The UNICORE Grid infrastructure

Mathilde Romberg
Research Center Jülich, Central Institute for Applied Mathematics, D-52425 Jülich, Germany
Tel.: +49 2461 61 3703; Fax: +49 2461 61 6656;
E-mail: m.romberg@fz-juelich.de

UNICORE (Uniform Interface to Computer Resources) is a software infrastructure supporting 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 the services provided.

Keywords: Grid infrastructure, HPC portal, Abstract Job, seamless access, Java, UNICORE

1. Introduction

The concept of Grid computing evolved during the past decade based on distributed computing and metacomputing. With the growing speed of network connections it became feasible to think of accessing distributed compute power, applications, and data as easily as one accesses the electrical power. The term computational grid is defined in [5] as the hardware and software infrastructure which provides dependable, consistent, pervasive, and cost-effective access to high performance computing resources. Research fields in this area are programming models and their support through scheduling, quality of service, security, and user interfaces. A variety of projects are dealing with application-specific interfaces to provide users with seamless access to distributed computer resources. Users can thus describe the job in application-specific terms instead of dealing with the system and site-specific idiosyncrasies. An example of a project in this field is WebSubmit at NIST (see [9]). It develops a common Web-based interface for applications (for example Gaussian94) on multiple heterogeneous systems. Other Grid computing projects deal with programming models for distributed applications, scheduling, quality of service and similar functions. Two of the outstanding Grid projects are Legion at the University of Virginia (see [7]) and Globus at Argonne National Laboratory and University of Southern California Information Sciences Institute (see [6]). Legion is developing an object-oriented metasystem which serves as a basis for applications. The goal is to provide a single coherent virtual system which is scalable, easy programmable, fault-tolerant, and secure and which supports site autonomy. The Globus project is focusing on the development of a metacomputer toolkit for Grid applications. The toolkit contains different services as for example resource control and process scheduling, communication, data access, and security. These services are used to build up more complex metacomputing services like job schedulers or support tools for parallel applications. Both projects focus on metacomputing at the application level. They combine a set of distributed systems to process one huge single application. Users have to adapt their application to match the supported programming model and services.

The various concepts and solutions have motivated researchers to set up a forum to discuss Grid computing aspects. The Grid Forum (see [16]) and the European Grid Forum (see [13]) were therefore founded and have recently joined together to become the Global Grid Forum (see [15]). These initiatives intend to work jointly on the development and dissemination of best practices, on guidelines for implementations, and on standards. The goal is to establish an integrated Grid architecture which can serve as a guideline and basis for further research and development.

UNICORE (Uniform Interface to Computing Resources), a German national research project, started in mid 1997 to develop a software infrastructure for seamless access to distributed supercomputer resources. The Motivation for the project (see [17]) was 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 resources the users need to learn about these differences. UNICORE

Scientific Programming 10 (2002) 149–157
ISSN 1058-9244 / $8.00

is designed to overcome the additional effort (further information on the project's background is given in [2]).

The main objectives of the project are security, seamlessness, ease of use, portability, and minimal interference with local site administration. The project task is the design of a system which fulfills the objectives and the implementation of a prototype. In contrast to the other projects in 1997, UNICORE selected an X.509-based security model which is now the standard within the Grid community. Seamless access calls for an abstract job model which allows a system-independent job description. This idea is unique among the various Grid projects. The Abstract Job Object (AJO) is the core part of UNICORE, generated at the user client and processed into real jobs for a target system by the server. Within the Global Grid Forum's classification of Grid architectures UNICORE is a so-called stovepipe system covering all layers of the Grid architecture; most of the other projects provide solutions for a small set of layers (i.e. providing low-level services).

In the remainder of this paper Section 2 will describe the architecture developed, the problems being investigated, and the design aspects including the lessons learned. Section 3 examines the user functions available, how transparency is achieved, and how jobs are mapped to resources. The experience gained by users and administrators is described in Section 4 together with the questions still to be answered. Finally, an outlook on further development plans is given.

