

```
public static IUASSecurityProperties getSecurityProperties()  
    if (securityProperties == null) securityProperties = new IUASSecurityProperties();  
    return securityProperties;
```

```
public static final String TSF_URI = "http://www.unicore-service.org/";
```

```
    @Target SystemFactory
```

```
    <service name="{TSF}" wsrf="true">
```

```
    <interface class="{TargetSystemFactory.class.getName()}">
```

```
    <implementation class="{TargetSystemFactoryHomeImpl.class.getName()}">
```

```
    </implementation>
```

```
    </interface>
```

```
    </service>
```

```
    </target>
```

```
    </binding>
```

```
    </binding>
```

```
    </binding>
```

```
    </binding>
```

```
    </binding>
```

```
    </binding>
```

```
    </binding>
```

```
    </binding>
```

```
    </binding>
```

```
    </binding>
```

```
    </binding>
```

```
    </binding>
```

```
    </binding>
```

```
    </binding>
```

```
    </binding>
```

```
    </binding>
```

```
    </binding>
```

```
    </binding>
```

```
    </binding>
```

```
    </binding>
```

```
    </binding>
```

```
    </binding>
```

```
    </binding>
```

```
    </binding>
```

```
    </binding>
```

```
    </binding>
```

```
    </binding>
```

```
    </binding>
```

```
    </binding>
```

```
    </binding>
```

```
    </binding>
```

```
    </binding>
```

```
    </binding>
```

```
    </binding>
```

```
    </binding>
```

UNICORE

UNICORE Summit 2011

Proceedings, 7–8 July 2011 | Toruń, Poland

Mathilde Romberg, Piotr Bała, Ralph Müller-Pfefferkorn,
Daniel Mallmann (Editors)

Forschungszentrum Jülich GmbH
Institute for Advanced Simulation (IAS)
Jülich Supercomputing Centre (JSC)

UNICORE Summit 2011

Proceedings, 7–8 July 2011 | Toruń, Poland

Mathilde Romberg, Piotr Bała, Ralph Müller-Pfefferkorn,
Daniel Mallmann (Editors)

Bibliographic information published by the Deutsche Nationalbibliothek.
The Deutsche Nationalbibliothek lists this publication in the Deutsche
Nationalbibliografie; detailed bibliographic data are available in the
Internet at <http://dnb.d-nb.de>.

Publisher and
Distributor: Forschungszentrum Jülich GmbH
Zentralbibliothek
52425 Jülich
Phone +49 (0) 24 61 61-53 68 · Fax +49 (0) 24 61 61-61 03
e-mail: zb-publikation@fz-juelich.de
Internet: <http://www.fz-juelich.de/zb>

Cover Design: Grafische Medien, Forschungszentrum Jülich GmbH

Printer: Grafische Medien, Forschungszentrum Jülich GmbH

Copyright: Forschungszentrum Jülich 2011

Schriften des Forschungszentrums Jülich
IAS Series Volume 9

ISSN 1868-8489
ISBN 978-3-89336-750-4

The complete volume is freely available on the Internet on the Jülicher Open Access Server (JUWEL) at
<http://www.fz-juelich.de/zb/juwel>

Persistent Identifier: [urn:nbn:de:0001-2011120103](http://nbn-resolving.org/urn:nbn:de:0001-2011120103)
Resolving URL: <http://www.persistent-identifier.de/?link=610>

Neither this book nor any part of it may be reproduced or transmitted in any form or by any
means, electronic or mechanical, including photocopying, microfilming, and recording, or by any
information storage and retrieval system, without permission in writing from the publisher.

Preface

When the foundations of UNICORE were laid in the late 90's of the 20th century the "ancestors" intended to create a uniform interface to computing resources for a (small) number of computing centres. Today - in the era of eSciences and large distributed eInfrastructures - UNICORE has become one of the most innovative and major middlewares in Grid Computing serving users around the world. The UNICORE Summit is a unique event for users, developers, researchers, administrators and service providers to meet. They hear the latest news, share their experiences, present recent and intended developments, get new ideas for projects, find partners and discuss the future of UNICORE. The Summit has been held every year since it started in 2005.

