Jül-4319

Jülich Supercomputing Centre (JSC)

UNICORE 6 – Recent and Future Advancements

Achim Streit, Piotr Bala, Alexander Beck-Ratzka, Krzysztof Benedyczak, Sandra Bergmann, Rebecca Breu, Jason Milad Daivandy, Bastian Demuth, Anastasia Eifer, André Giesler, Björn Hagemeier, Sonja Holl, Valentina Huber, Nadine Lamla, Daniel Mallmann, Ahmed Shiraz Memon, Mohammad Shahbaz Memon, Michael Rambadt, Morris Riedel, Mathilde Romberg, Bernd Schuller, Tobias Schlauch, Andreas Schreiber, Thomas Soddemann, Wolfgang Ziegler

JÜLICH
FORSCHUNGSZENTRUM

UNICORE 6 – Recent and Future Advancements

Achim Streit, Piotr Bala, Alexander Beck-Ratzka, Krzysztof Benedyczak, Sandra Bergmann, Rebecca Breu, Jason Milad Daivandy, Bastian Demuth, Anastasia Eifer, André Giesler, Björn Hagemeier, Sonja Holl, Valentina Huber, Nadine Lamla, Daniel Mallmann, Ahmed Shiraz Memon, Mohammad Shahbaz Memon, Michael Rambadt, Morris Riedel, Mathilde Romberg, Bernd Schuller, Tobias Schlauch, Andreas Schreiber, Thomas Soddemann, Wolfgang Ziegler

Berichte des Forschungszentrums Jülich; 4319
ISSN 0944-2952
Jülich Supercomputing Centre (JSC) Jül-4319

Vollständig frei verfügbar im Internet auf dem Jülicher Open Access Server (JUWEL)
unter http://www.fz-juelich.de/zb/juwel

Zu beziehen durch: Forschungszentrum Jülich GmbH, Zentralbibliothek, Verlag
D-52425 Jülich · Bundesrepublik Deutschland

☎ 02461/61-6123 · Telefax: 02461/61-6103 · e-mail: zb-publikation@fz-juelich.de

FORSCHUNGSZENTRUM JÜLICH GmbH
Jülich Supercomputing Centre
D-52425 Jülich, Tel. +49 2461 61-6402

UNICORE 6 – Recent and Future Advancements

Achim Streit, Piotr Bala, Alexander Beck-Ratzka, Krzysztof Benedyczak, Sandra Bergmann, Rebecca Breu, Jason Milad Daivandy, Bastian Demuth, Anastasia Eifer, André Giesler, Björn Hagemeier, Sonja Holl, Valentina Huber, Nadine Lamla, Daniel Mallmann, Ahmed Shiraz Memon, Mohammad Shahbaz Memon, Michael Rambadt, Morris Riedel, Mathilde Romberg, Bernd Schuller, Tobias Schlauch, Andreas Schreiber, Thomas Soddemann, Wolfgang Ziegler

February 2010

Authors

Achim Streit, Sandra Bergmann, Rebecca Breu, Jason Milad Daivandy, Bastian Demuth, André Giesler, Björn Hagemeier, Sonja Holl, Valentina Huber, Nadine Lamla, Daniel Mallmann, Ahmed Shiraz Memon, Mohammad Shahbaz Memon, Michael Rambadt, Morris Riedel, Mathilde Romberg, Bernd Schuller

Jülich Supercomputing Centre (JSC)
Forschungszentrum Jülich GmbH
Jülich, Germany

Piotr Bala, Krzysztof Benedyczak

Interdisciplinary Center for Mathematical and Computational Modeling (ICM)
Warsaw University
Warsaw, Poland

Thomas Soddemann, Wolfgang Ziegler

Fraunhofer Institute for Algorithms and Scientific Computing (SCAI)
Sankt Augustin, Germany

Anastasia Eifer, Tobias Schlauch, Andreas Schreiber

Simulation and Software Technology,
German Aerospace Center (DLR)
Cologne, Germany

Alexander Beck-Ratzka

Max Planck Institute for Gravitational Physics
Potsdam, Germany

Contents

Chapter 1

Introduction

In the last three years activities in Grid computing have changed; in particular in Europe the focus moved from pure research-oriented work on concepts, architectures, interfaces, and protocols towards activities driven by the usage of Grid technologies in day-to-day operation of e-infrastructure and in application-driven use cases. This change is also reflected in the UNICORE activities [1]. The basic components and services have been established, and now the focus is increasingly on enhancement with higher level services, integration of upcoming standards, deployment in e-infrastructures, setup of interoperability use cases and integration of applications.

The development of UNICORE started back more than 10 years ago, when in 1996 users, supercomputer centres and vendors were discussing "what prevents the efficient use of distributed supercomputers?". The result of this discussion was a consensus which still guides UNICORE today: seamless, secure and intuitive access to distributed resources.

Since the end of 2002 continuous development of UNICORE took place in several EU-funded projects, with the subsequent broadening of the UNICORE community to participants from across Europe. In 2004 the UNICORE software became open source and since then UNICORE is developed within the open source developer community. Publishing UNICORE as open source under BSD license has promoted a major uptake in the community with contributions from multiple organisations. Today the developer community includes developers from Germany, Poland, Italy, UK, Russia and other countries.

The structure of the paper is as follows. In Section 2 the architecture of UNICORE 6 as well as implemented standards are described, while Section 3 focusses on its clients. Section 4 covers recent developments and advancements of UNICORE 6, while in section 5 an outlook on future planned developments is given. The paper closes with a conclusion.

Chapter 2

Architecture and Standards

The architecture of UNICORE 6 is three-layered in client layer, service layer and system layer as shown in Fig. 2.1.

2.1 Client Layer

On the top layer a variety of clients are available to the users, ranging from graphical clients such as the Eclipse-based URC, a command-line interface named UCC to a programming API named HiLA. For more details on these clients see Section 3. For a tight integration of various types of applications, the GridBean concept [2] was invented, which offers an API to easily implemented graphical client extensions and connect them with UNICORE 6's core functionalities (cf. Section 4.6 for a few GridBean examples).

2.2 Service Layer

The middle layer comprises all services and components of the UNICORE Service-Oriented Architecture (SOA). Figure 2.1 shows three sets of services, the left and right one containing services at a single site while the middle shows the central services, such as the central registry, the workflow services and the information service, which serve all sites and users in a UNICORE Grid. The Gateway component [3] acts as the entry point to a UNICORE site and performs the authentication of all incoming requests. It is used for both the central services and the single site services. The XNJS component [4] is the job management and execution engine of UNICORE 6. It performs the job incarnation, namely the mapping of the abstract job description to the concrete job description for a specific resource according to the rules stored in the IDB (Incarnation Database). The functionality of the XNJS is accessible via two service interfaces in UNICORE 6's WS-RF hosting environment. UNICORE 6's proprietary interface is called UAS (UNICORE Atomic Services) [2] and offers the full functionality to higher level services, clients and users. In addition to the UAS, a standardised set of interfaces based on open, common standards is available (depicted as "OGSA-*" in Figure 2.1).