2. The UNICORE architecture

2.1. Requirements

The architecture has to satisfy a wide range of requirements from areas like security, user-friendliness, administration, and site environments. The architecture must

1. use standards where available,
2. provide strong user authentication and authorization,
3. ensure data integrity,
4. provide single sign-on,
5. cooperate with firewalls,
6. cooperate with local user administration,
7. require only minimal privileges for components on the target systems,
8. allow for abstraction/seamlessness,
9. support users, put minimal burden (installation, configuration, . . .) on users,
10. support various platforms,
11. provide resource information, and
12. not rely on transitive trust between sites/servers.

Standards or de-facto standards (1) give solutions for most of the requirements. X.509 allows for user authentication, data integrity, and single sign-on (2, 3, 4). Using standard protocols like SSL for communication over untrusted networks and socket-to-socket communication between fixed, predefined ports over the trusted intranet means that cooperation with firewalls can easily be achieved (5). With X.509 user certificates as the unique user identification within UNICORE cooperation with the local user administration (6) can be realized by mapping the user certificate to the local user identification. The necessary information has to be provided by the PKI (Public Key Infrastructure). One essential quality of the target-system-related part of the UNICORE system is that it can act on behalf of a user. Therefore part of the server has to have the setuid privilege; it does not need any other privileges (7).

Seamlessness, user support and flexibility with respect to supported platforms (8, 9, 10) can be tackled with Web-techniques and Java. The question which has to be answered here is whether an applet or an application is the appropriate solution for the user client. How to provide the resource and service information (11) is another question. An information service is necessary which allows access to static and dynamic resources and service information of the target systems. The research here is in a language for resource and service description (see [10]). In addition, XML and LDAP are standards which are evaluated. Regarding security aspects X.509 is the standard which allows for user authentication, data integrity, and single sign-on. The question still to be answered is how to do client – server – server communication without relying on transitive trust between servers. How can it be ensured that the job and all its parts prepared by a user are not modified on the way from the user to its final destination (12)?

2.2. Design

The UNICORE architecture consists of three tiers: user, UNICORE server, and target system tier as shown in Fig. 1 (see also [1,11]). User and server tiers are Java applications while the system tier is presently implemented in perl as Java is not available on all supercomputer platforms. The security architecture is based on the Secure Socket Layer (SSL) protocol (see [3]).

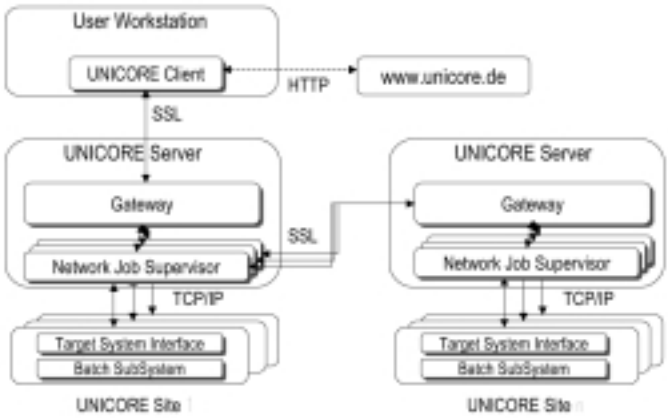

Fig. 1. Architecture overview.

SSL is used for communication between the components which talk to each other over public networks (Client – Gateway, Network Job Supervisor – Gateway). 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 are communicating with the actual user (or program) he or she (or it) claims to be. By default certificates signed by unknown signers are not accepted.

2.2.1. Abstract Job Object

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 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. [2] explain the AJO and its classes in more detail.

The AJO reflects the underlying recursive job model. A UNICORE job consists of

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

Each UNICORE job is assigned a UNICORE job directory (Uspace) 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 the data needed for the job execution has to be imported from permanent file space into the job directory. All the 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 that to be exported when constructing the job. The transfers are done by UNICORE and are transparent to the user.

The AJO with its recursive structure gives the clue to solving the problem of making sure that a job and all its parts prepared by a user are not modified on the way from the user to the destination without relying on transitive trust between the servers at the different sites. The user client signs all embedded AJOs in the main AJO as well as the main AJO with the user's X.509 certificate. A UNICORE server which receives an AJO unpacks it leaving all embedded AJOs which are destined for other sites as they are and forwards them to the appropriate server. Each UNICORE server can therefore test that the AJO is the original without having to trust other UNICORE servers.