The 2011 Summit took place in Torun on July 7 and 8. Torun is an ancient city in Northern Poland picturesquely located on the Vistula river. At about the same time when UNICORE was founded the medieval part of Torun was designated a World heritage by the UNESCO. The event was opened with an invited talk by Abdulrahman Azab from the University of Stavanger on the usage of R and UNICORE for medical research. In the following, 16 technical contributions gave more insights to various development and usage aspects. Additionally, posters were presented by the authors in a poster session on Friday. In the second invited talk Piotr Bala from Warsaw reported on the usage of UNICORE in the polish national Grid initiative PL-Grid. A unique feature of the Summit were the lively plenary discussions. The topics focused on the future ways UNICORE should go, with subjects like "UNICORE and Clouds" or "UNICORE and Portals".

We would like to thank all contributors for their presentations and papers as well as the organisers of the UNICORE Summit 2011 in Torun for their excellent work. We are sure every participant will keep this wonderful event in mind. More information about the UNICORE summit series can be found at <http://www.unicore.eu/summit/>. We are looking forward to the next UNICORE Summit 2012!

November 2011

Mathilde Romberg
Piotr Bała
Ralph Müller-Pfefferkorn
Daniel Mallmann

Contents

Preface	
<i>M. Romberg, P. Bała, R. Müller-Pfefferkorn, D. Mallmann</i>	i
UNICORE-Based Integrated Application Services for Multiscale Materials Modelling	
<i>I. Kondov, R. Maul, S. Bozic, V. Meded, W. Wenzel</i>	1
Serpens Suite for Kepler Workflow Orchestration System	
<i>M. Płóciennik, M. Owsiak, T. Żok, A. Gómez-Iglesias, F. Castejón</i>	11
Molecular Dynamics Science Gateway with Vine Toolkit Providing UNICORE Middleware Support	
<i>P. Dziubecki, P. Grabowski, T. Kuczynski, K. Kurowski, D. Szejnfeld, D. Tarnawczyk, M. Wolniewicz</i>	25
Advancing Cutting-Edge Biological Research with a High-Throughput UNICORE Workflow	
<i>R. Grunzke, R. Müller-Pfefferkorn, U. Markwardt, M. Müller</i>	35
A Web Framework for Workflow Submission and Monitoring via UNICORE 6 based on Distributable Scientific Workflow Templates	
<i>S. Bergmann, B. Demuth, V. Sander</i>	45
Integration of UNICORE 6 into the Gatlet Portal Framework	
<i>T. Antoni, A. Bender, B. Boegel, S. Bozic</i>	51
User Defined Runtime Environments Using UNICORE	
<i>B. Hagemeyer, K. Javaid</i>	57
Desktop Grids Opening up to UNICORE	
<i>M. Keller, J. Kovacs, A. Brinkmann</i>	67
NHiLA - Bridging the Gap Between .NET and UNICORE	
<i>D. Krott, M. Gerhards, S. Skorupa, V. Sander</i>	77
Running Local Jobs in the UNICORE Workflows	
<i>M. Borcz, R. Kluszczyński, P. Bała</i>	87
UNICORE Testbed of Physics Faculty of Saint-Petersburg State University	
<i>D. Kasterin, M. Stepanova</i>	95

EMI UNICORE Execution Services for Interoperability Across Middleware <i>M. Carpené, A.S. Memon, M.S. Memon, B. Schuller</i>	99
Compiling and Running Custom Code in the UNICORE Middleware <i>P. Piernik, P. Bała, K. Benedyczak</i>	109
Supporting Management of XACML Authorization Policies <i>T. Królikowski, P. Bała, K. Benedyczak</i>	117
Distributed Storage Management Service in UNICORE <i>T. Rękawek, P. Bała, K. Benedyczak</i>	125
UFTP: High-Performance Data Transfer for UNICORE <i>B. Schuller, T. Pohlmann</i>	135
UNICORE Data Management: Recent Advancements <i>K. Benedyczak, T. Rękawek, J. Rybicki, B. Schuller</i>	143