For authorisation of users the XNJS uses the XUUDB user database to perform the mapping from X.509 certificates to the actual users' logins. The

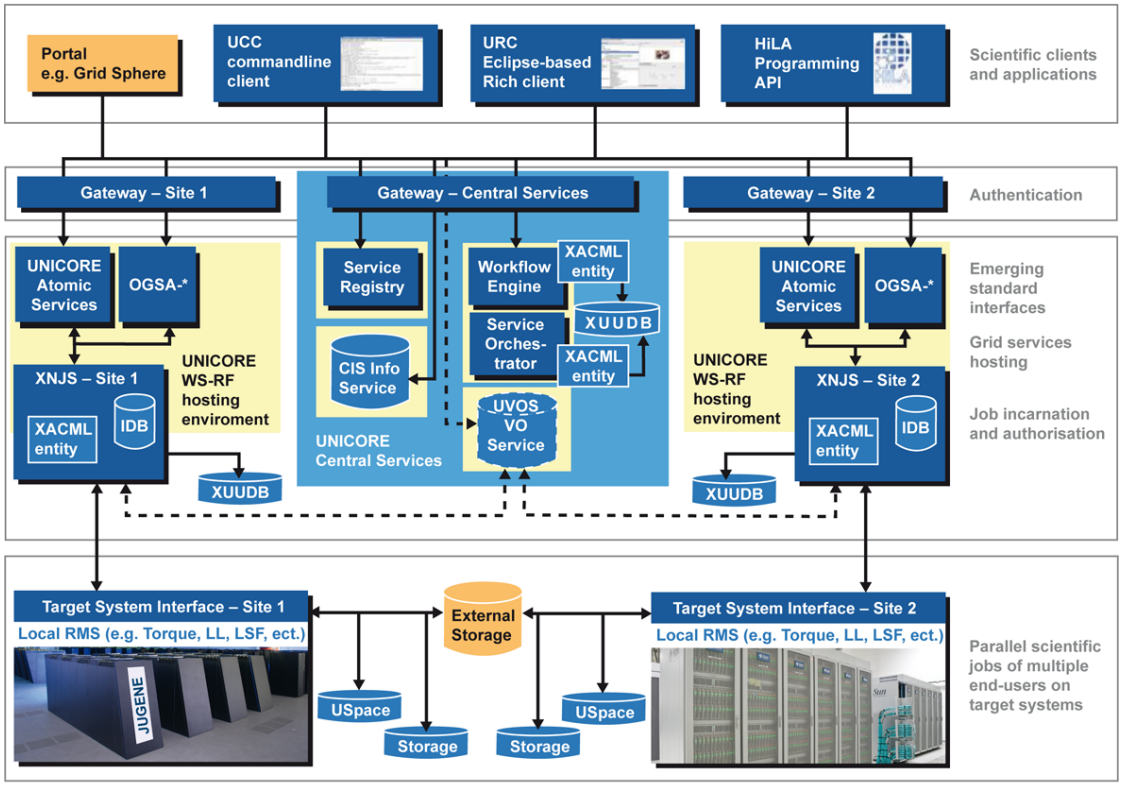

Figure 2.1: UNICORE 6 architecture

XUUDB component is a Web service in itself, which allows it to be used from multiple UNICORE installations, e.g. within the same computing centre. Like in many service-oriented environments, a central service registry is available, where the different services can register once they are started. The central service registry is necessary to build-up and operate a distributed UNICORE infrastructure. This service registry is contacted by the clients in order to "connect to the Grid". An infrastructure may have multiple central registries. From the beginning, workflow support is of major importance for UNICORE. The two layered design with a separation of the workflow engine and the service orchestrator was primarily done for better scalability, but also offers the possibility to plug-in domain-specific workflow languages and workflow engines. The workflow engine originates from the EU-project Chemomentum [5]. Besides simple job-chains, while- and for-loops, workflow variables and conditional execution are supported. The service orchestrator deals with brokering the execution, monitoring of the workflow and its respective parts as well as provides call-back mechanisms to the workflow engine. The resource brokering is performed via pluggable strategies. More details are found in [5]. The workflow capabilities are offered to the users on the client layer via the URC. Furthermore, the definition, submission, monitoring and control of workflows is also possible from the UCC. A Tracing Service collects runtime information from the workflow system, and allows generating performance metrics. End users can use the tracer service to visualise the execution of a complex workflow from within the URC.

2.3 System Layer

On the bottom layer the TSI (Target System Interface) component is the interface between UNICORE and the individual resource management/batch system and operating system of a Grid resource. In the TSI component the abstracted commands from the Grid layer are translated to system-specific commands, e.g. in the case of job submission, the specific commands like *llsubmit* or *qsub* of the resource management system are called. The TSI component is performing the proper setting of users' UID and invocation of his/her environment. Therefore the TSI component needs to be executed with root privileges. All other UNICORE 6 components at a site are executed under a standard user account, preferably a dedicated, UNICORE-related account. The TSI is available for a variety of commonly used batch systems such as Torque, LoadLeveler, LSF, SLURM, OpenCCS, etc. The USpace is UNICORE's job directory. A separate directory exists for every job, where the XNJS and TSI store all input data and where all output including stdout and stderr is written to. The TSI also allows access to local filesystems such as the user's Home directory. These are called UNICORE Storages and can be used in stage-in and stage-out operations.

2.4 Standards

Common open standards are of major importance to realise interoperable worldwide Grid infrastructures in order to satisfy scientists that require resources in more than one Grid. In order to increase the interoperability with Grid infrastructures that deploy other middleware solutions, several standards from the Open Grid Forum and OASIS are used in UNICORE 6 in various domains (cf. to the boxes on the right in Figure 2.1). A full Web services stack based on WS-RF 1.2, SOAP, and WS-I is implemented to build UNICORE 6's service-oriented architecture.

In security, full X.509 certificates are used as base line, while the access control is based on XACML policies. A support for SAML-based VOMS (virtual organisation management service) [6] is available as well as support for proxy certificates. Apart from standards in the area of security, standards play also a significant role in the area of information. In that context, the information service of UNICORE 6 named as CIS (Common Information Service) [7] is based on the GLUE 2.0 [8] information model and thus allows for a standardised way of exposing computational resources. It gathers both static and dynamic information from all connected XNJSs, which are then displayed either in raw XML or human-readable text form. As longitude and latitude information is also stored, a Google maps view illustrates a geographical representation of the Grid infrastructure. For job management, OGSA-BES [9] and several profiles (i.e. HPC-BP [10], HPC-FSP [11]) are used for the creation, monitoring and control of jobs, including data-staging, whilst job definition is compliant with the JSDL standard [12] and its profiles. In the area of data management and transfer, OGSA-ByteIO can be used for data transfer, both for site-to-site and client-to-site transfers [13]. For a transfer of data from and to external storage,

the GridFTP [14] transfer protocol can be used as an alternative to the default HTTP based mechanism. On the system layer a TSI version for the DRMAA standard [15] is available enabling a standardised interface between the TSI and the batch system.

Chapter 3

Using UNICORE

In the following we will describe UNICORE's core clients; namely the Eclipse-based graphical URC, the command-line client UCC as well as the programming API HiLA.

3.1 Eclipse-based URC

Since the beginning, UNICORE has offered graphical clients realising UNICORE's baseline slogan by providing seamless, secure and intuitive access to Grid resources. Today the Eclipse-based UNICORE Rich Client (URC) [16], shown in Figure 3.1, constitutes the most complete implementation of this idea.

Basing the major UNICORE client on the Eclipse rich client platform [17] comes with several benefits. First of all, Eclipse is widely known and commonly used due to its well designed and flexible graphical interfaces. This lowers the entry barrier for new users, as many of them are already familiar with the tool. Furthermore, although being written in Java, an Eclipse-based application contains some platform specific code. Through this approach, it looks just like a native application with a smoother integration into different platforms. Finally, the Eclipse platform has a very sophisticated plug-in mechanism: every software component in an Eclipse-based client is a plug-in, each of which adding a well-defined range of functions. Plug-ins interact with each other in various ways and almost every plug-in provides programming interfaces which can be used to extend its functionality and outer appearance. Following this paradigm, the URC is extremely extensible. For instance, integration of new Grid services or scientific applications is already accounted for in its design.