2.2.2. Client

The user tier consists of the graphical user interface. It offers the functions to prepare and control UNICORE jobs (for details see Section 3) and to set up and maintain the user's security environment. The UNICORE user's X.509 certificate is his or her UNICORE user identification. It is maintained by the user interface application in an encrypted database. The user interface also needs to know about the Certification Authority which signed the user and the Gateway certifi-

cates. 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 JPA offers the functions for generating and submitting a UNICORE job while the JMC allows to check job status, control a job, and retrieve job output.

The decision to have the user client as an application instead of an applet is based on the following: Java is a good choice to achieve portability. However, the Java Virtual Machines (JVM) on different systems are different. Programmers have to be conservative in using new features. The use of applets introduces another source for losing portability. During the first project phase (1997–1999) we developed the user client as a signed applet and experienced that browsers do not necessarily support all Java classes. As an example, Netscape did not support the Java Swing classes which facilitate the creation of the graphical user interface. In addition different browsers handle signed applets differently, so we had to restrict the use to the Netscape browser. The user client applet had to be signed as it needs access to local data on the user's workstation or PC and because it has to connect to different network sites (the Gateways at the different UNICORE sites (Usites)). To reduce this complexity in code writing and bug tracing it was decided to create the user client as an application. The only disadvantage of this solution is that the users have to install and maintain the client locally.

www.unicore.de serves as the root node for resource information. Currently it holds the addresses of the Gateways of the available Usites. Using these addresses the client connects to the Gateways to get the list of available Vsites and their resources. Right now a set of static resource information is available. It includes the number of CPUs or PEs, amount of memory, CPU or connect time, and amount of file space (see [10]).

2.2.3. Gateway

The Gateway is the first part of the UNICORE Server tier. It ensures the user authentication, secure communication between client and server, and provides the client with information about the resources available at the site. It also talks to the Network Job Supervisor (NJS) servers at the site so that jobs and data, status requests and control commands for further processing are sent and data is received to make it available to the user. UNICORE uses the UNICORE Protocol Layer (UPL) to send the AJO to the Gateway which hands it over to the NJS controlling the specified Virtual Site (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 Fig. 2. The Gateway and NJS communicate via sockets.

2.2.4. Network Job Supervisor

Each Network Job Supervisor (NJS) controls one Virtual Site (Vsite) which is a single system or a cluster of systems sharing the same userids and file space. The NJS fulfills the following tasks:

- analyzing the AJO (representing the UNICORE Job) and extracting parts to be executed locally;
- mapping the UNICORE userid to the local userid for the target system (authorization);
- translating the local jobs contained in the AJO into real batch jobs for the target system;
- sending job groups to be executed at other UNICORE sites to the corresponding Gateway;
- providing local resource information to the Gateway;
- ensuring the necessary file transfer;
- providing job status information and job output.

The NJS first checks whether the AJO is a correct AJO from the user who signed it. It unpacks the AJO with the user's public key and analyzes it for parts to be sent to other target systems. These are sent 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 certificate 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 information about which certificate belongs to which local userid is kept in the UNICORE user database (UDB). The UDB has to be maintained at each Usite for each NJS. This mechanism is necessary to provide a single sign-on environment. The user authenticates his or herself once in the client and the uidmapping at the different Vsites takes care of the authorization without the user having to reauthenticate. The NJS translates the abstract job definition into batch