UNICORE-Based Integrated Application Services for Multiscale Materials Modelling

Ivan Kondov¹, Robert Maul^{1,2}, Stefan Bozic¹, Velimir Meded^{1,3}, and Wolfgang Wenzel³

¹ Steinbuch Centre for Computing,
Karlsruhe Institute of Technology, Hermann-von-Helmholtz-Platz 1,
76344 Eggenstein-Leopoldshafen, Germany

² Center for Functional Nanostructures,
Karlsruhe Institute of Technology, Wolfgang-Gaede-Str. 1a,
76131 Karlsruhe, Germany

³ Institute of Nanotechnology,
Karlsruhe Institute of Technology, Hermann-von-Helmholtz-Platz 1,
76344 Eggenstein-Leopoldshafen, Germany

E-mail: {ivan.kondov,robert.maul,stefan.bozic,velimir.meded,wolfgang.wenzel}@kit.edu

Employing the UNICORE grid middleware we pursued the concept of service-oriented architecture in order to adequately address challenges in multiscale materials modelling, in particular reusable application interfaces and automated workflows, robust infrastructure for data exchange between workflow components, and software licensing authorization. GridBeans were used to create application interfaces with a graphical user interface for use in the UNICORE rich client and to include them into different workflows. In this contribution we will discuss the functionalities of the developed tools at the example of multiscale modelling of an organic light emitting diode (OLED). The employed model requires treatment on different size scales using different program codes — for quantum mechanics, molecular mechanics, kinetic Monte Carlo (coarse-grained method) and for finite element analysis (continuous method).

1 Introduction

Computational materials science is essential for development of products with novel properties, such as catalysts, semiconductors, polymers and alloys. Next generation materials, especially nano-structured materials, exhibit pronounced complexity and multiscale behaviour. Physical models and simulation protocols employ many different well established methods available in different codes to treat the studied systems on specific time or length scales. However, the lack of integration of these individual codes, the increasing complexity of models and the high demand for distributed high performance computing (HPC) resources reduces the industrial usability of the methods.

In computational materials research the expert knowledge to perform complex multi-step simulations using multiple program codes founded on different physics models can be hardly acquired by one single scientist or by a group of scientists. Multiple scales of modelling imply typically several steps in the simulation. On the other side, there are established expert research groups and program codes focusing on one specific part of the simulation. For example, there are groups that have implemented and routinely use in their research the density functional theory (DFT) to describe the material quantum mechanically on the atomic scale. Integrating this knowledge into a productive e-infrastructure

will make possible such simulation steps, and even multiple steps combined in workflows, to be performed routinely by non-expert users. The infrastructure has to provide reusable and easily adaptable interfaces to the individual program codes, a workflow service for execution of multiple-step simulations, means for data exchange between codes, solutions for licensing issues, security and reliability, capacity and capability, and distributed resources. All these user requirements pose a great challenge for both code developers and providers of e-infrastructures. Optimally these functionalities should be implemented as service oriented architectures (SOA) and provided as web services in order to allow quick and cost-efficient extension due to loose component coupling and existing SOA standards. In addition the SOA approach will allow reliable operation and sustainable business models.

Efforts towards integration of multiscale models are currently done, e.g., by the MAPPER project (www.mapper-project.eu) in which an infrastructure comprising high- and low-level services for programming and execution is developed for a broad range of application areas — fusion, hydrology, physiology, nano-material science and computational biology.