This client targets a wide range of users with varying Grid and IT experience. It provides a useful graphical view of the Grid, which can be filtered in order to find specific resources, services or files. As one of its main tasks, it can be used to submit computational jobs to Grid resources. To this end, small software packages provide tailored graphical user interfaces for many scientific applications available on the Grid. The same packages are responsible for visualising the output data of scientific simulations once the jobs have been executed and output files have been downloaded to the client machine. Detailed resource requirements for jobs (e.g. required number of CPUs, amount of RAM) can be

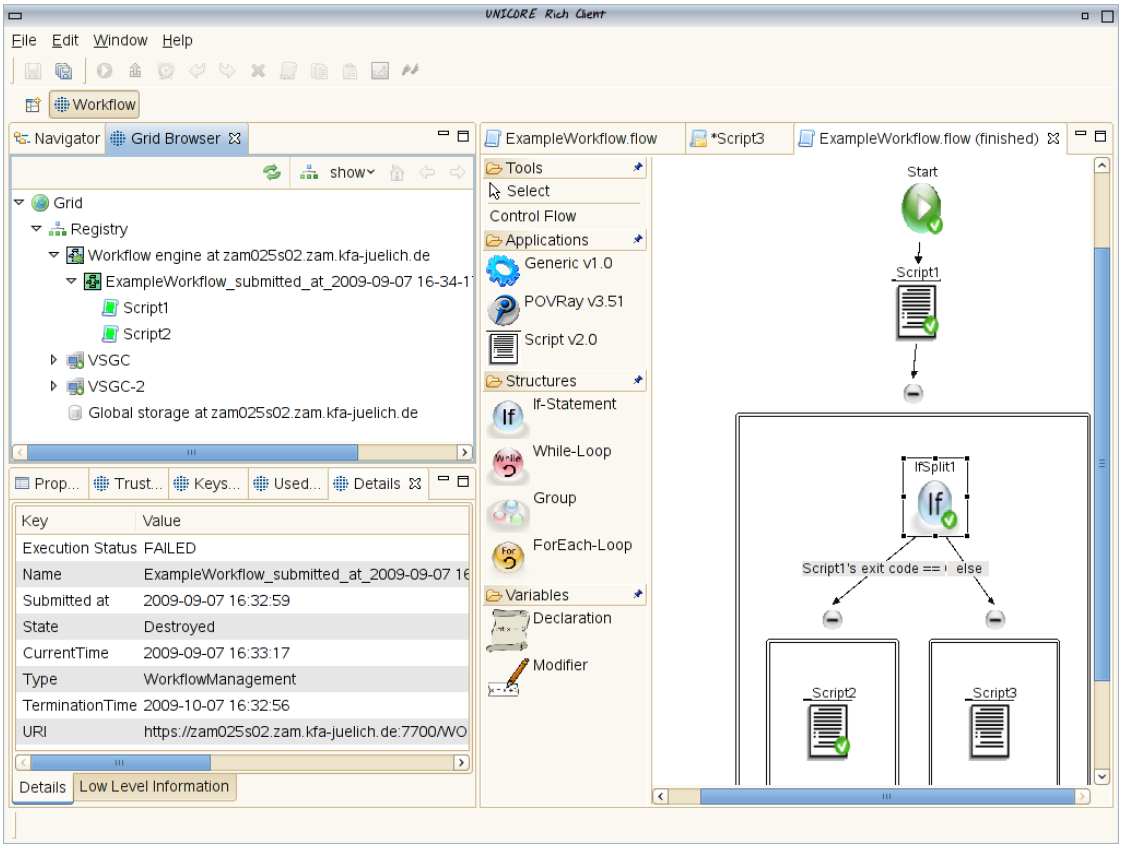

Figure 3.1: The UNICORE Rich Client (URC)

specified. Users are enabled to design complex scientific workflows that combine several applications to automate the completion of difficult tasks. To this end, a fully-fledged workflow editor is provided. It allows for graphical programming where building blocks like loops or if-statements can be arranged with just a few mouse clicks. As security and access control are essential aspects in distributed computing, dedicated panels deal with setting up security options so that users can specify whom they trust and how to identify themselves on the Grid. Experienced users can perform various administrative tasks on the Grid by accessing specific parts of the user interface.

3.2 Command-line client UCC

The UNICORE Commandline Client (UCC) [18] is a versatile command-line tool that allows users to access all features of the UNICORE service layer in a shell or scripting environment (cf. Figure 3.2 for an overview of available commands). It allows to run jobs, monitor their status and retrieve generated output, both in single job mode or in a powerful and flexible batch mode for multiple jobs; workflows can be submitted and controlled as well. Additionally, UCC includes several data management functions. Remote storages can be listed and files can be transferred from local to remote and vice versa as well as from server to server. UCC can be used for administrative purposes as well, for example to list jobs, or to perform some clean-up. An important feature of UCC is its extensibility. New commands can easily be added, and the `run-groovy`

command allows the execution of scripts written in the Groovy programming language which can access a variety of UNICORE-internal functions for more flexibility. A dedicated UCC mode for the popular Emacs editor is also available.

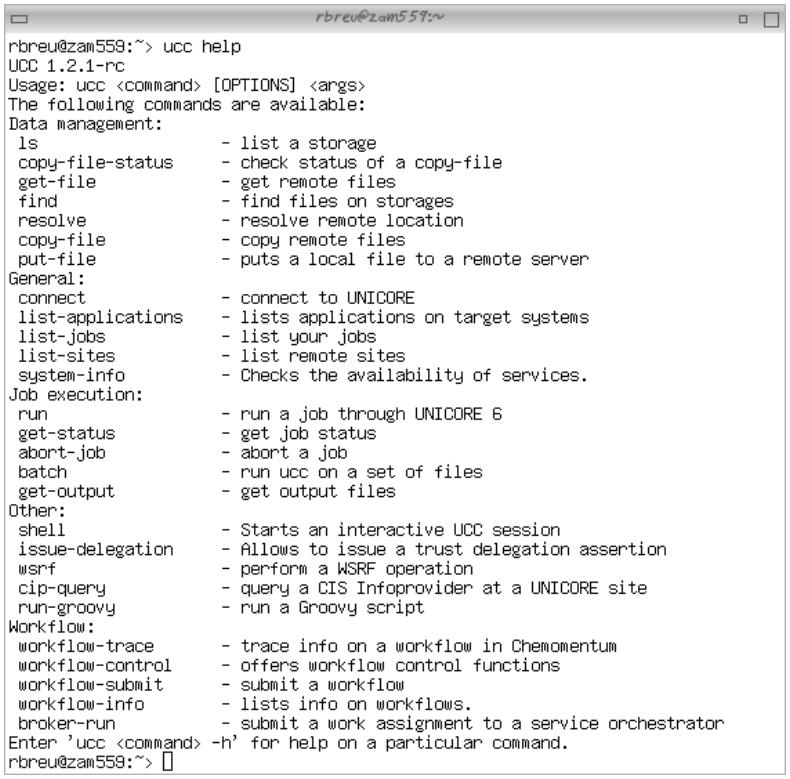

Figure 3.2: Available Commands of the UNICORE Commandline Client (UCC)

3.3 Programming API HiLA

HiLA is a High Level API for Grid Applications [19], which allows simple development of clients with just a few lines of code for otherwise complex functionality. It provides a single interface with multiple implementations for UNICORE 5, UNICORE 6 and OGSA-BES. HiLA was mainly developed in the EU-funded DEISA [20] and A-WARE [21] projects. It is used to integrate UNICORE 6 access into DESHL [22] (including SAGA [23] support) as part of the DEISA portal, and to connect GAT [43] with UNICORE 6 (cf. Section 4.5. The nature of the HiLA API leads to a concise coding toolkit for building collective tier Grid services and client interfaces. Following the UNICORE principle of seamlessness, the design of the API models the Grid through an object-oriented façade, presenting abstract representations of the underlying resources. Importantly, this includes encapsulating security configuration behind well defined interfaces, further enhancing the API.

In HiLA resources of the Grid are named following a URI naming scheme, e.g. `unicore6://sites/GROW/tasks/910c9b56-d97-46af37` for a job or `unicore6://sites/FZJ_JUGGLE/storages/home` for the user's home directory. The object navigation is based on a container/item model, navigation of locatable resources

is done generically. Types of objects referenced by locations can be inferred and thus it is possible to call object specific operations such as `status()` or `getOutcomeFiles()` on jobs. HiLA is currently evolving further to allow it to operate concurrently over multiple resources to perform powerful collective data management and monitoring.

Listing 3.1: The HILA API

```
Location l = new Location(
    "unicore6://sites/GROW/tasks/910c9b56-d97-46af37");
Task t = HiLAFactory.getInstance().locate(l);
assertTrue(TaskStatus.RUNNING, t.status());
List<File> fl = t.getOutcomeFiles();
```

As can be seen in Listing 3.1, the resources of the Grid are named following a URI naming scheme, another example for a storage resource representing the user's home directory would be `unicore6://sites/FZJ_JUGGLE/storages/home`. The Listing also shows that the object navigation is based on a container/item model, navigation of locatable resources is done generically. Types of objects referenced by locations can be inferred and thus it is possible to call object specific operations such as `status()` or `getOutcomeFiles()` on jobs. HiLA is currently evolving further to allow it to operate concurrently over multiple resources to perform powerful collective data management and monitoring.