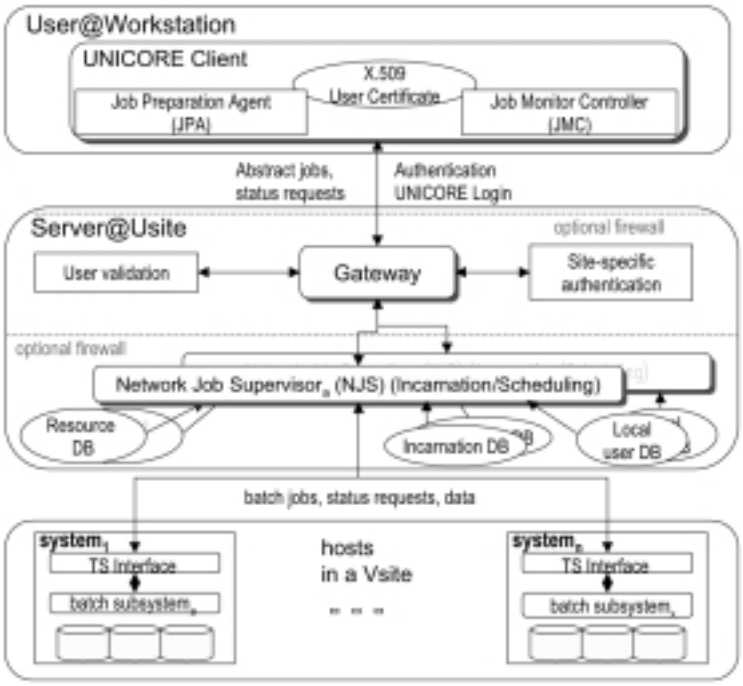

Fig. 2. Detailed architecture.

jobs for the destination system and sends them to the Target System Interface (TSI) according to the dependency graph. The translation of the AJO into a real batch job is table-driven. Each NJS holds an incarnation database (IDB) which contains the abstract terms and their translation for the target system. It is therefore possible to let the user specify the job in an abstract way without being concerned about system and site specific details.

2.2.5. Target System Interface

Each of the systems in a Vsite 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. Besides the incarnation database it is the second instance for realizing transparency. The TSI knows about the commands and their syntax for submitting jobs, copying data, asking for job status, and controlling jobs.

3. The user functions

Within the UNICORE environment the user has a convenient way of using distributed computing resources without having to learn site or system specifics. It is all done in a seamless way. The user specifies the job requirements (e.g. application, resources, command options, input and output files, etc.) independently of the target system and site the user has selected. As described above, the NJS at the destination site 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 obtain access to the resources at the various UNICORE sites. At the top level the graphical user interface offers the functions for maintaining the security settings, preparing UNICORE jobs, and monitoring them.

A UNICORE job can be composed of multiple parts which can be executed asynchronously or in dependence 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;
- CPMD task, to prepare Car-Parrinello Molecular Dynamics applications;
- transfer tasks, to specify necessary file transfers between two parts of a job to be run at different Vsites;
- job groups, to build sub-jobs for other target systems.

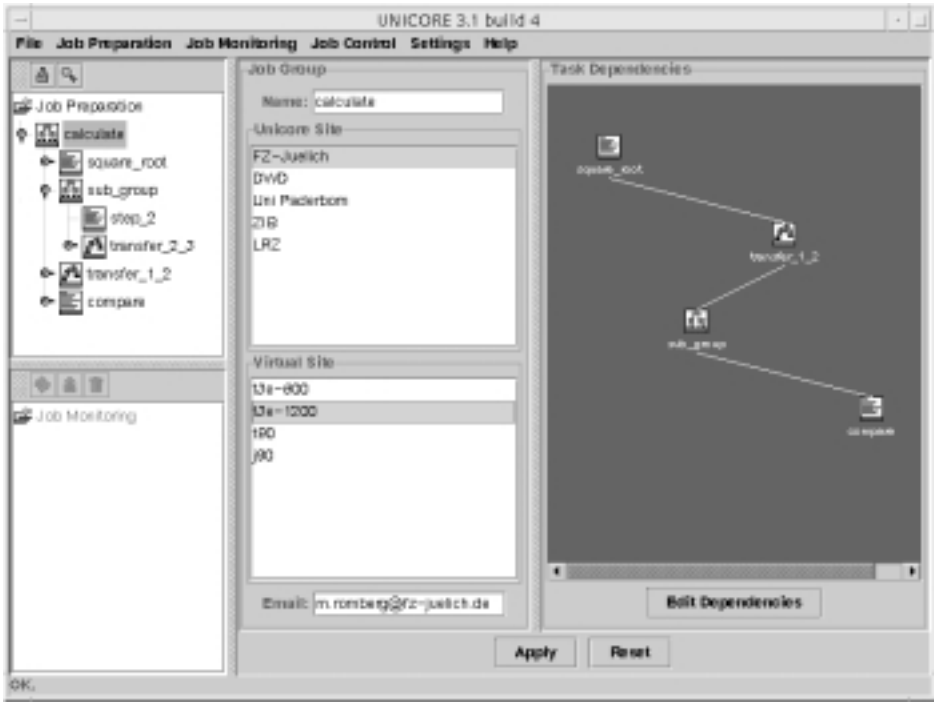

Fig. 3. Job preparation agent.

