UNICORE: A Grid Computing Environment

Dietmar W. Erwin and David F. Snelling

Forschungszentrum Jülich GmbH, ZAM
D.Erwin@fz-juelich.de
Fujitsu European Centre for Information Technology
snelling@fecit.co.uk

Abstract. This paper describes the result of the UNICORE (BMBF grant 01 IR 703) project, the goals and the initial results of the follow-on project UNICORE Plus (BMBF grant 01 IR 001). It outlines the original goals of the German funded project to provide a seamless batch interface for German HPC centers and its evolution toward a Grid Computing Environment. The focus is on technical results, like abstraction to achieve seamlessness and use of certificates for authentication and security. The rationale behind technical decisions will be provided and the future role of UNICORE will be presented.

1 Introduction

Grid Computing is an extremely valuable concept and a convincing and compelling computing paradigm. It is easily understood. Directives, from the European heads of state at the June 2000 summit in Lisbon to the European Commission to take appropriate actions, affirm it. Grid Computing has it roots in may developments (references can be found in [2]) that preceded the ground-breaking book by Ian Foster and Carl Kesselmann [1]. By this token, UNICORE is a is a true grid project, although it originated as a project about seamless access to remote resources. The acronym 'Uniform Interface to Computing Resources' indicates this, and it is still well characterized in these terms.

As terminology changes, new concepts are used as buzzwords by the media, politicians, and even insiders, and therefore expectations change. This paper will present both the research and development results of the UNICORE projects and the technical, and sometimes non-technical context, in which the work took place.

2 A Brief History of UNICORE

UNICORE was conceived in 1997 to produce a solution to a real problem: Make the German HPC resources, located for example in Berlin, Munich, Stuttgart and Jülich, accessible to users in a seamless, secure and intuitive way over the Internet. This was what the directors of the German HPC centers demanded [5], and this was fully consistent with the recommendations of the German science council [4]. After reviewing related work, especially in Europe and the US, a project proposal was developed and submitted by the prospective project partners to the BMFT (Federal

R. Sakellariou et al. (Eds.): Euro-Par 2001, LNCS 2150, pp. 825-834, 2001.

Ministry for Research and Technology), later renamed to BMBF (Federal Ministry for Education and Research).

The project partners decided to combine existing and emerging technologies with new ideas to build a software infrastructure for remote access without relying on the successful and timely completion of other related projects. It was also a major challenge to include European partners – even without funding – into a project funded by a German agency. The challenge was further magnified by the inclusion of US and Japanese vendors of HPC systems, which were critical to the success of UNICORE. The project was fortunate to manage this, and the complete list of partners can be found in [6].

The original goals of UNICORE can be summarized as follows:

- Development of a seamless interface to allow creation, manipulation, and control of complex batch jobs to be executed at heterogeneous systems at different remote sites including automatic data staging.
- Use of existing and emerging technologies, components, and standards where ever possible.
- Minimal intrusion into existing computing center practices and policies, especially in the areas of security and administration.
- Modular design that allows replacement of components whenever new solutions are mature enough.

The implementation was based on Java and Java applets; it used the Jigsaw Web Server (see [3]) with Security extension from IAIK Graz and Codine from Genias. It relied on X.509 certificates.

In retrospect, the project was launched at the right time. Traditional HPCN funding declined and novel ideas could prosper and eventually evolve into new strategic directions, even in the absence of accepted terminology. The UNICORE project started in August 1997 for a duration of two years initially, with a target to develop a prototype within this time frame. The result was successfully demonstrated in an international symposium on 'Metacomputing and Distributed Computing' [7] in Jülich, Germany in September of 1999.

