

The UNICORE Grid Infrastructure

Mathilde Romberg

Research Center Jülich
Central Institute for Applied Mathematics
D-52425 Jülich, Germany
m.romberg@fz-juelich.de

Abstract

UNICORE (Uniform Interface to Computer Resources) is a software infrastructure to support seamless and secure access to distributed resources. UNICORE allows uniform access to different hardware and software platforms as well as different organizational environments. Based on the abstract job model it offers services for security, translation of abstract jobs into real batch jobs for different target systems, and a public key infrastructure. This paper describes the UNICORE architecture and provided services.

1 Introduction

In 1997 the German Ministry for Education and Research (BMBF) approved and funded a two years project to develop a software infrastructure for seamless access to distributed supercomputer resources. Motivation for the UNICORE (UNiform Interface to COmputing REsources) project¹ has been that users who have to solve large problems in computational science usually need resources on a variety of systems at different locations. These users are faced with different site policies and practices (security, user identification, data management, ...), different system architectures, and system software. For the efficient use of the resources the users need to learn about these differences. UNICORE is designed to overcome the additional effort. The project goal is to provide batch users of the German supercomputer centers with a seamless, secure, and intuitive Web-based interface for job preparation, submission, and monitoring. The developed UNICORE prototype runs in test-mode at the centers in Berlin (ZIB), Jülich (FZJ), Karlsruhe (RUKA), München (LRZ), Offenbach (DWD), and Stuttgart (RUS) for the target systems T3E, SP2, and VPP. It provides the user with a single sign-on environment via X.509 certificates. Through the graphical interface the user can prepare multi-step UNICORE jobs consisting of script jobs for shell-scripts of existing applications, Compile-Link-Run-jobs for new applications, and file transfer jobs. Jobs to be run at the same target system are grouped together in job-groups. The user specifies general information like the target system per job group. Resource requirements for CPU time, number of

¹See <http://www.fz-juelich.de/unicore>; UNICORE has been funded by BMBF, grant period 1.8.1997 - 31.12.1999, grant id 01 IR 703

CPUs, amount of main memory, and disk space are specified per job. The user input is mapped by UNICORE to the target system specific commands and options. The job monitor allows for checking the job status and accessing the standard output and error files.

A follow-on project, UNICORE Plus², started in January 2000 and is planned for a duration of three years. Based on the work done in the UNICORE project UNICORE Plus is going to develop a grid (see [3]) infrastructure for seamless and secure access to supercomputer resources. This includes a sophisticated graphical user interface and security mechanism as well as server components for the translation of the uniform job specification into batch jobs for the target system, for scheduling etc. The project goals are to develop an improved, robust prototype which allows for multi-part, multi-site jobs with elaborate job flow functions and which supports application specific interfaces, efficient data transfer, resource modeling, and meta-computing at application level. As within UNICORE it is the design principle to keep site autonomy.

Project partners are the German Weather Service (DWD), Research Center Jülich, Computer Center of the University of Stuttgart (RUS), Pallas GmbH, Brühl, Leibniz Computer Center, Munich (LRZ), Computer Center of the University Karlsruhe (RUKA), Paderborn Center for Parallel Computing (PC²), Konrad-Zuse-Center, Berlin (ZIB), and Center for High Performance Computing at TU Dresden (ZHR). The project is structured in eight sub-projects each dealing with one aspect, including software development and project-management. Topics of the sub-projects are test, quality management, administration, and public key infrastructure (PKI), resource modeling, application specific interfaces, data management, job control flow, and meta-computing. While in the UNICORE project the hardware vendors³ were non-funded project partners they now are members of the recently founded UNICORE Forum e.V. to further support the UNICORE deployment.