The project MMM@HPC¹ brings together scientists from industry and academia into a unified community which is able to use the e-infrastructure for modern real-life applications in the specific field of materials science. In order to implement the concept in project MMM@HPC outlined in this paper, we have teamed up partners from three different areas of expertise — HPC resource providers, program code developers and users including industrial partners.

In the following section we will outline our approach based on the UNICORE middleware stack. In Section 3 will provide a proof of principle at the example of an organic light emitting diode (OLED) simulation and in Section 4 we will make a summary.

2 Approach Based on UNICORE

The platform of our choice is the UNICORE middleware² that has been broadly and productively deployed in different grid infrastructures such as D-Grid, DEISA and NGI-DE. Moreover, UNICORE 6 is part of the middleware stack deployed in PRACE and thus it provides access to leading supercomputing facilities in Europe. Thus, this middleware enables the access to resources for both capability and capacity computing that can be effectively combined in multiscale models.

Our development is located within the client layer of UNICORE, which runs on top of the service layer and the target systems. The Eclipse-based UNICORE rich client (URC) is used to interface the service layer of UNICORE. The URC has functionality for authentication, monitoring, service and resources discovery (grid browser), and for data, job and workflow management.

2.1 GridBeans

GridBeans is a plug-in based technology especially designed to perform abstraction from the underlying complex and constantly changing grid interfaces. A GridBean³ is a component embeddable in the URC via the GridBean API. The GridBean provides on one hand a graphical user interface to the user with application relevant tools and on the other hand

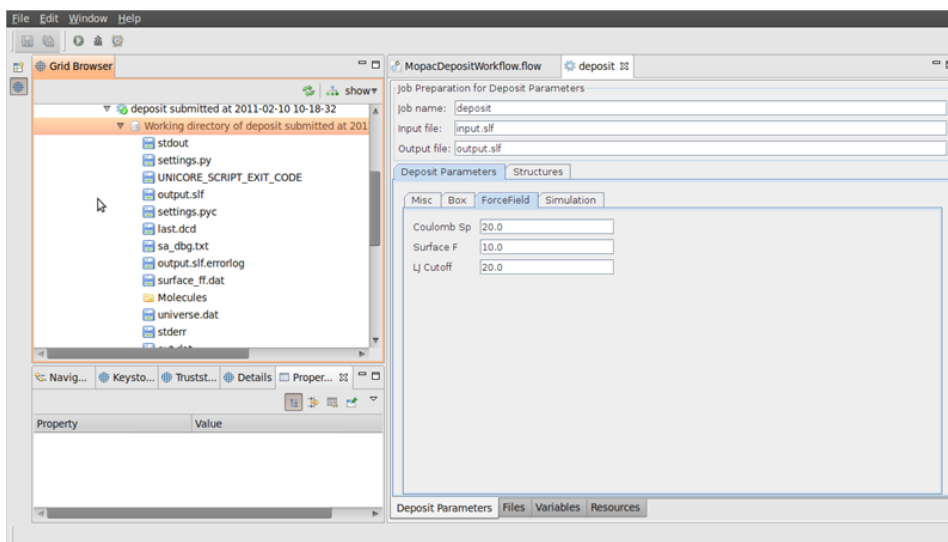


Figure 1. A snapshot of the DEPOSIT GridBean showing it the context of a workflow after a step with the MOPAC GridBean.

access to grid middleware services. The GridBean technology thus provides uniquely efficient (low effort – low cost) means to integrate scientific codes and build a higher level research infrastructure with new qualities (more than the sum of the individual components). In addition, UNICORE provides a GridBean deployment service from which a GridBean can be discovered and downloaded by the user with the URC. As software packages the GridBeans are distributed independently from scientific codes to circumvent potential license issues.