Chapter 4

Recent Developments

In the following section, recent developments are reported one by one, that enhance the functionalities of UNICORE 6. They were done in the context of various projects and by different teams.

4.1 Virtual Organisation Management

The UNICORE Virtual Organizations System (UVOS) [25] was created in course of the EU-funded Chemomentum project [5]. However due to its openness and standards-compliance UVOS can be used with other Grid middlewares, too. UVOS communicates with other Grid components using standard protocols and provides tools and features which will make it useful in both OGSA (so SOA) and WWW environments.

A key element of UVOS is the central server which acts both as authentication service and attribute authority. It is used by two kinds of clients: consumers and management clients. Consumers do not modify the UVOS content but query it. The modification of the VO data can be performed using management clients: graphical and command line. The UVOS system is highly portable and all UVOS server operations are available via the web services interface. The UVOS consumers use open standard SAML 2.0 as a communication protocol and the server implements core SAML specification together with additional profiles:

- XACML Attribute Profile [26]
- SAML Attribute Query Deployment Profile for X.509 Subjects [27]
- SAML Attribute Self-Query Deployment Profile for X.509 Subjects [27]
- OGSA Attribute Exchange Profile Version 1.2 [28]

UVOS organises VO members within a hierarchical group structure. Top level groups of this structure are called virtual organisations. The virtual organisations are however not different than other groups therefore group membership is inherited in UVOS: members of a subgroup automatically become members of the parent group. Group members may have assigned a set of attributes. A single entity can possess multiple representations, for example in different formats.

These equivalent incarnations of the same entity are called identities and are usually invisible for an outside user. Every entity has a unique label and one or more tokens that represent it. Tokens must be in one of the supported formats, which currently are: a full X.509 certificate, an X.500 distinguished name, or an e-mail address with an password used for authentication. A token along with its type is called an identity. All of the identities that compose an entity sharing the same characteristics (attributes, group membership, permissions, etc.). UVOS internally uses entities.

UVOS provides two management clients: the command line client (UVOS CLC) and VO Manager. The UVOS VO Manager is a powerful GUI application based on the Eclipse Rich Client platform like the URC and offers an intuitive interface to create and handle users related information.

4.2 Shibboleth Integration in URC

This section describes the approach how Shibboleth [29] and advanced authorisation technologies such as SAML2 [30] and XACML [31] are combined to achieve Single Sign-On (SSO) among multiple UNICORE 6 Grid sites.

Shibboleth is a standards-based identity management system and an open source implementation of the SAML2 Web SSO profile [26] aiming to fulfill single sign-on across or within organisational boundaries. It consists of three main components namely the Service Provider (SP), the Identity Provider (IdP) and the Where-Are-You-From (WAYF) or Discovery Service (DS) which requires user to authenticate and authorise while using a Web browser. This creates a mismatch with the UNICORE 6 security model, as in UNICORE 6 the client usually is a non-Web browser and uses X.509 certificates for user authentication and SOAP for message exchange, which is not possible by using Shibboleth in a straight forward manner. Therefore a GridShib-Certificate Authority (CA) [32] is endorsed as a SP, thus enabling the Shibboleth clients (Web browser) to retrieve X.509 based Short Lived Grid Credentials (SLC) with embedded SAML2 assertions on authentication with their IdP; the embedded SAML assertion manifests the subject's attributes.

Nonetheless, the GridShib-CA does not solve the problem of using Shibboleth with UNICORE 6 as it still restricts the user to use a Web browser to fetch the user's SLC. Therefore a client API was developed by extending the architecture defined in [33], the component is generalised to allow any non-Web browser client to communicate with a GridShib-CA, as shown in Figure 4.1.

The GUI prototype is developed as an Eclipse-based URC plug-in in order to facilitate the client to fetch her SLC by inputting the Gridshib-CA URL, username, and password. The request (besides fetching SLC) from client to the UNICORE 6 server is bipartite; in the first step it extracts the SAML assertion and public key information from the SLC and then sends the request in combination with the job submission as a SOAP message with all the identity information in its headers. On the server side, before the invocation of the Target System service, the SAML assertion is extracted from the SOAP header and processed, which requires checking the attributes against UNICORE 6 XACML policies and XUUDB entries. This process leads to the decision whether the service access request is denied or granted. The UNICORE 6 authorisation and authentication framework is inherently based on pluggable mechanism by

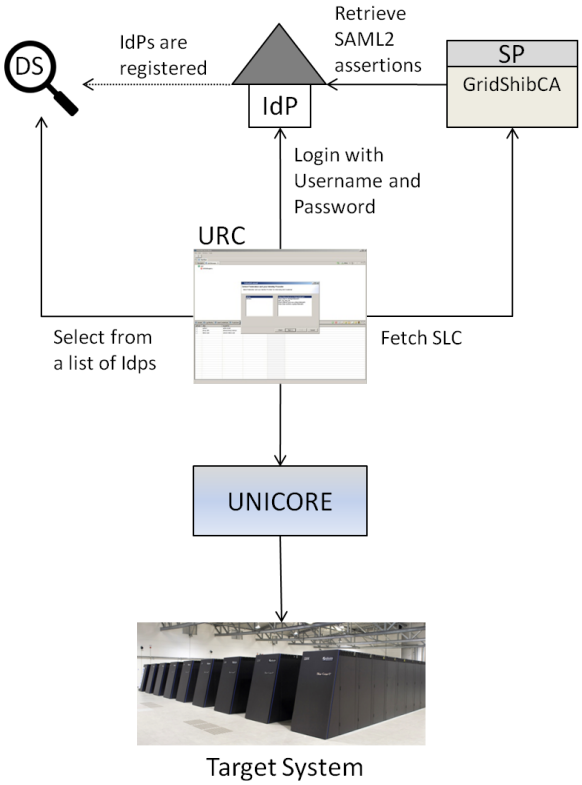

Figure 4.1: Architecture of bridging Shibboleth and UNICORE 6

using handlers, some of which provide capabilities of handling SAML2 assertions namely `SAMLInHandler` and `SAMLOutHandler` as described in [6]. Bridging Shibboleth and the UNICORE 6 security model enables authentication and authorisation interoperability with all other middlewares supporting Shibboleth and GridShib-CA.

4.3 Interactive Access with X.509-based SSH

The key component and warrantor for security on the server side is a SOCKS 5 proxy [34]. It only routes traffic to the standard SSH server, once the connection has been authenticated using the XUUDB. Thereby X.509-based authentication with the security level of UNICORE 6 is enabled. After successful authentication, all traffic from and to a designated SSH Server is routed (delegated) by the SOCKS 5 proxy component.

On the client side, the URC realises the interaction with the SOCKS Proxy and establishes the actual SSH connection. For this purpose, the URC is extended with Eclipse plug-ins, which provide terminal emulation and SSH communication functionality. The URC is extended with Eclipse plug-ins, which provide a terminal emulation and SSH communication functionality. The terminal plug-in is based on the terminal emulation software from the gEclipse project [35]. It offers a standard VT100 emulation, which allows the display of graphical output from programs using the curses library.

The plug-ins have been designed in an extensible way, so that different communication providers (apart from the X.509-enabled and SOCKS 5 based ap-

proach) can be plugged in. These communication providers are likewise implemented as Eclipse plug-ins which encapsulate the connection and communication logic and extend the functionality of the terminal plug-in. It is important to note that the X.509 based SOCKS communication provider connects to the SOCKS 5 proxy server with the users certificate, which is available within the keystore of the URC, and provides terminal access to a specific target site without any further password input.

Currently, three different communication providers have been implemented: the X.509 enabled and SOCKS 5 based SSH solution, GSI-SSH known from the Globus Toolkit [36], and standard plain SSH with password. The latter plug-in could in principle also be used without UNICORE 6 in any other Eclipse framework.

4.4 DataFinder Integration

The amount of data handled by applications is ever growing and data retrieval for later use becomes a serious problem. The level of support for data management is very different in current systems for resource management (UNICORE 6, Globus Toolkit, gLite) and distributed data management (dCache [37], Storage Resource Broker [38], Integrated Rule-Oriented Data System [39], Chemomentum [5], and DataFinder [40]).