Each UNICORE job and each job group contains general information for the job:

- the job name;
- the target system, which can be selected from all available Usites/Vsites;
- the user's e-mail address, where the system should send messages to.

Figure 3 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 job group icon with the job name is highlighted. The center part of the display shows the general information for the job group.

The UNICORE job consists of four different job steps to be executed on two different target systems (two job groups: *calculate* and *sub_group*). *square_root*, *step_2*, and *compare* are script tasks as can be seen from the icon. The other two are transfer 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 is activated for usage. The icons on the right are arranged according to the specified dependencies which reflect the necessary synchronization between the job steps. The dependencies are already defined via the *Add Dependency* button at the bottom.

Seamlessness for the user is achieved by abstraction. The interface offers selection menus with predefined values for e.g. the kind of shell or job priority. The selected values are transferred to and translated by the NJS to the appropriate value for the target system.

As a reference Fig. 4 shows a picture of the AJO corresponding to the example of a job given above. Each box represents an AJO. The outermost box contains the whole UNICORE job while the other bold framed box contains the sub job group. The small boxes contain the AJOs for the tasks which are the job steps generated from the user input. The first task in every job group is the generation of a job directory (the Uspace). Import of data, execution of the user application, and transfer follow. The internal dependencies between the tasks are represented by the arrows. Each job group has a clean-up section with the tasks 'spool output' to make the output available to the user and 'remove job directory'. The tasks are incarnated into shell scripts executed on the target system either directly (i.e. 'generate job directory') or as a batch job.

The Job Monitor Controller (JMC) offers functions for displaying the job status, listing and saving job output, and for killing jobs. Figure 5 shows the JMC interface with a list of Vsites and the job submitted

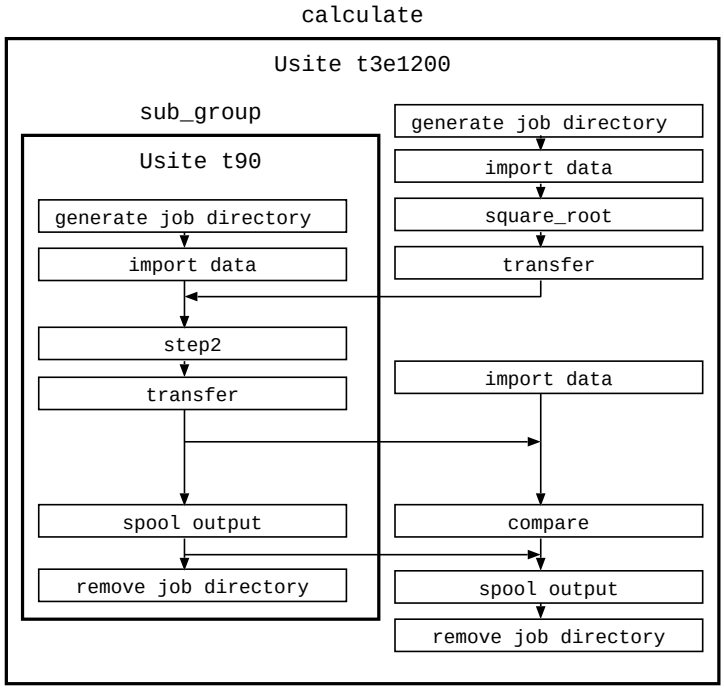

Fig. 4. AJO for job example.

in the example above. The status display lists jobs sorted by target sites. The job icons on the left can be expanded to show the complete job tree. Selecting an icon on the left results in displaying the standard output and error files of that component. The icons are marked with colors to represent the current status within UNICORE: blue – queued, yellow – running, red – abnormally ended, green – finished successfully. The user has access to the job status and the output files until he or she explicitly removes them from the status display.