A variety of GridBeans have been made available by projects Chemomomentum, OpenMolGRID and UNICORE Life: Amber, MOPAC, Gaussian, NAMD⁴, GAMESS, Clustal and Blast. In our example in Section 3 we have reused the MOPAC GridBean made available by OpenMolGRID project. In addition, further GridBeans have been developed, e.g. the DEPOSIT GridBean for generation of morphology on the atomic scale while PairFinder and TOFET are under development. In Figure 1 we present an illustration of the DEPOSIT GridBean. On the right panel of the URC the relevant parameters for the simulation, such as the structure of deposited molecules, box geometry, as well as optimisation and force-field parameters, can be specified using the graphical user interface. The generated job can be submitted and monitored until it is finished and then the simulation results are collected via the same user interface (see the Grid Browser on the left panel). In addition, the morphology created and optimised by DEPOSIT can be inspected using the built-in Jmol viewer.

2.2 Workflows

A formal description of the simulation protocols is done via generic workflows⁵. A workflow can be reused for construction of new workflows containing similar simulation steps.

All necessary workflow components, such as workflow editor, workflow monitor / tracer, and workflow engine, have been implemented as services and became part of the UNICORE distribution (with version 6). These extensions of the UNICORE system for workflow support enable the design of various application scenarios for use in dedicated collaborative environments. Both the operativeness of the services and the workflow editor have been demonstrated on examples of application scenarios from the field of computational life science. A workflow can be constructed in the URC and submitted to the UNICORE workflow engine which in turn executes the workflow on resources⁶.

The usage of GridBeans as steps in workflows has interesting implications. First, different GridBeans performing the same task for a certain step are interchangeable. For example, a DFT calculation with a given density functional, basis set and molecular geometry definition can be performed using different codes like Gaussian or TURBOMOLE. Depending on the available license the user can select the corresponding GridBean and will have the same user interface for the step. On the other hand, a specific GridBean is reusable in different workflows, where the adjacent steps (i.e. other GridBeans) may vary. In addition, using GridBeans in the workflows allows implementation of data exchange between adjacent steps, e.g. to specify which output files have to be processed and made available as input for the next step.

2.3 Dataflow Management

In the field of materials Modelling a variety of data formats are used and virtually every individual code has its own input and output formats. No field-wide *de facto* standard exists. Thus, the interoperability and general applicability of the data interfaces of the individual GridBeans needs to be improved. Moreover, due to increasing complexity of data exchanged there is increasing need of generally applicable and abstract dataflow engine to manage the graph of data movements, such like the workflow engine to manage the graph of tasks.

To this end, we currently employ the Chemical Markup Language (CML) standard as data exchange format for any atomistic data definitions, such as molecular and crystal structures and their properties. However, not all data that the individual simulation codes exchange in the course of the simulation can be captured by the CML standard. This is why we extended the interfaces by using the OpenMolGRID^{7,8} framework. This model provides client and server components for UNICORE and is extensible for any further formats via provision of adaptors for the formats to be supported. Currently the OpenMolGRID framework supports different applications and formats. However, the concept is still format-centric and does not provide an API to an abstracted data model without prior definition of the format used in the storage backend.

In computational materials research data complexity is a more important issue for dataflow management systems than the data throughput. Examples for dataflow systems are the Kepler workflow system with the actor model⁹ and the proprietary, however SOA-based system Pipeline Pilot of Accelrys¹⁰ which has been recently used to develop workflows for applications in materials science¹¹ as well as in life sciences for virtual screening¹².

Recently, we started to evaluate the UML based model MEMOPS used in the computational NMR community to exchange data between experiments and structure Modelling/analysis applications¹³. In this model the data structures of the different codes are

described using UML representation. From this data structure definition, APIs callable from each individual code are automatically generated. These APIs, available for different programming languages, such as C++, Java and Python, are abstracted from the data format used by the MEMOPS backend (a file or a database). Very similar concept has been recently adopted in the Universal Access Layer (UAL) used in the fusion plasma physics community together with the Kepler workflow system¹⁴.