Most of the considered systems support a kind of storage virtualisation. Additionally, some systems provide metadata management functionalities allowing simple annotations on data objects which are achieved using a metadata catalog. Unfortunately, further means regarding data organisation are missing. An exception from that rule are the data-related services developed in the Chemomentum project [5] and the DataFinder [40] system which are providing sophisticated means for data organisation. Here it has been decided to use DataFinder as basis for a dedicated data management system as it offers required means for data organisation and its productive use in various scenarios at the German Aerospace Centre provides a good basis demonstrating and investigating the system.

The developed data management system [41] delivers logical organisation of data objects and abstracts common data management concepts hiding the specifics of storage systems. On this basis advanced means for data organisation through metadata management functionalities and support of custom data models are provided. Moreover, the integration with UNICORE 6 is achieved by the data management client API which has been integrated in the UNICORE Rich Client. The developed system and its compatibility with DataFinder have been successfully validated in the project AeroGrid [42].

4.5 Java-GAT

Making applications Grid-aware requires the knowledge of the API of different Grid middlewares. The complexity of these systems, both in terms of underlying technology and functionality, remains high. The Grid Application Toolkit (GAT) [43], a high-level API for accessing Grid services, provides application developers with a unified, simple and middleware-independent interface to the

Grid. It allows developers to easily include Grid functionality in application codes. This is performed by writing plug-ins for the different Grid middlewares.

The GAT architecture is divided to two parts: GAT engine and GAT adaptors (Figure 4.2). The GAT-API delivers the commands to the GAT engine which selects a GAT adaptor. An adaptor converts a file copy or job submission statement written in the GAT-API into a corresponding statement of a Grid middleware. GAT uses the first adaptor which does not fail but it is also possible to force usage of selected adaptors.

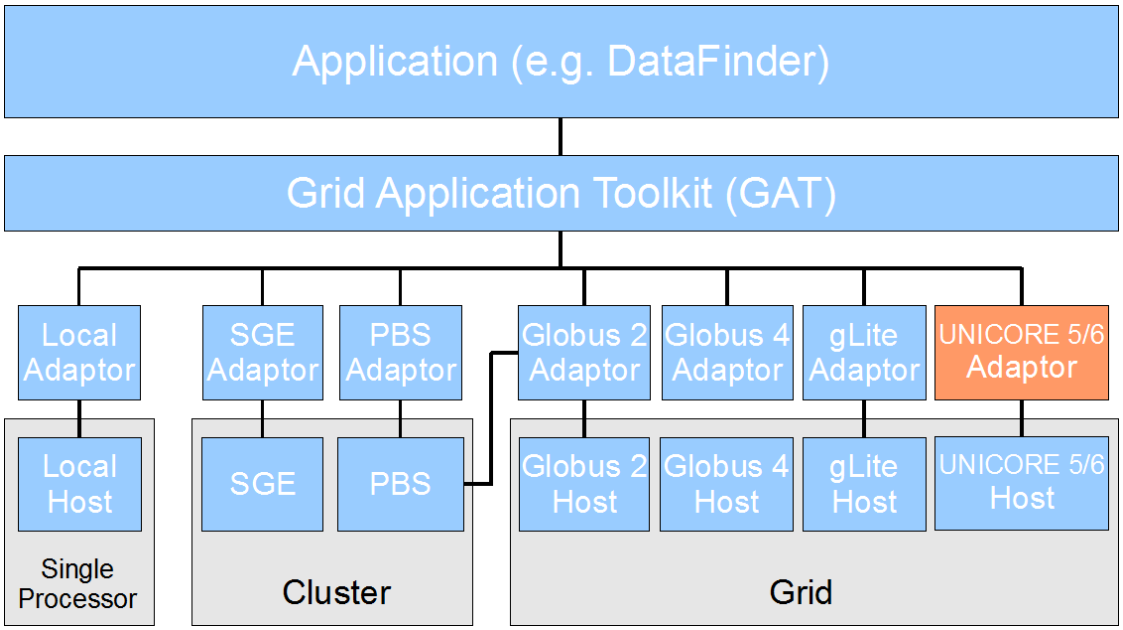

Figure 4.2: JavaGAT software architecture.

GAT has first been implemented in C within the European project GridLab [44]; the development of the Java version of GAT (JavaGAT) has also been started during the GridLab project. The C Implementation comes with a C++ and a Python wrapper. While C-GAT is out of support, JavaGAT is still maintained by the Free University of Amsterdam, and JavaGAT is used within the project AeroGrid [42]. One of the major advantages of JavaGAT compared to C-GAT is that JavaGAT enables the Grid access without an additional installation of a Grid middleware.

The UNICORE adaptor for JavaGAT [45] has been implemented recently within the scope of the D-Grid project DGI2-FG1 [46]. The implementation is based on HiLA [19] (cf. Section 3.3) as it supports the access to UNICORE 6 via an easy and unique API. Furthermore it is not necessary to install components of UNICORE on the submitting host.

4.6 Application Support through GridBeans

Application support in the URC is realised by the GridBean concept [47]. It is a plug-in technology that allows to create and configure an application Grid job via feature rich interfaces, so called GridBeans, that represent the application's behaviour. Each GridBean consists of a GridBean model and one or more graphical panels. The GridBean model stores the application parameters

and represents an abstract Grid job description. The panels provide graphical components for the preparation and validation of user input. The URC itself contains a set of standard panels for each GridBean. These include a file panel to define input and output files, a resource panel to specify resource requirements, and a variables panel to set environment variables. In addition, GridBeans can provide own panels in order to set application specific parameters and files as well as to monitor application behaviour, or to visualise output results, e.g. to represent a molecular 3D structure or to display various statistics over time.

The advantage of the GridBean technology is that GridBeans can be easily developed and independently distributed to the users. The URC distribution package contains by default the Script GridBean, shown at the left hand site of Figure 4.3, which provides an interface for editing and submitting any kind of scripts and the new Generic GridBean, which offers generic panel elements for individually configuring and submitting any application installed on the target system. Furthermore, several GridBeans were developed for specific scientific applications, e.g. for molecular dynamics simulations packages Gaussian [48] and AMBER [49].

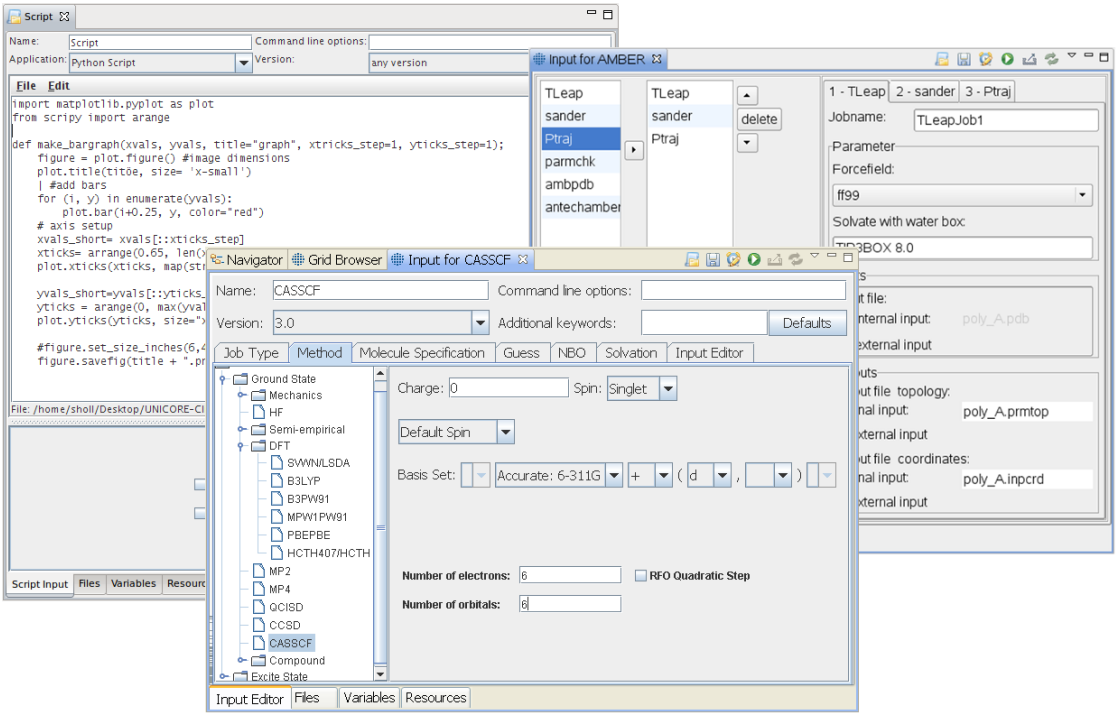

Figure 4.3: The left screenshot shows the standard Script GridBean, to submit any kind of scripts. The middle screenshot shows the Gaussian GridBean, and the right screenshot shows the AMBER GridBean.