During the course of the project it became obvious that some components used for building the first UNICORE system were themselves early implementations of emerging technologies. They made it possible to demonstrate the concept in a very short time but the total solution lacked the necessary robustness to permit its deployment at partner centers in day-to-day operation. The results of UNICORE secured funding for a follow-on project – obviously named UNICORE Plus – for a three year period from January 2000 to December 2002. This project's overall objectives are:

- Re-implement the UNICORE software to create a robust and extensible system for use at partner installations and in additional projects.
- Extend the capabilities of UNICORE in selected areas that had been omitted from the first prototype.
- Add functions to allow maintenance and administration of the UNICORE environment.
- To take the UNICORE software from a research tool to a viable, production

quality tool for a research environment.

- Ensure the commercial exploitation of UNICORE after the end of the project.

The final point is one of the formal requirements of BMBF funding and will be fulfilled by Pallas GmbH, which intends to market and support UNICORE software and services.

Since UNICORE Plus, like its predecessor, is a fixed budget project, its scope has to be restricted to meet the objectives stated in the proposal. It is expected that other projects will add functions to extend UNICORE and that UNICORE's design allows interoperation and integration with other Grid solutions. This is indeed already happening.

3 R&D Results of UNICORE

3.1 The UNICORE Architecture

The functions of UNICORE and its implementation have been described in [8] and [6]. In short, UNICORE, as implemented in the first production prototype, creates a three-tier architecture.

The UNICORE client supports the creation, manipulation, and control of complex batch jobs, which may require multiple systems at one or more UNICORE sites to complete. The jobs and actions the user defines are represented as *Abstract Job Objects,* effectively Java classes, which are serialized and signed when transferred between the components of UNICORE. The server level of UNICORE consists of a Gateway, the secure entry point into a UNICORE site, which authenticates requests from UNICORE clients based on X.509 certificates and passes them to a *Network Job Supervisor (NJS)* for further processing. The NJS maps the abstract request, as represented by the AJO, into concrete jobs or actions to be performed by the target system, if the target system is part of the local UNICORE site. This process is called incarnation. Sub-jobs that have to be run at a different site are transferred to this site's gateway for subsequent processing by the peer NJS (see Fig. 1.). Additional functions of NJS are: Synchronization of jobs to honor the dependencies specified by the user, automatic transfer of data between UNICORE sites as required for the job execution, collection of results from jobs, especially *stdout* and *stderr*, import and export of data between the UNICORE space and target system, and client workstation.

The third tier of the architecture is the target host which executes the incarnated user jobs. A small daemon, called the *Target System Interface (TSI)* resides on the host to interface with the local batch system on behalf of the user. A stateless protocol is used to communicate between NJS and TSI. Multiple TSIs may be started on a host to increase performance.

3.2 The UNICORE Job and Data Model

Sometimes the question is asked why an old-fashioned concept like batch processing is supported at all by UNICORE. The answer is pragmatic and threefold:

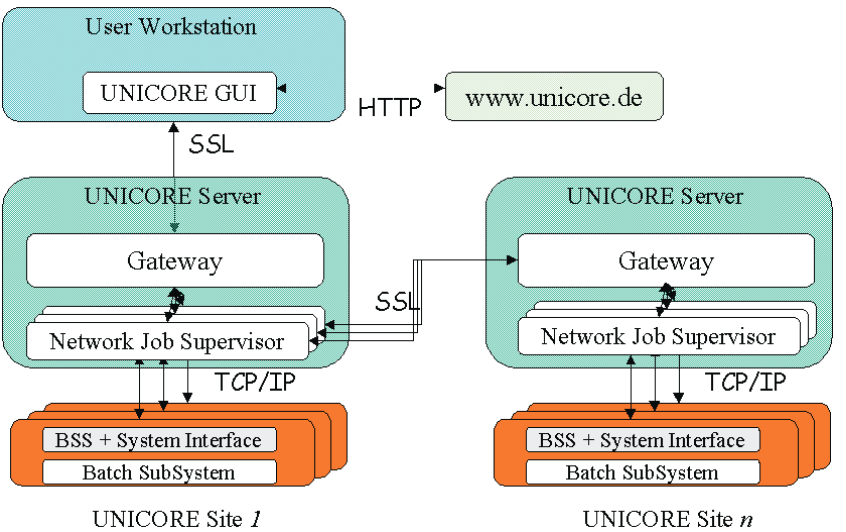

Fig. 1. Multisite UNICORE systems

- Up to 80 percent of the cycles in the German HPC centers are still consumed in batch mode (even scientists sleep some times) and the production environments of HPC centers are UNICORE's primary target.
- Automatic data transfer between sites can be slow over the Internet, but for batch applications the associated latencies can be hidden.
- The need for interactive access to remote systems using the full security and functions of UNICORE are recognized. This however, will be implemented in other UNCIORE based projects, like EUROGRID.

UNICORE defines two important terms: Task and Job Group. A task is the atomic entity that is eventually incarnated into a real batch job for the target system. This fact is not an architectural requirement, but depicts the present implementation. A Job Group is recursively defined to consist of Job Groups and Tasks. The top level Job Group is called a UNICORE job. It is the entity that represents the complete computational solution to the user's problem. The UNICORE job is submitted as a whole to UNICORE.

For the duration of a UNICORE job, the NJS creates a temporary UNICORE space (*Uspace*) using installation specified disk space. The Uspace holds temporary files created by NJS, for example the incarnated jobs, consolidated stdout and stderr, files transmitted from the client to the target, or data created by the executing jobs. UNICORE provides functions to transfer files between Uspaces and to import or export data from or to a permanent file space to which the user has access.

3.3 The Abstract Job Object

When UNICORE was designed to address seamless computing, it became obvious that this goal could realistically be realized only through abstraction. The operations that are to be performed on behalf of the user on a target system are specified by the

user in a system independent way using UNICORE's graphical client. Examples are: perform a Fortran compilation, produce highly optimized code, transfer data between two systems, or to show the status of a job. The actions are likewise represented in a system independent way as Abstract Job Objects, which the client creates. These abstract descriptions are translated into system specific actions during the incarnation process. This approach has several advantages:

- The user need not concern herself with system specific conventions and can execute jobs at a foreign installation without having to learn their local conventions.
- The installation can make changes without having to re-train all users.
- The installation can add new systems and have them used in the best known way by all users, not only those who take the time to learn the intricate features.

The Abstract Job Object (AJO) is the basis for the platform and site neutral specification of requests for computational and data resources. The AJO is a conceptual representation of a "Job" as a collection of possibly interdependent operations to be carried out on various computational and data services at collaborating sites. The object-oriented structure and syntax of the AJO are intended to support a specification largely independent of hardware architecture, system software interfaces, and site-specific operational rules. In this way, seamless descriptions of user's work can be created without interfering with site autonomy.
As with the platform-independence realised in the Java virtual machine, complete uniformity of the AJO may in some cases conflict with performance goals. There may, for instance, be architectural features of HPC platforms for which it is not possible to construct an abstraction that maps uniformly to all HPC platforms. When the detailed control of such features is necessary to achieve the maximum performance of the architecture, a performance versus seamlessness trade-off may result. For example the AJO class CompileTask, which includes parameters strongly affecting machine performance, defines uniform concepts such as optimisation levels, to which machine-specific parameters can be assigned. More work will be required in this area. However, the UNICORE consortium have agreed that the following abstract notion of an abstract job is rich enough to describe most of the functions necessary in a batch oriented supercomputing environment. It is not, however, intended to replace the operating system or its batch subsystem, but to inter-operate with them.
Although the complete definition is beyond the scope of this paper, the following summary should illuminate its structure. The AJO is composed of Java Object Classes, as follows:

- **AbstractAction** is an Abstract Class; the superclass JobGroup and AbstractTask.
  - A **JobGroup** is a directed acyclic graph (DAG) of AbstractActions. The DAG structure is the mechanism for specifying dependencies among its actions.

An **AbstractJob** is a specialized JobGroup that provides security services. It acts as the container for the top level UNICORE Job and for sub jobs to be run at remote sites.