2 The UNICORE Architecture

The architecture used in the UNICORE Plus project is based on the three tier architecture developed in the predecessor project (see [1], [4], <http://www.fz-juelich.de/unicore>). It consists of user, UNICORE server, and target system tier as shown in Figure 1. User and server tiers are Java applications while the system tier is presently implemented in perl. The user tier consists of the graphical user interface. It offers the functions to prepare and control UNICORE jobs (for details see Chapter 3) and to set up and maintain the user's security environment. The security architecture is based on the Secure Socket Layer (SSL) protocol. SSL is used for the communication between the components which talk to each other over public networks. From the user input the user interface generates an Abstract Job Object (AJO) which is sent via SSL to the Gateway. The AJO is a

²See <http://www.fz-juelich.de/unicoreplus>; UNICORE Plus is a BMBF funded project, grant id 01 IR 001

³Hitachi, HP, IBM, NEC, SGI/Cray, Siemens/Fujitsu, Sun

key component in the architecture. It comprises the UNICORE protocol between user interface and the Network Job Supervisor together with the abstract job specification generated from the user input. The AJO is realized as a Java class library. [4] and <http://www.fz-juelich.de/unicore/DARR.ps> explain the AJO in more detail.

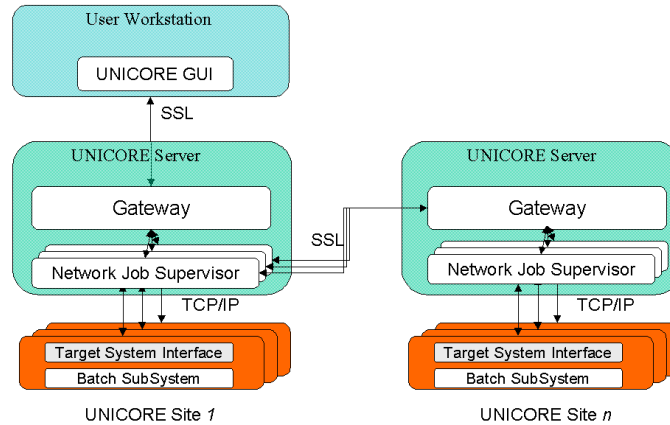


Fig. 1: Architecture Overview

The Gateway is the first part of the UNICORE Server tier. It takes care of the user authentication, secure communication between client and server, and provides the client with information about the available resources at the site. It also talks to the Network Job Supervisor (NJS) servers at the site to send jobs and data, status requests and control commands for further processing and to receive data to make it available to the user. Each NJS controls one Vsite (Virtual Site) which is a single system or one cluster of systems which share the same Userids and file space. It fulfills the following tasks:

- analyze the AJO (representing the UNICORE Job) to extract local and remote jobs;
- map the UNICORE user id to the local userid for the target system;
- translate the local jobs contained in the AJO into real batch jobs for the target system;
- send job-groups to be executed at other UNICORE sites to the corresponding gateway;
- provide local resource information to the gateway;
- take care of the necessary file transfer;
- provide job status information and job output.

The underlying recursive job model is such that a UNICORE job consists of

- job groups, which build a frame with general information for this part of the job like the target system; they contain them-self jobs groups, jobs, and dependencies,
- jobs or tasks, which are to be translated into batch jobs for the target,
- dependencies between the elements to reflect the necessary synchronization; they build a directed acyclic dependency graph.

Each UNICORE job is assigned a UNICORE Job directory which is the UNIX working space for the job. It is a temporary directory existing only during the lifetime of the job at the site. All data needed for the job execution has to be imported from permanent file space to the job directory. All data which is needed after the job has finished has to be saved to permanent file space. The user specifies the data to be imported and those to be exported when constructing the job. The transfers are done by UNICORE transparently to the user.

