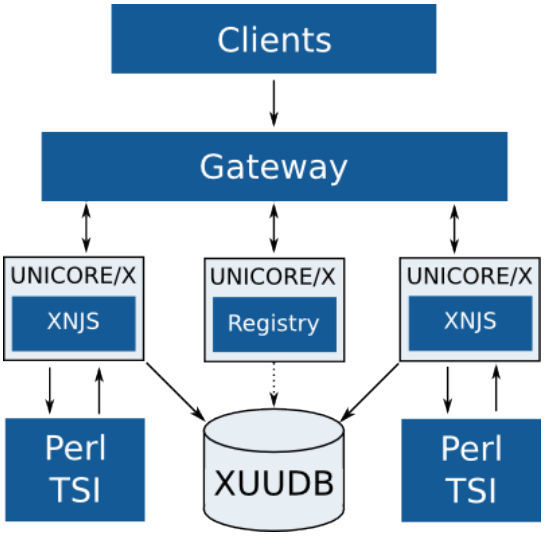

Figure 2: two target system configuration

4.1 Configuration of the UNICORE/X Service

The configuration of an additional UNICORE/X service is similar to the first UNICORE/X. In preparation for it you have to obtain a certificate for this service, a port number, a VSite name, and a GCID and make a copy of the `unicorex` directory. Staying with the example configuration used earlier, you now have on your server `unicore2.example.org` besides the `unicorex` directory a directory `secondunicorex`

Adaptations in secondunicorex/conf/uas.config Some basic settings have to be provided here:

```
#
# VSite name
#
uas.targetsystem.sitename=CLUSTER2
...
# Grid component ID used when querying XUUDB
# could be the same as for first unicorex if both sites share ↩
    the same userids
uas.security.attributes.XUUDB.xuudbGCID=CLUSTER2
```

Adaptations in secondunicorex/conf/wsrflite.xml Besides the security settings (server identity and truststore) the following settings have to be adapted:

```
<!-- provide the URL to access the service through the  ↩
    Gateway -->
   <property name="unicore.wsrflite.baseurl"
```

```
        value="https://unicore.example.org:8080/CLUSTER2/ ↩
            services"/>
<!-- physical hostname (or IP address) and port -->
<property name="unicore.wsrflite.host" value="unicore2. ↩
    example.org"/>
<property name="unicore.wsrflite.port" value="PORT2"/>
```

Changes in secondunicorex/conf/xnjs_legacy.xml Here basic information on the target system, its features and how to address it are to be specified.

```
<!-- properties -->
...
  <!-- Directory used for job directories (must be on the TSI ↩
       machine -->
  <eng:Property name="XNJS.filespace" value="/some/directory/ ↩
      on/the/target/system"/>
...
<eng:Property name="XNJS.tsiclass" value="de.fzj.unicore.xnjs ↩
    .legacy.LegacyTSI"/>
<!-- classic TSI properties -->
<eng:Property name="CLASSICTSI.machine" value="login.cluster2 ↩
    .example.org"/>
<!-- the port on the TSI machine where the TSI listens -->
<eng:Property name="CLASSICTSI.port" value="tsiPort"/>
<!-- the port on the XNJS machine which the TSI talks to -->
<eng:Property name="CLASSICTSI.replyport" value="tsiNjsPort ↩
    "/>
<!-- a user who may see all the jobs on the batch system (can ↩
     not be 'root'!) -->
<eng:Property name="CLASSICTSI.priveduser" value="unicore"/>
...
<!-- various command locations (on the TSI machine) -->
...  <!-- adapt paths to commands etc. here -->
```

Changes in secondunicorex/conf/simpleidb Here available resources, applications, and environments on the target system have to be specified. More detailed information is provided in the IDB documentation.

4.2 Configuration of Target System Interface

The TSI has to be installed and configured as described for the initial TSI above.

4.3 Addition of users to the XUUDB

In case the user IDs on the new site (`CLUSTER2`) are different from the user IDs on `CLUSTER`, the users have to be added to the XUUDB with gcid `CLUSTER2`.

4.4 Additions to the Gateway

In the current example the Gateway's auto-registration-option is not enabled therefore the new unicorex site has to be added to `gateway/conf/connections.properties`:

```
CLUSTER2 = https://unicore2.example.org:PORT2
```

5 Multi-Site Installation Options

By multi-site installation we mean that multiple UNICORE installations are working together, and users may use resources from more than one site --- the very basic idea of a Grid.