- **AbstractTask** is an Abstract Class; the superclass of all task classes.

ExecuteTask is an Abstract Class representing a program execution.

- **CompileTask:** A class of object which performs a compilation.
- **LinkTask:** A class of object which executes a linker.
- **UserTask:** A class of object which executes a user's program.

FileTransfer is an Abstract Class representing transfer of files.

- **ImportTask**: A class of object which transfers a file from a storage server, which the user is authorised to access, into the temporary UNICORE space (Uspace) in which ExecuteTasks will run.
- **ExportTask**: A class of object which transfers a file from the Uspace to an external storage server.

Note that this basic AJO object structure is both hierarchical and recursive. Since a JobGroup contains a DAG of AbstractActions and is also a subtype of AbstractAction, an AJO may itself contain AJOs, called sub-AJOs, which may execute at the same or different site.

In addition to this basic structure, the AJO model is supported by an architecture of object hierarchies for representing such things as users, files, priorities, projects, administration parameters, security specifications, and resource requirements. For example, a CompileTask includes a set of resource requirements, a priority, source files, object files, and uniform notions specifying levels for optimisation and compiler output. All these are translated to system/site-specific representations by the NJS at the target site.

Given the diversity of HPC systems, the variety of site administrative policies, and the breath of user requirements, the AJO architecture models the process of job submission and control remarkably well. The aspect of the AJO that makes this both a challenge and ensures its success is that the AJO aims to model the problem domain rather than provide a language to describe all possible situations. In this way, the AJO is more like the CORBA infrastructure of precisely defined services than the resource description language RSL of Globus [1].

The choice of a Java Class Library to represent AJOs has many advantages, not the least being that it was directly usable by the implementation language Java, practically a must for project in 1997. This has also allowed the easy extension of services and abstractions without disrupting the rest of the UNICORE architecture. In the UNICORE Plus project, with its emphasis on deployment of frequently used applications, application specific services are being developed. The AJO has been extended to support these developments with the addition of a new sub-class of AbstractJobObject, called Idiomatic. This subclass provides a collection of frequently used job templates. This allows the development of application specific clients, without the need for a detailed understanding of the complexities required to construct recursive UNICORE jobs.

The AJO architecture and the Unicore Protocol Layer are open interfaces to which implementers are welcome to develop alternative or specialized implementations. The source and documentation are under the control of the UNICORE Forum and can be found at www.unicore.de.

3.4 The UNICORE Client

Graphical user interfaces are important to allow end-users to access Grid resources in a seamless fashion. Batch submission through web interfaces has been developed in several projects, for example Websubmit [9]. The client development in UNICORE focuses on functionality, consistency, and portability. The primary role of the client is to guide the user to create complex jobs consisting of tasks that may execute on multiple systems and/or at different remote sites. The user can combine previously created tasks or job groups to build new jobs, incorporate legacy scripts, specify temporal and data dependencies between tasks, and instruct UNICORE to transfer data. A client component, called the *Job Preparation Agent (JPA)*, creates an Abstract Job Object which can be submitted to a UNICORE gateway once the Job is syntactically correct.

The latter is an important design objective; the client prevents submission of obviously incorrect jobs, thus avoiding the all too familiar pitfall of batch processing - a user waiting many hours for the completion of a job only to find out that the job could not be run because the specified resource limits exceeded the those allowed at the target system. This consistency check occurs, for example, when the user changes the destination. The JPA verifies the values and gives the user the option to adjust the resource requirements. In addition to these client side checks, the NJS supports an extensive framework of defaults, which further increase the chances of successful execution.

A second component of the client is the *Job Monitor Controller (JMC),* which allows the user to monitor the progress of submitted jobs. An intuitive color scheme indicates the status of the job. Icons representing the job groups and tasks are marked in green (completed successfully), red (failed), yellow (executing), or blue (queued). The user may drill down to the individual task to obtain its status, retrieve stdout or stderr or data created by the job. In addition, the user may terminate executing and remove or hold queued jobs.