In the following we take a closer look at the architecture explaining the three tiers in more detail with respect to the security features. The security is based on the Secure Socket Layer (SSL) protocol and the X.509V3 type certificates (see [2]). SSL uses public key cryptography to establish connections between client and server. Therefore each component has to have a public-private key pair with the private key kept secret and the public part being known by the others. The keys have to be certified by a certification authority (CA) so that the components can be sure that they communicate with the user (or program) he or she (or it) claims to be. By default certificates signed by unknown signers are not accepted. The UNICORE user's X509 certificate is his or her UNICORE user identification. It is maintained by the user interface application in a encrypted data base. The user interface also needs to know about the Certification Authority which signs the user and the gateway certificates. With this information a secure connection with mutual authentication of client and gateway can be established. Figure 2 shows at the top the user tier hosting the main GUI components Job Preparation Agent (JPA) and Job Monitor Controller (JMC) as well as the user's certificate. The user is authenticated by the Gateway when presenting his or her certificate. The certificate is also part of the AJO for authorization at different sites and the AJO components as well as the whole AJO are signed with the user's certificate. This is necessary to make sure that no one can tamper with the AJO on its way without being noticed.

UNICORE uses the UNICORE Protocol Layer (UPL) to send the AJO to the Gateway which hands it over to the NJS controlling the specified Vsite. In case a UNICORE site uses additional authentication methods like DCE (Distributed Computing Environment) or SecureID cards, the Gateway allows for site specific additional authentication as depicted in Figure 2. Gateway and NJS communicate via sockets. The NJS first checks whether the AJO is an correct AJO from the user who signed it. It unpacks the AJO with the user's public key and analyses it for parts to be send to other target systems. These are send either to another local NJS or to a gateway at another UNICORE site at the point in time the dependency graph defines. Each NJS has its own X.509 cer-

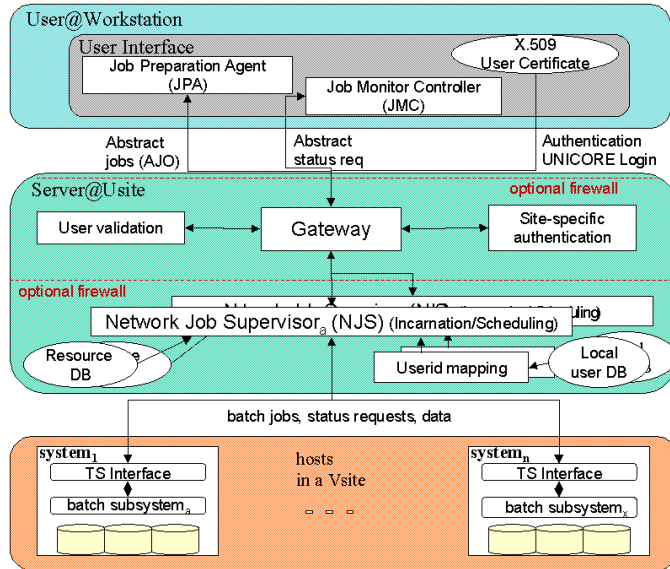


Fig. 2: Detailed Architecture

tificate which it uses to communicate to a Gateway at another site via SSL. The Gateway then knows, that the AJO is part of another AJO and that it has to use the user's public key to unpack the AJO instead of the one from the sending NJS. For tasks to be processed locally the NJS performs the userid mapping which means that it maps the user's certificate to his or her user identification at the local system. The NJS translates the abstract job definition into batch jobs for the destination system and sends them according to the dependency graph to the target system interface (TSI) on the system. The UNICORE server inter-operates with firewalls. The server can either be completely behind the site's firewall or the firewall is located between Gateway and NJS.

The Vsite is either a cluster of systems which share the same userids and file space or a stand-alone system. Each of the systems is governed by a target system interface (TSI). The TSI talks to the local batch system to maintain the job directory, submit jobs on behalf of the user, and to control the job status.

A key component for the security architecture in UNICORE is the Public Key Infrastructure (PKI). It is needed to generate the certificates for users, Gateways, software developers, and NJSes. The UNICORE PKI policy is based on the regulations defined by the DFN-PCA (German Research Network - Policy Certification Authority⁴). The UNICORE Certification Authority (CA) is located at LRZ, all partner centers operate a Registration Authority (RA).