2.4 License Management

Many of the codes used in the computational materials science community are provided under non-free, proprietary, licenses. This causes considerable difficulties with the decentralised deployment of the simulation code over computing grids, i.e. across multiple administrative domains. Therefore, a complex authorization matrix (basically mapping permissions of users/groups for different software) in the virtual Organization has to be considered and integrated into the provided services. To this end we are now working on a solution basing on the Virtual Organization Membership Service (VOMS), UNICORE Virtual Organisations Systems (UVOS) and Security Assertion Markup Language (SAML)¹⁵ closely together with the UNICORE developers.

3 Simulation of Charge Transport in Alq3 thin Layers

An OLED, as shown in Figure 2, left picture, consists of multiple, usually amorphous, thin layers deposited on a glass/polymer substrate containing different organic compounds. The interaction on the interface between the layers along with details of charge transport through the layers is very complex warranting for multiscale modelling of the whole OLED device. Different methodologies as steps in one simulation protocol – quantum mechanics (QM), molecular mechanics (MM), coarse-grained (CG) methods and finite element analysis (FEA) on the macroscopic scale (see Figure 2, left picture) are all most necessary. In the following section we will show as a proof of concept the simulation of charge transport in Alq3 thin layers. We note that the goal of the current work is to demonstrate that an already known, however quite complex, simulation can be now carried out automatically using the developed infrastructure.

The selected example is based on the work by Kwiatkowski et al¹⁶ shown in Figure 2, right hand side. Experiments have shown that in amorphous Alq3 holes are trapped more often (about 100 times more) than electrons. This was very well captured in the model applied by Kwiatkowski et al¹⁶. The model system is a cubic box with dimension 10 nm containing amorphous Tris(8-hydroxyquinoline) aluminium (Alq3). At two opposite walls of the simulation box voltage is applied and, additionally, at one side of the layer electrons are being injected while being captured on the other. The holes move in the opposite direction — they are created at the drain electrode and annihilated at the injection electrode side. One goal of the simulation is to predict the electron and hole currents as function of the applied voltage, and to allow understanding of the efficiency of the overall process. The physical phenomena take place on different time scales: from the electronic excitations processes 10^{-15} s, through the hopping process up to the time for an electron to move across the whole structure between the electrodes (in average 10^{-3} s).

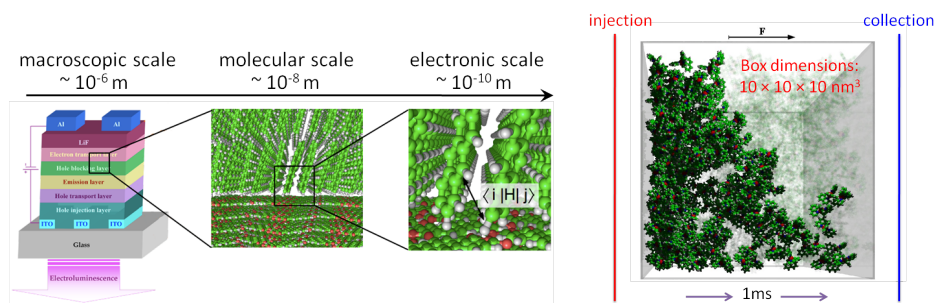


Figure 2. Left hand side: A sketch of a model of an OLED device that requires treatment on different size scales using different code types — QM, MM, KMC and finite element analysis. Right hand side: A simulation box containing amorphous Alq3 as part of the original work by Kwiatkowski et al¹⁶. The highlighted molecules in red have been occupied at least once by a hole (2000 holes are created).