Consistency of the graphical interface is achieved by adhering to the JAVA Look and Feel Design Guide [10] and by sharing components developed by the partners. This approach was selected over of trying to present a Windows look-and-feel to PC users and a Unix desktop to Unix users.

When the first UNICORE project was started in 1997 there was great hope that the promise of Java ‘Write once, run anywhere’ would be fulfilled soon. The client was implemented as a signed applet which allowed rapid deployment of functions to end-users. However, the developers learned the hard way that subtle differences exist between Java implementations on different platforms and some not so subtle differences between browsers. At one time X.509 certificates would either be recognized Netscape or Internet Explorer, but not by both. This forced the project to restrict client support to Netscape, because it is available on Windows and Unix. Based on this experience, the UNICORE Plus project decided to implement the client as a Java application for higher stability, portability, and performance. The results are very promising. There is of course a small price to pay: a tool has to be included into the UNICORE distribution to automate the installation of new client versions as much as possible.

3.5 Application Support in UNICORE

UNICORE's key features, seamless access to remote resources and transparent data transfer, can be exploited by many applications. It would, however, be impractical to create a full new client for each potential application. The project develops mechanisms to generate graphical interfaces for new applications and to integrate applications with existing GUIs into UNICORE. The latter is prototyped for codes like Fluent and STARCD. To support new applications, a wizard has been designed with input from a scientist performing simulations using the Car-Parinello Molecular Dynamics code (CPMD). The wizard guides the user through the process of specifying the relevant parameters for the simulation. Although CPMD is primarily used as an example to demonstrate the technique, it is a worthwhile endeavor in itself. In addition, the researchers can exploit UNICORE to run the CPMD simulation as part of a more complex job including visualization at the workstation or post-processing on a different system. The final project result will be a toolkit to ease integration of existing and new applications into UNICORE.

3.6 System Integration and Security

New software, especially if it is still under development, is accepted in production environments only if it provides substantial additional value and if it requires almost no changes to existing practices or procedures. This might not be true outside Germany. Nevertheless only exposure and usage in production will demonstrate the viability of a new software. Therefore, one of the key objectives of UNICORE is to coexist with exiting administration and security policies.

The task of combining two independently administered, multi-user Unix systems, possibly at different sites is not a challenge, it is a nightmare. The necessary changes to external user names and internal Unix uids and gids creates work and frustration for both the user and the administrator. To require that installations agree on global user names and Unix identification would have certainly prevented deployment. Therefore, UNICORE decided from the beginning to rely on the newly standardized X.509V3 certificates for grid-wide user identification and for authentication of users, systems, and software. An X.509 certificate issued by an accepted Certificate Authority (CA) is a mandatory requirement to access UNICORE resources at participating installations. The NJS maps the user's certificate to a login name for the target system controlled by the NJS. This means that installation need not change their naming conventions, and users need not bother with different identities at different systems.

Hereby, installations retain full autonomy over their resources at two levels: only users that are registered in the login data base may submit jobs at all, and at the second level, the system dependent regulations and limits, like user or project quotas for a particular machine, may be used unchanged. UNICORE explicitly assumes that grid resources have to be accounted for and that the responsibility and accountability remains with the installation.

X.509 certificates are also used to encrypt the communication between UNICORE components, for example a client and a gateway or an NJS and its peer at a different site. Gateways coexist with firewalls. AJOs in transit are signed with the user's private key to protect against tampering. UNICORE implements end-to-end security and privacy.

The decision to use certificates has proved to be excellent one. The expectation, however, that e-commerce would proliferate certificates during the lifetime of the project, to such an extend that everybody would use certificates and that the necessary Public Key Infrastructure (PKI) would be well established, did not come true. As a consequence the project had to establish its own CA under the umbrella of the DFN-PCA [11].