⁴see <http://www.cert.dfn.de/dfnpca>

3 The User Interface

Within the UNICORE environment the user has a comfortable way to use distributed computing resources without having to learn for site or system specifics. It is all done in a seamless way: The user specifies the job requirements (e.g. resources, command options, input and output files, etc.) independently from the selected target system and site. The NJS translates them into the real batch job for the target. A key point is that a user has a single UNICORE userid, his or her UNICORE X.509 certificate, to get access to the resources at the various UNICORE sites. The graphical user interface offers at top level the functions to maintain the security settings, to prepare UNICORE jobs, and to monitor them.

A UNICORE job can be build from multiple parts which can be executed asynchronously or dependently on different systems at different UNICORE sites. Currently the following elements are offered by the Job Preparation Agent (JPA):

- Script task, to submit existing job scripts via UNICORE;
- Compile-Link-Run task, to prepare new applications;
- Job groups, to build subjobs for other target systems.

Each UNICORE job as well as each job group contains general information for the job:

- the job name;
- the target system, which can be selected from all available Vsites;
- the account id, it is currently not used but will be when it becomes necessary for a UNICORE user to have different accounts;
- the user's e-mail address, where the system should send messages to.

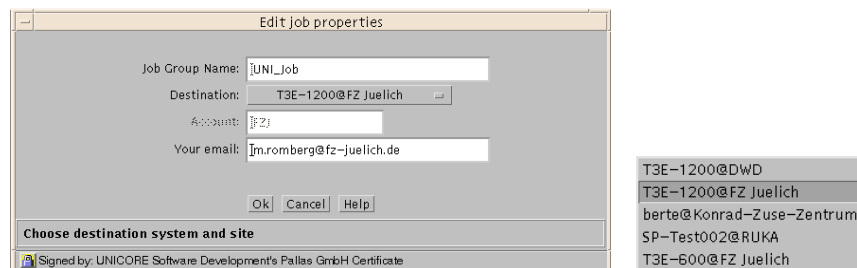


Fig. 3: General Information in a Job Group

Figure 3 shows the input panel and an example list of known target systems the user can select the *Destination* from. User defaults for these values can be set via *Preferences*.

Figure 4 shows an example of the JPA with a job displayed at top-level. On the left side all job elements are listed while on the right side the elements of the selected job group are shown together with their dependencies. The selected

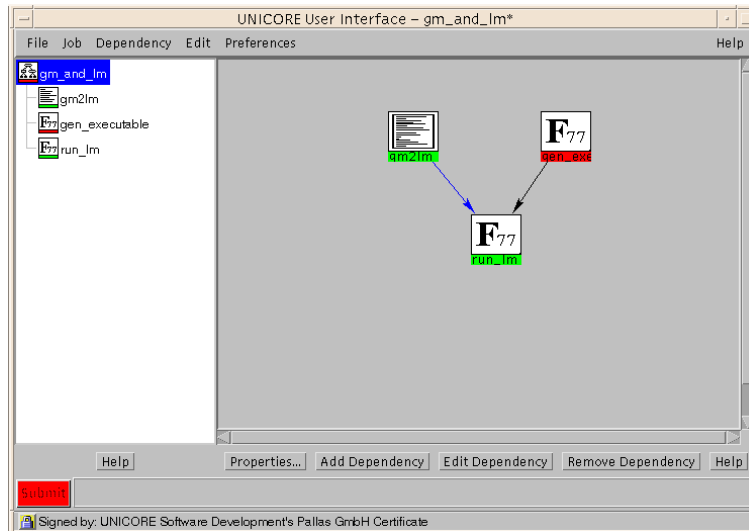


Fig. 4: A Job Example

job group icon with the job name is highlighted. The UNICORE job consists of three different job steps, all of them are to be executed on the same target. *gm2lm* is a script as can be seen from the icon. The other two are compile-link-run tasks. The icons are marked by colors representing their preparation status (red - not yet ready for submission; green - ready for submission). In case all steps of a job are marked *ready for submission* the *Submit* button also turns color and is activated for usage. The icons on the right can be moved by the user to an appropriate location and they can be connected via dependencies to reflect the necessary synchronization between the job steps. There are two dependencies already defined via the *Add Dependency* button at the bottom. They result in an independent execution of *gm2lm* and *gen_executable* while *run_lm* can be scheduled only after both predecessors have finished successfully. Within the dependency definition the user can specify the data set names of the files created by the predecessor and which are needed by the successor job step.