The AMBER GridBean was established within a new GridBean concept, the sequence GridBeans [50]. This concept was developed to manage applications, which consist of a set of programs that require to be executed in a sequence one program after another on the same target system. As the molecular dynamic simulation package AMBER consists of approximate 50 programs, and for one molecular simulation a sequence of different programs is required, the AMBER GridBean was realised by usage of this new sequence concept. This provides a sequence framework, which we extended with GridBean models, one for each

required program. These extensions are managed and organised by the framework. Moreover, the framework also provides an automatic build of graphical interfaces for sequence of programs and for each program itself, in order to reduce the developers efforts in supporting new applications. The automated GUI, as shown at the right hand site of Figure 4.3, allows for designing the sequence and setting up the inputs and parameters for the individual programs without being familiar with the application batch system commands.

The Gaussian GridBean assists a user with the preparation of the input for submission to Gaussian quantum chemistry program. Gaussian98 and it's latest version Gaussian09 is designed to predict energies, molecular structures, vibrational frequencies and numerous molecular properties for a broad range of molecular systems in the gas phase and in solution, and it can model both their ground state and excited states. The Gaussian GridBean makes preparing complex input easy for many types of Gaussian calculations. It displays the availability of Gaussian and it's versions on remote systems and allows a user to generate the input parameters automatically without necessity to know of the internal format of the input file. In addition, it allows to set up environment variables, e.g. in order to indicate the locations of the scratch files on a remote system. The user input and specific resource settings will be verified to check correctness of a job setting before a job can be submitted to a remote system.

4.7 Monitoring of UNICORE 6 Services in Operation

Running Grid middleware on distributed systems always means running a quite complex infrastructure. The more systems are connected in the Grid the more complex the installation will get. The users of the Grid require 24/7 availability of the services they need. Hence the administrator's goal should be to get information about any problems before the user runs into problems. Therefore, it is essential to have efficient monitoring tools for the respective Grid middleware.

For UNICORE 6 a tool called SIMON 6 has been developed to check for the health status of every UNICORE 6 component. Besides easy installation and configuration, a central design requirement for SIMON 6 was to check not only the availability of the UNICORE 6 services but also their functionality. A service which is available in the process list does not necessarily deliver the expected functionality.

To be able to check both availability and functionality at the same time, SIMON 6 uses the UNICORE 6 internal job description. The jobs are submitted via the UCC into the UNICORE 6 Grid isochronously and automatically.

The SIMON 6 concept design is shown in Figure 4.4: The availability and functionality tests are submitted into the UNICORE infrastructure using the Gateway as the entry point. All UNICORE 6 components (registry, workflow engine, XNJS, XUUDB and TSI) are monitored one after the other, and afterwards the results come back to SIMON 6 again. If any problem with the UNICORE 6 components occur, the administrator will be informed by email or a short text message to his/her mobile phone.

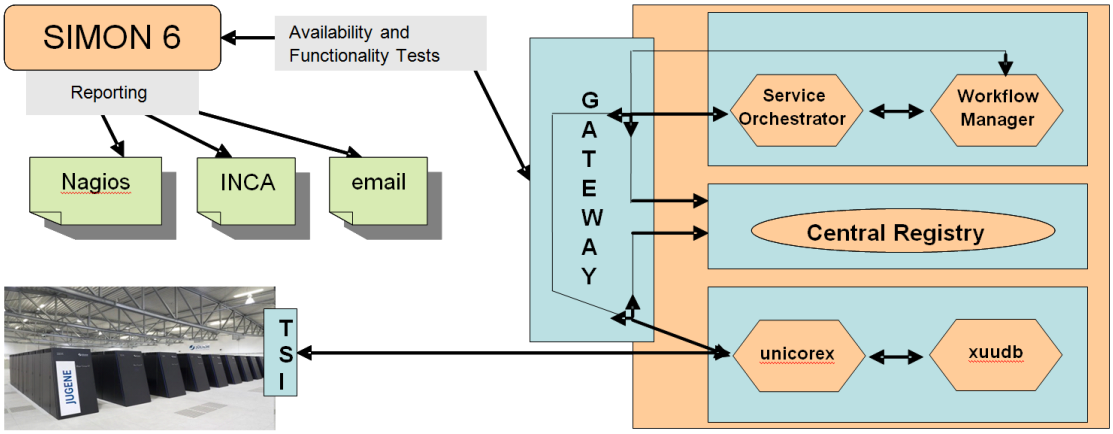

Figure 4.4: SIMON 6 concept design

4.8 Remote administration of UNICORE 6

The decentralised topology and distributed operation of a Grid make a strong case for remote administration capabilities of crucial Grid components. UNICORE 6 is being enhanced by different mechanisms to address those needs.

4.8.1 Dynamic Service Deployment

Given the fact that UNICORE 6 exposes its services as Web services, service administration (installation and removal) plays a crucial role in maintaining administrative flexibility. However, until recently, installing a new Web service to a UNICORE 6 server had been a manual process requiring a restart, thus interrupting the availability of the services. The conflict between flexibility and availability has been resolved by extending UNICORE 6 with dynamic deployment capabilities allowing for installation and removal of services at runtime. These capabilities have been integrated into the underlying Web service framework of UNICORE 6 by developing a dedicated deployment manager that takes a JAR (Java archive) file containing the Web service classes, creates a class loader for this Web service to load its classes, deploys the Web service and modifies the service configuration.

There are two ways to dynamically deploy UNICORE 6 services. Either via an administration Web service interface, which can be toggled at runtime, allowing a local administrator to overrule a remote one. Or on the file system level by copying (either locally or remotely) a Web service JAR into a specified directory triggering an automatic automatic deployment task. This scenario requires the JAR file to contain deployment-specific metadata that can be supplied within the JAR manifest (a simple text file) or as a Java annotation in the Web service interface class.

4.8.2 AdminService – An Administration Web Service Facade

The AdminService is a Web service tailored to administrative functionality. In its current shape, it provides means for:

- (un)deploying UNICORE 6 services
- (de)activating dynamic and automatic deployment of UNICORE 6 services
- creating/inspecting/deleting/terminating UNICORE 6 service resources
- listing available UNICORE 6 services and UNICORE 6 service resources
- getting/setting UNICORE 6 configuration properties
- retrieving mutual dependencies among UNICORE 6 services (dependencies have to be annotated on service source code level)

In addition the AdminService logs all its actions and outcomes (success or failure and cause) in a retrievable manner. Since there might be scenarios in which remote administrators shouldn't be privy to all UNICORE 6 configuration properties (e.g. keystore and database passwords), local administrators must explicitly enable remote access (either 'read' or 'read-write'). Additionally, a mechanism has been added to allow local administrators to set UNICORE configuration properties at runtime without having to use an application programming interface as well as to overrule remote administrators. Essentially it is a periodically parsed configuration file containing key-value pairs denoting UNICORE 6 system properties with corresponding values.

4.9 New Workflow Constructs – for-each-loop

The UNICORE workflow system offers basic workflow constructs like *while-loops* for repeating certain tasks automatically and *if-statements* for altering the flow of execution based on the evaluation of conditions. These constructs can also be nested in order to address more complex tasks. In addition, the system supports the parallel execution of many similar jobs or sub-workflows by introducing the *for-each-loop*. This powerful construct helps to minimise the effort of creating workflows with many jobs and reduce the size of the resulting workflow descriptions. There are numerous usage scenarios where extensive parallel job processing is desired and for-each-loops can be applied. For covering these scenarios, the for-each-loop offers two different modes of operation.

The first mode provides means to iterate over sets of differing parameter values. To this end, the user can declare one or more workflow variables. For each of these variables, he/she may then specify a set of possible values to iterate over. This is done by setting an initial variable value, a modifier expression that is used to alter the value from one iteration to the next, and an exit condition that defines the last value in the set. The variables can be referred to within the body of the for-each-loop (the loop body is a simple container that can hold jobs or sub-workflows). The most common use case for this mode would take the declared variables as input parameters for jobs in the loop body. If multiple variables are declared, all combinations of possible values for the variables are generated and sweeped.