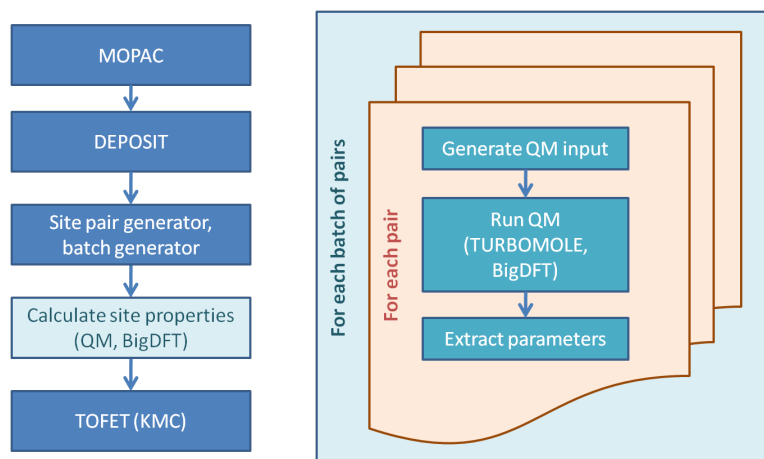


Figure 3. Block diagram of the workflow for the simulation of charge transport in thin layers.

The simulation protocol consists of several steps as summarised in the block diagram in Figure 3. We shall explain the steps in the following.

- **Geometry optimisation of a single Alq3 molecule** We employed MOPAC (Molecular Orbital PACKAGE)¹⁷, which is a semi-empirical quantum chemistry program based on Dewar and Thiel's NDDO (Neglect of Diatomic Differential Overlap) approximation with default PM6 method¹⁸ for this step.
- **MM: Film deposition with generation of a disordered film morphology** To this end we apply relaxation of the molecule positions and orientations using Metropolis & simulated annealing. The disordered phase was simulated by depositing molecules individually using an adapted basis hopping method on top of a surface modeled as an impenetrable plane. The code applied for this step is DEPOSIT developed at KIT¹⁹.

- **Determination of site pairs** According to whether the distance between the molecular centers of mass falls within a certain distance (in our case 13 Å) pairs of neighboring Alq3 molecules are created. The electron hopping partners are defined in this manner, partners that are later used to model electron hopping phenomena. Because there are thousands of molecules in the simulation box, the model can set up millions of individual pairs. Thus the list of pairs is indexed.
- **Pairwise QM calculations** The created pairs are used to create QM jobs that are then bundled in batches and each batch is sent by the Task-GridBean (shown as a bright blue rectangle in Figure 3) to a computing resource over the grid. At this step the HOMO-1, HOMO, LUMO, LUMO+1 energies, the transfer integrals (electronic couplings) and the reorganization energies for each site pair are calculated. Once running on a computing resource each job processes only the QM calculations of its own batch, i.e. iterating over the own list of pairs. Thus, the simultaneous submission of a large number of jobs is avoided and by adjusting the ratio between number of batches and QM calculations per batch the necessary scaling is achieved. At the end of each pairwise QM calculation the effective centers of the sites in the pair and the charge hopping rates using extended Marcus rate formula are calculated. The results from all pairwise QM calculations from all batches are concatenated asynchronously.
- **Coarse-grained model** This step has the goal to compute the charge (electron-hole) mobilities. Following the work by Kwiatkowski et al¹⁶ we use time-of-flight kinetic Monte Carlo (KMC) as implemented in the code TOFET²⁰. Basically, we simulate the mobility measurement experiment without making any approximations regarding the film morphology. The charges (fewer than five at the time) are generated at the bottom 5% and collected at the top 5% of the structure. For this step the TOFET GridBean has been developed.
- **FEA: Calculate current density (not shown)** This step is a significant extension compared to previous work and is currently in development. For this purpose we consider the finite element code Elmer²¹ where the mobilities computed within the previous step enter the electrostatics model based on finite elements. This model can simulate the spacial profile of the currents in the material and the average current density.

In Figure 4 we give a snapshot of the workflow developed within the URC using the machinery described above. In the middle of the URC window there is a container under the tab “Applications” with all additionally developed GridBeans. On the right of the URC window the graphical representation of the OLED workflow is shown. The GUI of the DEPOSIT GridBean is zoomed out on the right hand side of Figure 4. The generated morphology can be inspected visually using the program Jmol which is embedded in the GridBeans (window not shown). Note that there are multiple codes that implement same methodology and not all codes available are considered in this example. Our code selection criteria are maturity, accessibility, high parallel performance, and available expertise by partners.