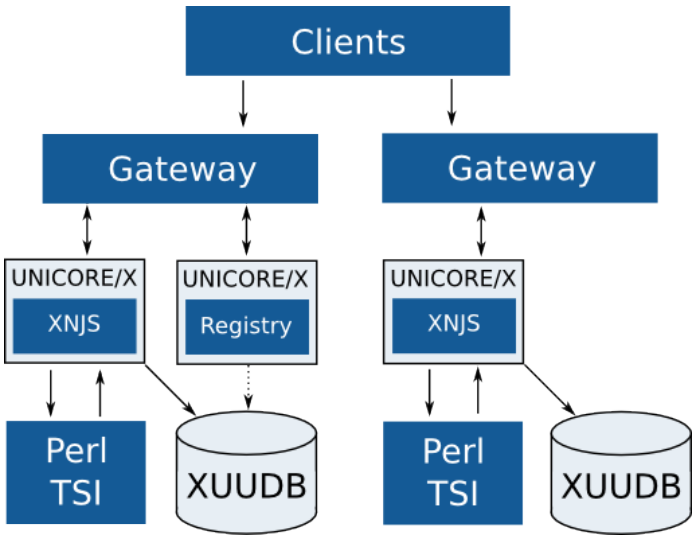

Figure 3: multi-site Grid

In fact there is nothing new in such a setting. Of course, the sites need to trust each other, so the CAs that issued the certificates of the gateways and servers needs to be trusted by the other sites. Some services in such a multi-site Grid are shared, most notably one would need one or more registries, and workflow services should be able to access all the execution sites.

5.1 Multiple Registries

For redundancy, multiple Registry servers may be present in a UNICORE Grid. Users can then choose to connect to one or more of these.

On the server side, each UNICORE/X server can register with more than one registry. This is done in the `unicorex/conf/uas.config` file

```
#
# URLs of registries to register with
#
# the one at our site
uas.externalregistry.url=https://unicore.example.org:8080/REGISTRY/ ↩
    services/Registry?res=default_registry
# another one
uas.externalregistry.url.2=https://othersite.org:8080/OTHER- ↩
    REGISTRY/services/Registry?res=default_registry
# more are possible...
# uas.externaregistry.url.3=...
```

6 Setting up the Workflow Services

For the execution of workflows the UNICORE installation needs to have the workflow services set up. The `unicore-workflow-6.3.2.tgz` consists of workflow and servorch service. It can be downloaded from `http://sourceforge.net/projects/unicore/files/1%20UNICORE%206%20`

Note
This document still describes the 6.3.2 version, which is due for an update quite soon. However no serious change in the installation is anticipated. If using a newer release, make sure to read the change log.

There are two servers that make up the workflow system

- the *Workflow* engine offers workflow submission and management services to clients. It deals with the high-level workflow execution, processing of control structures, conditions etc. It creates work assignments corresponding to single UNICORE jobs which are submitted to a Service orchestrator service

The *Service Orchestrator* (*Servorch* for short) deals with individual work assignments from the Workflow engine. It brokers the available UNICORE resources and finds sites matching the job requirements. The job is submitted and monitored until it has finished execution. Since these tasks are fairly resource-intensive, multiple Servorch servers may be present in the Grid to share the load.

6.1 Preparation

The workflow server components are the Workflow and Servorch (short for *Service Orchestrator*) servers, which are contained in the release bundle `unicore-workflow-6.3.2.tgz`.

The installation procedure is very similar to the core servers. Indeed both Workflow and Servorch servers share a lot of their configuration with the UNICORE/X, since they are "just" special purpose UNICORE/X servers.

If you haven't already done so, download the .tgz file, and un-tar it into a directory such as `/opt/unicore`.

In addition you should have a gateway installed to protect and an xuudb for authorizing users to use the workflow services. In the UNICORE Grid, there should be at least one UNICORE/X server which offers a storage factory service. This is needed for handling of workflow data.

You have several options to configure the components.

- (recommended) You may use the `configure.properties` file and set up the local configuration parameters there before you run the `configure.py` script.
- You may go through the configuration files and adapt them to your local setup, e.g. hostnames and ports, local paths etc. which should be straight forward. Specific configuration options are covered in the next chapters.

We prefer the first approach, since it will give you a working configuration, which you then can customise later.

So here is the recommended procedure

1. Edit the `configure.properties` file, and customize it according to your deployment plan
2. Install and configure workflow and servorch
3. Configure security (keystores, truststores) for the components
4. Start the servers

In the following the necessary steps are detailed. We are targeting a Grid layout as shown in Figure Figure 4. The Workflow server uses one or multiple Service Orchestrator services to take care of job brokering and execution. A shared registry is used: on the one hand for registering the workflow services, so users can find and use them, on the other hand for finding target systems for running jobs. An XUUDB controls which users have access to the workflow services.