The second operational mode of the for-each-loop helps to iterate over a set of files. The user may specify any number of files he/she wishes to include in the set. Each iteration in this mode will then deal with one of the declared files, e.g. by importing it into the working directory of a job and processing it

with an application program. The files in the set can be located in different places (e.g. on the client computer, on UNICORE 6 storages, on FTP servers, etc.) and the file addresses may contain wildcards if the files' source locations support this. Since the number of files or variable values to iterate over can be very large, the user may limit the amount of iterations that should be processed in parallel in order to keep congestion of the Grid reasonable.

Chapter 5

Future Developments

5.1 Execution Environments

Scientific applications require different execution environments on computing resource, e.g. a parallel MPI execution. These environments are mostly defined by the execution mode as well as its specific parameters and variables. Since UNICORE 6 users want to work in different execution environments, the usage of different and configurable execution environments should be ensured in Grid infrastructures. To facilitate this, a new design concept is developed to support the execution of applications in different execution environments within UNICORE 6. This new development provides a powerful and easy to use configuration mechanism in the URC. In detail, a new section in the URC resource panel, to set the execution environment will be added. It will allow for modifying parameters as well as variables, and provides an immediate validation of those. All available execution environments, their parameters, variables and validators of a specific system are described by the system administrator in the XNJS and its IDB. These descriptions are exported by the URC and are automatically graphically displayed in the new section of the resource panel, to be configured by the user.

To sum up, this new execution environment concept, will provide users with additional options for easier and more precise definition of their application execution on the Grid.

5.2 Licence Management

5.2.1 Current Situation

The currently existing licensing models for commercial applications are focussing on software used on compute resources within an administrative domain. Licenses are provided on the basis of named users, hostnames, or sometimes as a site license for the administrative domain of an organisation. These licensing models make it almost impossible to run license protected applications in a distributed service oriented infrastructure, because the licenses are usually bound to hardware within the domain of the user and do not allow access from the outside thus enforcing *local* use of the protected applications only.

Grid environments are usually spread across multiple organisations and their administrative domains, and virtualised infrastructures, like utility and cloud computing, hide the underlying hardware and their performance indicators, e.g. CPU type and frequency, that are often used in license agreements. While current mechanisms limit the usage of licensed software in Grid environments and virtualised infrastructures, the changing paradigms of IT infrastructures towards usage of virtualised environments and infrastructures make overcoming these limitations vital.

5.2.2 SmartLM Solution

The SmartLM solution [51] is a generic and flexible licensing virtualisation technology based on standards for new service-oriented business models. The solution implements software licenses as Grid services, providing platform-independent access just like other Grid resources and being accessible from resources outside organisational boundaries. This enables users to execute license protected applications on all compute resources within a UNICORE operated Grid infrastructure.

In order to overcome the limitations of the current monolithic licensing model, licenses are reformulated as manageable Grid resources or even Grid services using Service Level Agreements (SLAs) based on the WS-Agreement specification [52]. Secure agreements are used to transport licenses through the Grid and to make them available on the resource to which a user has been granted access for the execution of the application. The license agreement and conditions of use for an application are reached through negotiation between service providers and service customers. Thus, allowing managing and orchestrating licenses in a job or workflow together with other resources like compute nodes, data storage, and network Quality of Service (QoS). The solution is currently being integrated into the major Grid middleware UNICORE and Globus. New service-oriented business models for this approach have been identified and a number of widely-used license-protected commercial applications are currently adapted to be executed under control of the new licensing mechanisms and will become part of a high quality show-case to convince more ISVs to adapt their applications.

5.3 Enhanced Remote Administrative Tools

Apart from recent remote administration extensions (cf. to Section 4.8), there are additional ones still under development, both on the client and service layers. On the server-side, the AdminService is in the process of being extended with mechanisms allowing for remote and dynamic modification of the UNICORE 6 access policy, thus further enhancing administrative flexibility. Currently, the URC is being extended with an administration view that will expose AdminService operations in an intuitive way, allowing for prompt and remote administrative reactions at runtime.

Chapter 6

Summary

In this paper we presented recent and future advancements of the UNICORE Grid technology. UNICORE has a three layered architecture with client, service and system layer. The three available clients – graphical, command-line and API – allow to submit jobs and workflows satisfying the users' requirements and favourite working styles. Several standards from the Web services and Grid domain are implemented in UNICORE 6 so that standards-based interfaces are offered. This makes UNICORE 6 an ideal Grid middleware to be used for interoperability of Grid e-infrastructures worldwide.

Recent functional additions to UNICORE 6 contain a support for VOs and Shibboleth in the security domain, an interactive access based on X.509 certificates and the integration of the DataFinder for improved data management. The support for applications from users is improved through new application-specific GridBeans and the implementation of JavaGAT as a new client to UNICORE. In addition new workflow constructs to allow for-each-loops and an iteration over file-sets was added. To improve the operational aspects of UNICORE 6 a new monitoring tool was developed, which allows to monitor both the availability and functionality of UNICORE 6 services. Furthermore the administration capabilities of UNICORE 6 were enhanced with a dynamic service deployment technology.

Future advancements of UNICORE 6 contains improved Execution Environments to e.g. specify in a more intuitive way the parameters and variables of MPI jobs. Furthermore license management solutions will be developed and integrated with UNICORE 6, that address the usage of licenses in distributed environments across multiple administrative domains. Finally enhanced remote administration tools will be added to UNICORE 6. This will enable a further uptake of UNICORE 6 both in the academic and commercial domain.

Bibliography

[1] A. Streit, D. Erwin, Th. Lippert, D. Mallmann, R. Menday, M. Rambadt, M. Riedel, M. Romberg, B. Schuller, and Ph. Wieder: UNICORE – From Project Results to Production Grids. L. Grandinetti (Edt.), Grid Computing: The New Frontiers of High Performance Processing, Advances in Parallel Computing 14, Elsevier, 2005, pp. 357 – 376

[2] M. Riedel, B. Schuller, D. Mallmann, R. Menday, A. Streit, B. Tweddell, M.S. Memon, A.S. Memon, B. Demuth, Th. Lippert, D. Snelling, S. van den Berghe, V. Li, M. Drescher, A. Geiger, G. Ohme, K. Benedyczak, P. Bala, R. Ratering, and A. Lukichev: Web Services Interfaces and Open Standards Integration into the European UNICORE 6 Grid Middleware. Proceedings of 2007 Middleware for Web Services (MWS 2007) Workshop at 11th International IEEE EDOC Conference "The Enterprise Computing Conference", 2007, Annapolis, Maryland, USA, IEEE Computer Society, ISBN 978-0-7695-3338-4, pp. 57 – 60

[3] R. Menday: The Web Services Architecture and the UNICORE Gateway. Proceedings of International Conference on Internet and Web Applications and Services (ICIW 2006), Guadeloupe, French Caribbean, 2006, IEEE Computer Society Press, p. 134

[4] B. Schuller, R. Menday, and A. Streit: A Versatile Execution Management System for Next-Generation UNICORE Grids. In Proc. of 2nd UNICORE Summit 2006 in conjunction with EuroPar 2006, Dresden, Germany, LNCS 4375, Springer, pp. 195 – 204

[5] B. Schuller, B. Demuth, H. Mix, K. Rasch, M. Romberg, S. Sild, U. Maran, P. Bala, E. del Grosso, M. Casalegno, N. Piclin, M. Pintore, W. Sudholt, and K.K. Baldridge: Chemomentum – UNICORE 6 based infrastructure for complex applications in science and technology. Proceedings of 3rd UNICORE Summit 2007 in Springer LNCS 4854, Euro-Par 2007 Workshops: Parallel Processing, pp. 82 – 93

[6] V. Venturi, M. Riedel, A.S. Memon, M.S. Memon, F. Stagni, B. Schuller, D. Mallmann, B. Tweddell, A. Gianoli, V. Ciaschini, S. van de Berghe, D. Snelling, and A. Streit: Using SAML-based VOMS for Authorization within Web Services-based UNICORE Grids. Proceedings of 3rd UNICORE Summit 2007 in Springer LNCS 4854, Euro-Par 2007 Workshops: Parallel Processing, pp. 112 – 120