4. Experiences

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 database for the translation of the abstract job specification into a real batch job for the target system is easily adaptable to new platforms: The contents of the translation table (incarnation database) have to be filled with the new target values and the TSI has to be adapted (command names and options for job submission, status request, move, copy, . . .). Currently CRAY T3E, CRAY T90 and J90, NEC SX-4, Fujitsu VPP 700, Siemens hpcLine, Hitachi SR2201, and IBM SP3 are integrated and the batch subsystems NQS, PBS, CCS, and LoadLeveler. It is deployed at the project partners' centers. Up to now (August 2001) a few test users have been "playing" the system. They use it for submitting existing job scripts and for preparing CPMD jobs. The general feedback is that the system is easy to use and that especially the application-specific support for CPMD helps a lot (see [8]). The possibility of submitting jobs to different targets is appreciated but not yet used to any great extend. The client performance is rated as good.

As mentioned in Section 2.2.2 during the first project phase the user interface had been implemented as a signed applet which led to problems with portability and down-load performance. Also in the first phase a full-blown Web server had been used to host the Gateway (as a servlet) and an existing batch system had been modified to play the role of the NJS. The advantage was to have a running prototype with good functionality within a short time frame. However, there were quite a few reasons to redesign this and implement all components ourselves and in Java. The design presented here offers all the necessary functions. The system is now lean and easy to control. The table-driven mechanism

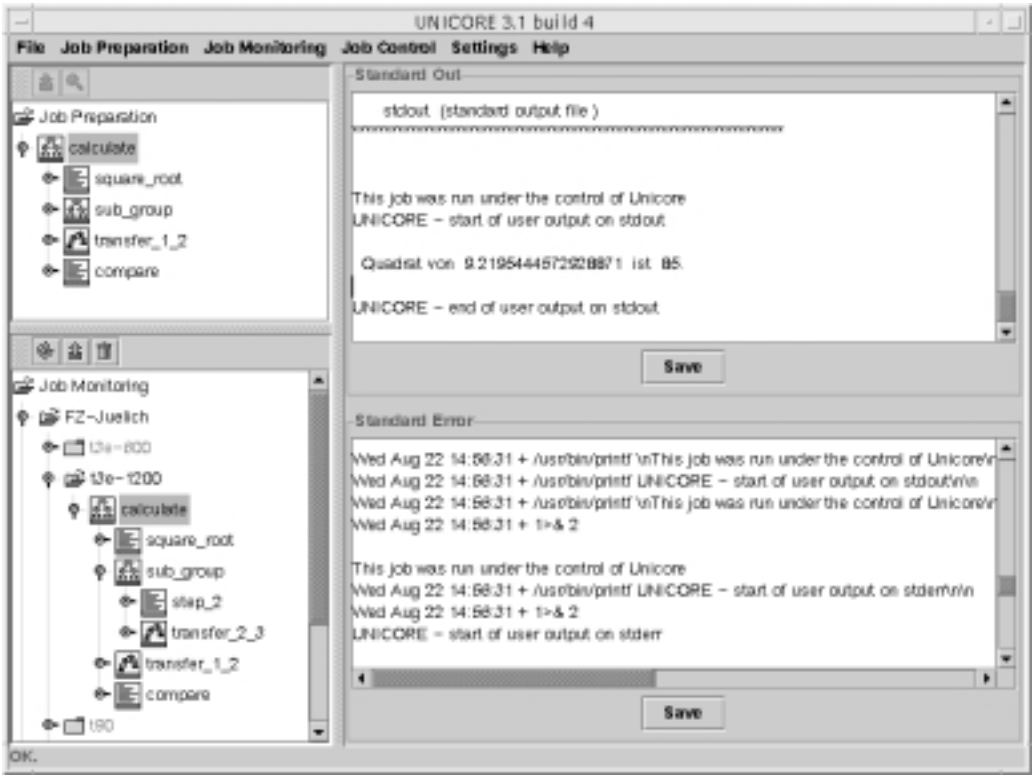

Fig. 5. Job monitor controller.

for incarnation provides the flexibility to easily adapt it to other target systems.