Another example shows the input panel for a compile step of a compile-link-run task (see Figure 5). It is divided into three major areas: general information, Input, and Output. Besides job step name the resources for the job have to be specified. Figure 6a shows the current resources input panel. The number of processors or CPUs, the connect or CPU time, the amount of memory, and job disk space requirements can be specified. From the resource information received from the NJS systems the JPA knows about the maximum values for the selected target system and will not allow a user to specify a value higher than that. This supports the user in creating jobs suitable for the target.

With the *Execution Contexts* (see Figure 6b) a user specifies the needed compiler and special libraries. Again the JPA knows from the resource pages which

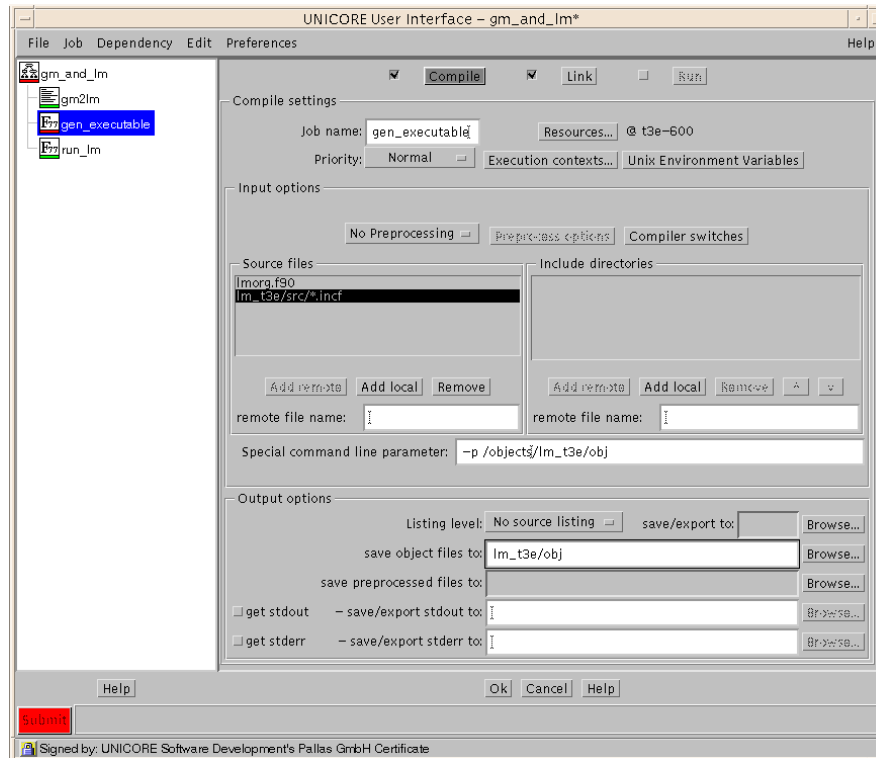
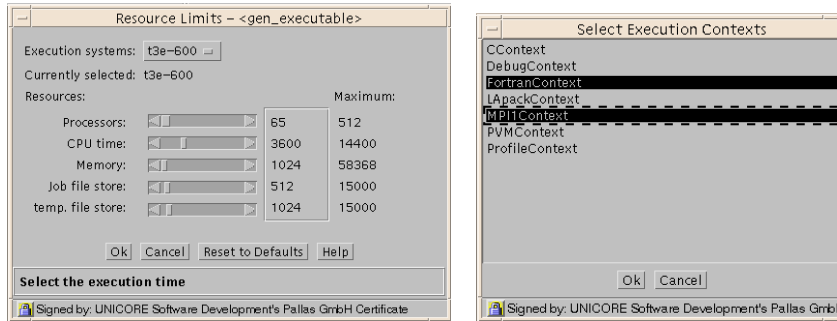