[7] A.S. Memon, M.S. Memon, Ph. Wieder, and B. Schuller: CIS: An Information Service based on the Common Information Model. Proceedings of 3rd IEEE International Conference on e-Science and Grid Computing, Bangalore, India, December, 2007, IEEE Computer Society, ISBN 0-7695-3064-8, pp. 465 – 472

[8] S. Andreozzi et.al. GLUE Specification v. 2.0, OGF Grid Final Document (GFD) #147, http://www.ogf.org/documents/GFD.147.pdf

[9] I. Foster et.al. OGSA Basic Execution Service Version 1.0, OGF Grid Final Document (GFD) #108, http://www.ogf.org/documents/GFD.108.pdf

[10] B. Dillaway et.al. HPC Basic Profile, Version 1.0, OGF Grid Final Document (GFD) #114, http://www.ogf.org/documents/GFD.114.pdf

[11] G. Wasson et.al. HPC File Staging Profile, Version 1.0, OGF Grid Final Document (GFD) #135, www.ogf.org/documents/GFD.135.pdf

[12] M. Marzolla, P. Andreetto, V. Venturi, A. Ferraro, A.S. Memon, M.S. Memon, B. Twedell, M. Riedel, D. Mallmann, A. Streit, S. van de Berghe, V., Li, D. Snelling, K. Stamou, Z.A. Shah, and F. Hedman: Open Standards-based Interoperability of Job Submission and Management Interfaces across the Grid Middleware Platforms gLite and UNICORE. Proceedings of International Interoperability and Interoperation Workshop (IGIIW) 2007 at 3rd IEEE International Conference on e-Science and Grid Computing, Bangalore, India, December, 2007, IEEE Computer Society, ISBN 0-7695-3064-8, pp. 592 – 599

[13] M. Morgan et.al. OGSA ByteIO Specification, OGF Grid Final Document (GFD) #87, http://www.ogf.org/documents/GFD.87.pdf

[14] Grid Ecosystem – GridFTP,
http://www.globus.org/grid_software/data/gridftp.php

[15] M. Riedel, R. Menday, A. Streit, and P. Bala: A DRMAA-based Target System Interface Framework for UNICORE. Proceedings of Second International Workshop on Scheduling and Resource Management for Parallel and Distributed Systems (SRMPDS'06) at Twelfth International Conference on Parallel and Distributed Systems (ICPADS'06), Minneapolis, USA, IEEE Computer Society Press, pp. 133 – 138

[16] Eclipse-based UNICORE Rich Client (URC),
http://www.unicore.eu/download/unicore6/

[17] Eclipse, http://www.eclipse.org/

[18] UNICORE Command-line Client (UCC),
http://www.unicore.eu/documentation/manu-als/unicore6/ucc,
download: http://www.unicore.eu/download/unicore6/

[19] HiLA 1.0,
http://www.unicore.eu/community/development/hila-reference.pdf

[20] DEISA – Distributed European Infrastructure for Supercomputing Applications, http://www.deisa.eu/

[21] A-WARE project, http://www.a-ware-project.eu/

[22] T.M. Sloan, R. Menday, T. Seed, M. Illingworth, and A.S. Trew: DESHL – standards-based access to a heterogeneous European supercomputing infrastructure. In Proc. 2nd IEEE International Conference on e-Science and Grid Computing – eScience 2006.

[23] S. Jha et.al. A Simple API for Grid Applications (SAGA), OGF Grid Final Document (GFD) #90, http://www.gridforum.org/documents/GFD.90.pdf

[24] Grid Application Toolkit – GridLab Project, http://www.gridlab.org/WorkPackages/wp-1/documentation.html

[25] The UVOS project, http://uvos.chemomentum.org

[26] J. Hughes et al.: Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0, OASIS Standard, 15 March 2005, http://docs.oasis-open.org/security/saml/v2.0/

[27] T. Scavo: SAML V2.0 Deployment Profiles for X.509 Subjects, OASIS Committee Specification, 27 March 2008. http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml2-profiles-deploy-x509-cs-01.pdf

[28] V. Venturi, T. Scavo, D. Chadwick: OGSA Attribute Exchange Profile Version 1.2. Open Grid Forum, 2007.

[29] Shibboleth, http://shibboleth.internet2.edu

[30] SAML Specifications, http://saml.xml.org

[31] OASIS eXtensible Access Control Markup Language (XACML), www.oasis-open.org/committees/xacml

[32] T. Barton, J. Basney, T. Freeman, T. Scavo, F. Siebenlist, V. Welch, R. Ananthakrishnan, B. Baker, M. Goode and K. Keahey: Identity Federation and Attribute-based Authorization through the Globus Toolkit, Shibboleth, GridShib, and MyProxy. 5th Annual PKI R&D Workshop, 2006.

[33] A. Faroughi, R. Faroughi, P. Wieder, W. Ziegler: Attributes and VOs: Extending the UNICORE Authorisation Capabilities. Proceedings of Euro-Par 2007 Workshops: Parallel Processing, HPPC 2007, UNICORE Summit 2007, and VHPC 2007, Revised Selected Papers, Springer, 2008, LNCS 4854, ISBN 978-3-540-78472-2, pp. 121 – 130

[34] SOCKS 5, http://www.rfc-editor.org/rfc/rfc1928.txt

[35] gEclipse Project, http://www.geclipse.eu/

[36] The Globus Alliance, http://www.globus.org/

[37] P. Fuhrmann, V. Gülzow: dCache – Storage System for the Future. In Euro-Par 2006 Parallel Processing, Springer, 2006, pp. 1106 – 1113

[38] C. Baru, R. Moore, A. Rajasekar, M. Wan: The SDSC Storage Resource Broker. In Proceedings of CASCON, 1998

[39] A. Rajasekar, M. Wan, R. Moore, W. Schroeder: A prototype rule-based distributed data management system. In HPDC workshop on "Next Generation Distributed Data Management", 2006

[40] T. Schlauch, A. Schreiber: Datafinder – A Scientific Data Management Solution. In: Ensuring the Long-Term Preservation and Value Adding to Scientific and Technical Data, PV 2007, Oberpfaffenhofen, Germany.

[41] UNICORE DataFinder project, http://tor-2.scai.fraunhofer.de/gf/project/unicoredata/

[42] AeroGrid, http://www.aero-grid.de

[43] G. Allen, K. Davis, T. Dramlitsch, T. Goodale, I. Kelley, G. Lanfermann, J. Novotny, T. Radke, K. Rasul, M. Russell, E. Seidel, O. Wehrens: The GridLab Grid Application Toolkit. Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing (HPDC), 2002, IEEE Computer Society, p. 411

[44] GridLab: A Grid Application Toolkit and Testbed, http://www.gridlab.org

[45] A. Beck-Ratzka, T. Zangerl: GAT beginners guide. http://www.aei.mpg.de/~alibeck/documents/gat-usage.pdf

[46] D-Grid: The German Grid Initiative., http://www.d-grid.de

[47] R. Ratering, A. Lukichev, M. Riedel, D. Mallmann, A. Vanni, C. Cacciari, S. Lanzarini, K. Benedyczak, M. Borcz, R. Kluszcynski, P. Bala, and G. Ohme. GridBeans: Supporting e-Science and Grid Applications. Second IEEE International Conference on e-Science and Grid Computing: IEEE Conference Proceedings, 2006, p. 45

[48] Gaussian, http://www.gaussian.com/

[49] The Amber Molecular Dynamics Package, http://ambermd.org/

[50] S. Holl, M. Riedel, B. Demuth, M. Romberg, A. Streit, V. Kasam: Life Science Application Support in an Interoperable E-Science Environment. Proceedings of IEEE International Symposium of Computer-Based Medical Systems (CBMS) 2009, to appear

[51] SmartLM project, http://www.smartlm.eu/

[52] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Kakata, J. Pruyne, J. Rofrano, S. Tuecke, M. Xu: Web Services Agreement Specification. OGF Grid Final Document (GFD) #107, http://www.ggf.org/documents/GFD.107.pdf

Jül-4319
Februar 2010
ISSN 0944-2952