3.7 New Functions

The first UNICORE project restricted itself to demonstrating the basic set of functions. It included a simple, static – but never the less sufficient – resource model. The UNICORE Plus project extends this to a flexible and extensible resource model, which allows the addition of new resource categories in a non-disruptive way and which supports dynamic resource information, such as system load. The interface will enable UNICORE to exploit resource brokers; their implementation, however, will occur as part of the EUROGRID project.

UNICORE's data management is being extended to support specialized servers to move data between different sites and in and out of archives, like HPSS or ADSM, using the full UNICORE security model. Initially, these servers will be run parallel to the context of job submission, be tested independently, and then be evaluated prior to full integration into UNICORE.

There are many scientific applications which require repetitive or conditional execution of jobs with different parameter sets to run computational experiments. To support these scenarios the AJO is enhanced to handle extended control of tasks; graphical support will be provided in the UNICORE client to let the user formulate loops, if-then-else constructs, case statements, and specialized error handling.

4 The EUROGRID Project

The UNICORE project restricted its goals to meet budget and resources limitations, expecting that additional funding could be secured for further development. As one of the first European Grid projects the European Commission decided to fund EUROGRID (EC contract IST 20247) for a period of three years, to demonstrate grid technology based on UNICORE using the existing European network infrastructure. The functions to be added include interactive access, resource brokering, and application services. The project is to demonstrate a European Bio-Grid, a Meteo-Grid, and a CAE-Grid, with the results being commercialized by Pallas GmbH.

5 Globus and UNICORE

Among the international Grid projects, Globus has a well established position. Its goals and functions are described for example in [12]. When the UNICORE project was proposed to BMBF in 1997 it decided not to repeat the development of Globus, but to create a software infrastructure focusing on the practical needs of users of HPC Centers. Today, the strength of Globus is clearly in providing layers of grid services,

that enable developers to build additional layers on top of Globus. UNICORE and Globus complement each other. To demonstrate this, a project was started between Argonne National Laboratory and Forschungszentrum Jülich, to develop an interface between Globus and UNICORE, leaving the core components of the two environments unchanged. UNICORE will serve a batch submission portal to Globus resources. A Globus Grid, or part thereof, would in turn assume the role of a UNICORE site making Globus services available to UNICORE clients. Once this has been demonstrated and the respective similarities and differences are fully understood at the technical level, optimized technical solutions can be developed.

6 Summary

UNICORE is demonstrating that Grid computing can be used to solve real problems in day-to-day production. Its particular strength is the seamless, secure and intuitive access to remote heterogeneous resources over the Internet. UNICORE's key achievements, the Abstract Job Object – an open specification – as well as security and authentication based on the X.509 standard, make it a viable implementation of a Grid computing environment.

References

1. Foster and C. Kesselmann, (Ed.) The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufman Publishers, 1998
2. http://dast.nlanr.net/Clearinghouse
3. http://www.w3c.org/Jigsaw/Doc
4. Wissenschaftsrat: Empfehlung zur Versorgung von Wissenschaft und Forschung mit Höchstleistungsrechenkapazität, Wissenschaftsrat, Drs. 2104/95, 7.7.1995
5. F. Hoßfeld, et. al. Verbund der Supercomputer-Zentren in Deutschland – eine Machbarkeitsanalyse Jülich, 1997
6. D. Erwin, (Ed.) UNICORE – Uniformes Interface für Computing Ressourcen (Final report – in German) http://www.unicore.org, 2000
7. http://www.fz-juelich.de/mcdc
8. Romberg, M. The UNICORE Architecture–Seamless Access to Distributed Ressources
   Proceedings of the Eight IEEE International Symposium on High Performance Computing, Redondo Beach, CA, USA, August 1999, Pages 287-293
9. http://www.itl.nist.gov/div895/sasg/websubmit/websubmit.html
10. http://java.sun.com/products/jlf
11. http://www.cert.dfn.de/dfnpca/
12. http://www.globus.org