The current version of the UNICORE system (V3.0) is also used as the basis for the EU-funded project EUROGRID (see [14]) which started in November 2000. It will establish a European GRID network of leading High Performance Computing centers from different European countries.

5. Outlook

Within the UNICORE Plus project (see [18]) the UNICORE system is going to be extended and put into production at the partner sites. Areas of development in the project frame are: administration, resource and service description, application-specific interfaces, data transfer, extended job control, and metacomputing. For the application-specific interfaces MSC-NASTRAN, FLUENT, and STAR-CD have been selected as candidates. It is also planned to build a generic interface to allow for easy 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.

One of the most important issues is 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 of UNICORE jobs but using the UNICORE security mechanisms.

In the area of metacomputing on the 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 metacomputer applications.

Other efforts to enhance UNICORE are being undertaken in the EUROGRID project (e.g. resource broker) and in a joint project of Research Center Juelich and Argonne National Laboratory which deals with Grid interoperability.

Acknowledgments

The UNICORE project ([17]) was funded by the German Federal Ministry of Education and Research (BMBF) during the grant period August 1, 1997 to December 31, 1999 under grant id 01 IR 703. The UNICORE Plus project ([18]) is being funded by BMBF for three years (2000–2002) under grant id 01 IR 001.

The results presented in this paper have been obtained by the collaboration of all UNICORE partners: 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 of Karlsruhe (RUKA), Paderborn Center for Parallel Computing (PC^2), Konrad Zuse Center for Information Technology Berlin (ZIB), Center for High Performance Computing at TU Dresden (ZHR),[1] Fujitsu European Center for Information Technology (fecit),[2] and Genias Software GmbH, Regensburg.[3]

References

[1] J. Almond and D. Snelling, UNICORE: Uniform Access to Supercomputing as an Element of Electronic Commerce, *Future Generation Computer Systems* **613** (1999), 1–10.

[2] D.W. Erwin and D.F. Snelling, UNICORE: A Grid Computing Environment, *Proccedings of Euro-Par 2001*, Springer LNCS 2150, August 2001, pp. 825–834.

[3] J. Feghhi, J. Feghhi and P. Williams, *Digital Certificates – Applied Internet Security*, Addison-Wesley, 1998.

[4] I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufman Publishers, 1998.

[5] I. Foster and C. Kesselman, Computational Grids, in: *The Grid: Blueprint for a New Computing Infrastructure*, I. Foster and C. Kesselman, eds, Morgan Kaufman Publishers, 1998.

[6] I. Foster and C. Kesselman, Globus: A Metacomputing Infrastructure Toolkit, *Intl. J. Supercomputer Applications* **11**(2) (1997), 115–128.

[7] A. Grimshaw, W.A. Wulf and the Legion team, The Legion Vision of a Worldwide Virtual Computer, *Communications of the ACM* **40**(1) (January 1997), 39–45.

[8] V. Huber, Supporting Car-Parrinello Molecular Dynamics Application with UNICORE, *Proceedings of the Computational Science – ICCS 2001 International Conference*, (Part I), San Francisco, May 2001, pp. 560–566.

[9] R. McCormack, J. Koontz and J. Devaney, *WebSubmit: Web-based Applications with Tcl*, NISTIR 6165, June 1998.

[10] A. Reinefeld, H. St′uben, T. Steinke and W. Baumann, Models for Specifying Distributed Computer Resources in UNICORE, First European Grid Forum Meeting, *Proceedings of the ISThmus 2000 / EUNIS 2000 Conference*, Poznan, April 2000.

[11] M. Romberg, The UNICORE Architecture: Seamless Access to Distributed Resources, *Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing*, August 1999, pp. 287–293.

[12] http://www.cert.dfn.de/dfnpca.

[13] http://www.egrid.org.

[14] http://www.eurogrid.org.

[15] http://www.globalgridforum.org.

[16] http://www.gridforum.org.

[17] http://www.fz-juelich.de/unicore.

[18] http://www.fz-juelich.de/unicoreplus.

[19] http://www.unicore.org.

[1] during the second project phase 2000-2002 only

[2] as a sub-contractor of Pallas

[3] during the first project phase 1997-1999 only