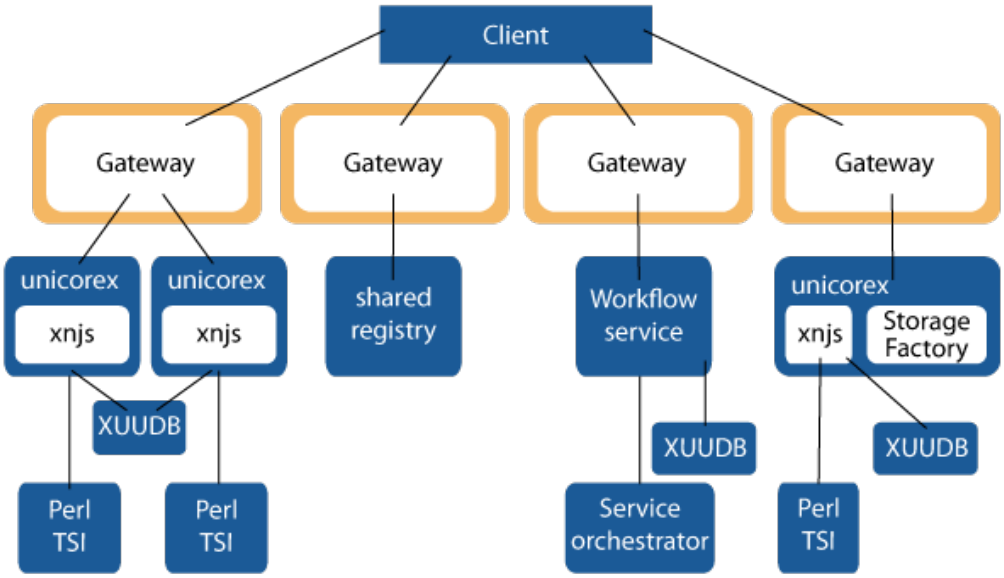

Figure 4: simple multi-site Grid with workflow services

configure.properties

```
# which components to install

workflow=true
servorch=true
#
# base directory, use "currentdir" to leave as-is
# (otherwise use an absolute path, such as "/opt/unicore/unicore- ↩
    workflow")
#
INSTALL_PATH=currentdir

#
# Gateway host and port
#
gwHost=unicore.example.org
gwPort=8080

#
# enable auto-registration of the servers with the gateway?
#
gwAutoRegistration=false

#
```

```
# WORKFLOW server
#

#server host
uasHost1=workflow.example.org

#server port
uasPort1=7700

#workflow vsite name
sitename1=WORKFLOW

# Port used by JMX
jmxPort1=9100

#
# SERVORCH server
#

#server host
uasHost2=workflow.example.org

#server port
uasPort2=7701

#servorch vsite name
sitename2=SERVORCH

# Port used by JMX
jmxPort2=9101

#
# settings for defining the external registry
#

#use external Registry?
useExternalRegistry=true
#either give fixed URL
urlExternalRegistry=https://unicore.example.org:8080/UNICORE/ ↩
    services/Registry?res=default_registry
#optionally, auto-discover?
findExternalRegistry=false

#
# XUUDB settings
#

# XUUDB server host
xuudbHost=unicore.example.host
```

```
# XUUDB server port
xuudbPort=34463

# Grid component ID to use
xuudbGCID=WORKFLOW
```

Copy your `configure.properties` somewhere safe, e.g. to your home directory, for later use and reference.

Now you're ready to configure the components using `configure.py`. This will insert your configuration values into the relevant files.

6.2 Workflow

The workflow service's configuration is contained in `workflow/conf/uas.config` and `workflow/conf/wsrflite.xml`. The main settings have already been done through the `configure.py`. The only thing left is the definition of security credential and truststore, which is done in `wsrflite.xml`.

wsrflite.xml

```
...
<!-- security configuration  -->

<property name="unicore.wsrflite.ssl" value="true"/>
<property name="unicore.wsrflite.ssl.clientauth" value="true"/>

<property name="unicore.wsrflite.ssl.keystore" value="/path/to/ ↩
    your/workflow/keystore.p12"/>
<property name="unicore.wsrflite.ssl.keypass" value="********"/>
<property name="unicore.wsrflite.ssl.keytype" value="pkcs12"/>

<property name="unicore.wsrflite.ssl.truststore" value="/path/to ↩
    /your/workflow/truststore.jks"/>
<property name="unicore.wsrflite.ssl.truststorepass" value ↩
    ="********"/>
<property name="unicore.wsrflite.ssl.truststoretype" value="jks ↩
    "/>
```

6.3 Service Orchestrator

For the service orchestrator there is also just the definition of the security credentials and the truststore left. These have to be set in `servorch/conf/wsrflite.xml`.

wsrflite.xml

```
<!-- security configuration  -->

<property name="unicore.wsrflite.ssl" value="true"/>
<property name="unicore.wsrflite.ssl.clientauth" value="true"/>

<property name="unicore.wsrflite.ssl.keystore" value="/path/to/ ↩
    your/servorch/keystore.p12"/>
<property name="unicore.wsrflite.ssl.keypass" value="********"/>
<property name="unicore.wsrflite.ssl.keytype" value="pkcs12"/>

<property name="unicore.wsrflite.ssl.truststore" value="/path/to ↩
    /your/servorch/truststore.jks"/>
<property name="unicore.wsrflite.ssl.truststorepass" value ↩
    ="********"/>
<property name="unicore.wsrflite.ssl.truststoretype" value="jks ↩
    "/>
```