Fig. 5: GUI for COMPILE Task

libraries and application packages are available. Also the *Compiler Switches* are a good example for seamless job preparation. The options specified in Figure 6c are going to be translated by the NJS in what the compiler at the target system understands. As an example, there are three *Optimization levels* offered by the interface: *No Optimization*, *Safe Optimization*, and *Aggressive*. The optimization level *Aggressive* is translated into the Fortran 90 option `-O3,unroll2` for a Cray T3E while for a NEC SX-4 the translation produces `-Wf,-dir par -P auto -C hopt -pi -Wf,-pvctl,fullmsg, vwork=stack,noassume,loopcht=10000000`.

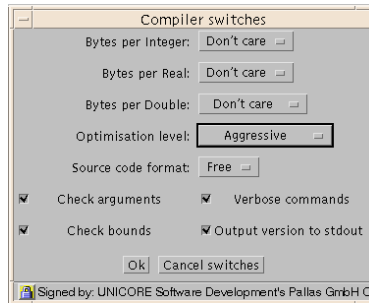
4 Outlook

The UNICORE system as it exists today already provides seamless and secure access to remote resources, supports data transfer between the connected sites, and has established a public key infrastructure. The software written in Java allows for portability, so it can run on a variety of client and server platforms. The data base for the translation of the abstract job specification into a real batch job for the target system is easily adaptable to new platforms. Currently CRAY T3E, NEC SX-4, Fujitsu VPP 700, and IBM SP2 are integrated and the



(a) Resource Specification

(b) Execution Contexts



(c) Compiler Switches

Fig. 6: Compile Task Selection Panels

batch subsystems NQS and LoadLeveler.

Within the UNICORE Plus project the UNICORE system is going to be extended and put into production at the partner sites. In Chapter 1 the areas of development have already been mentioned. Within the short term application specific interfaces for CPMD (Car-Parrinello Molecular Dynamics), MSC-NASTRAN, FLUENT, and STAR-CD are developed. These interfaces will guide users through the configuration step for their application and use the UNICORE mechanisms to seamlessly submit and control the jobs. The interfaces are going to be integrated into the JPA so that e.g. a NASTRAN job can be one job step in a larger UNICORE job. For the longer term it is planned to build a generic interface to allow easier integration of other applications.

The current resource model includes a small set of static resource information. The project will extend the set and will include dynamic information as well so that, for example, a user gets hold of current system load as a decision criterion for target system selection. The new resource model uses XML.

One of the most important issues in the UNICORE Plus project is the data management: How to efficiently transfer data between UNICORE sites, especially if large amounts of data need to be staged to a site for job processing. In addition, access to data archives will be integrated. It is also planned to add a file transfer feature for data transfer independent from UNICORE jobs but using the UNICORE security mechanisms.

In the area of meta-computing on application level three topics are under research: Scheduling of mpp-applications to be run in parallel at different sites (most of the batch subsystems do not support features like advance reservation), techniques for distributed execution of applications including a fault tolerant, batch oriented startup procedure, and mechanisms for visualization of the performance of such meta-computer applications.

All these efforts together with strong testing and quality management will lead to a user, site, and system friendly infrastructure for grid computing.

References

1. Almond, Jim and Dave Snelling, "UNICORE: Uniform Access to Supercomputing as an Element of Electronic Commerce", in *Future Generation Computer Systems*, Vol. 613, 1999, pp.1-10
2. Feghhi, Jahal, Jalil Feghhi, and Peter Williams, "Digital Certificates - Applied Internet Security", Addison-Wesley, 1998
3. Foster, Ian and Carl Kesselman, "The Grid: Blueprint for a New Computing Infrastructure", Morgan Kaufman Publishers, 1998
4. Romberg, Mathilde, "The UNICORE Architecture: Seamless Access to Distributed Resources", in *Proceedings of the eighth IEEE International Symposium on High Performance Distributed Computing*, August 1999, pp.287-293