6.4 Configuration changes to other components

6.4.1 Gateway

Edit `gateway/conf/connections.properties` and add the entries for the workflow servers:

```
WORKFLOW = https://workflow.example.org:7700
SERVORCH = https://workflow.example.org:7701
```

WORKFLOW and SERVORCH are the VSite names you defined in `configure.properties` or the services `uas.config` for attribute `uas.targetsystem.sitename`.

6.4.2 XUUDB

It is assumed that you have an XUUDB up and running. Then you have to add all users who are allowed to use the workflow service to the database with the GCID WORKFLOW as defined in the `configure.properties` file as shown above. The value can also be found in the `uas.config` file.

6.4.3 Adding a storage factory to a UNICORE/X server

At least one UNICORE/X server in the UNICORE Grid needs to provide filespace for workflow execution. On that UNICORE/X server, the StorageFactory service needs to be configured in `uas.config` and `wsrflite.xml`. For better load and data distribution multiple storage factories may be available.

uas.config

```
################################################
### StorageFactory service configuration ########
################################################

#
# Available storage types
#
uas.storagefactory.types=DEFAULT

#
# Configuration for the "DEFAULT" storage type
#
uas.storagefactory.DEFAULT.description=Default filesystem

# Base path. Denotes an absolute path on the TSI machine / cluster  ↩
    filesystem
# Must be world writable (chmod 1777)
uas.storagefactory.DEFAULT.path=/path/to/storagefactorybase

# If this is set to true, the directory corresponding to a storage  ↩
    instance will
# be deleted when the instance is destroyed.
# Defaults to "true"
uas.storagefactory.DEFAULT.cleanup=true

# this can be used to override the general list of SMS filetransfer ↩
     protocols
uas.storagefactory.DEFAULT.protocols=BFT RBYTEIO SBYTEIO
```

wsrflite.xml The storage factory service must be defined and enabled in the service section of the file.

```
    <!-- the storage factory service -->
   <service name="StorageFactory" wsrf="true" persistent="true"  ↩
       enabled="true">
     <interface class="de.fzj.unicore.uas.StorageFactory" />
     <implementation class="de.fzj.unicore.uas.impl.sms. ↩
         StorageFactoryHomeImpl"/>
   </service>
```

After enabling, the "ucc system-info -l" must show the new storage factory service.

7 Glossary

USite	A USite (= UNICORE site) is the UNICORE term for a set of Vsites in a common administrative domain. Each USite has a single Gateway
VSite	A VSite (= virtual site) is the UNICORE term for a Grid node, such as a compute resource. It corresponds to a UNICORE/X server.
XUUDB	The UNICORE user database maps a user certificate to attributes such as the user account and the user role. These attributes are used to control access to UNICORE resources.
UNICORE/X	UNICORE/X is the central server component. It offers Web Services, interfaces to the XUUDB or other attribute sources, and provides access control. Using the XNJS library, it executes and manages jobs. Jobs are passed to the TSI for execution.
XNJS	an internal library providing the functionality underlying many of the Web Service interfaces of UNICORE. For example, the interface to the TSI or the file staging capabilities are provided through the XNJS. The XNJS is also used in the Workflow engine for handling workflows.
UAS	the set of basic UNICORE Web Service interfaces (TargetSystemFactory, StorageManagement, etc) are sometimes referred to as UNICORE Atomic Services.
IDB	The IDB is a file which describes system specific values like paths to executables, amount of physical memory and CPUs, etc. It is used by UNICORE/X to translate an abstract job into a system specific job.
Global Registry	The Global Registry is used by clients (and services) to find available services in a UNICORE Grid. UNICORE/X servers register services with one or more Global Registries.
Gateway	A site's Gateway to the public network. It is designed to be the only component which has an open port in the firewall, and it can serve multiple UNICORE components within the same firewall.
TSI	The Target System Interface executes a system specific job on the target system on behalf of a user. It communicates with the local resource management system and allows access to the local filesystems.
GCID	Used to differentiate different sets of entries in the XUUDB. Thus it is possible to share a XUUDB between several servers where each server uses a different GCID. To share XUUDB entries between multiple servers, give these components the same GCID.

OGSA	The Open Grid Services Architecture
OGSA-BES	OGSA Basic Execution Service
JSDL	Job Submission Description Language
URI	Uniform Resource Identifier
HTTP	Hypertext transfer protocol
XML	eXtensible Markup Language
SSL	Secure Socket Layer
CA	Certification Authority