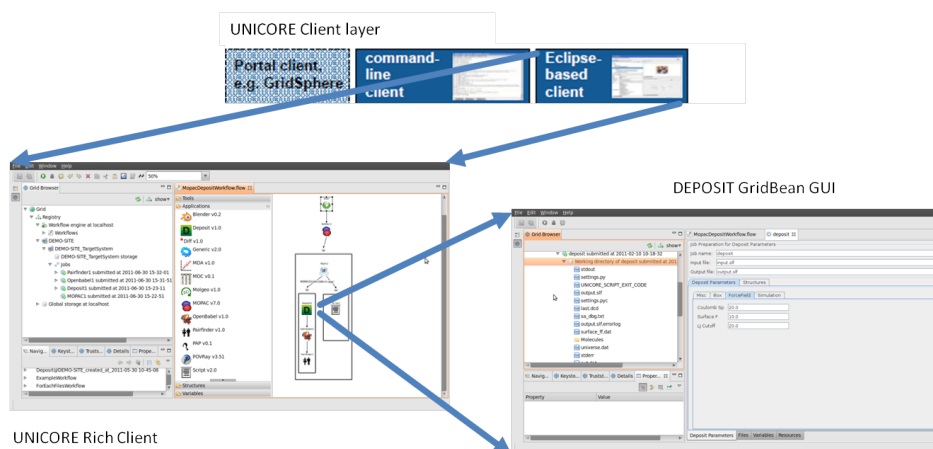


Figure 4. Snapshot of the graphical user interface of the implemented simulation tools.

4 Conclusion

We have employed UNICORE middleware to develop an integrated higher-level service as solution for the modern computational materials research. In particular, we considered physics models which treat the material on multiple size and time scales employing different simulation codes in a multi-step workflow. In order to integrate the simulation codes for the required steps into the workflow special application interfaces, GridBeans, have been used. Some GridBeans have been adopted from other projects and others, e.g. DEPOSIT, PairFinder and TOFET have been additionally developed. To ensure data format interoperability between different steps we have used the OpenMolGRID stack. In future work alternative and more flexible dataflow models will be considered, particularly such ones abstracting from intermediate formatting of the input/output data. Such models and tools are currently in evaluation. Furthermore, a concept for managing user licenses for simulation codes deployed on the grid is currently developed.

We demonstrated the developed functionality by carrying out a completely automated simulation of charge transport in Alq3 thin layers. Such simulation can be then performed routinely in productive environments expediting the development process when varying the chemical composition and morphology of the device.

Acknowledgments

This work has been funded by the 7th Framework Programme of the European Commission within the Research Infrastructures with grant agreement number RI-261594, project MMM@HPC.

References

1. Multiscale Material Modelling on High Performance Computer Architectures (MMM@HPC), www.multiscale-modelling.eu.
2. A. Streit, P. Bala, A. Beck-Ratzka, K. Benedyczak, S. Bergmann, R. Brey, J. Daivandy, B. Demuth, A. Eifer, A. Giesler, B. Hagemeier, S. Holl, V. Huber, N. Lamla, D. Mallmann, A. Memon, M. Memon, M. Rambadt, M. Riedel, M. Romberg, B. Schuller, T. Schlauch, A. Schreiber, T. Soddemann, and W. Ziegler, *UNICORE 6 — Recent and Future Advancements*, *Annals of Telecommunications*, **65**, 757–762, 2010.
3. R. Ratering, A. Lukichev, M. Riedel, D. Mallmann, A. Vanni, C. Cacciari, S. Lanzarini, K. Benedyczak, M. Borcz, R. Kluszczynski, P. Bala, and G. Ohme, *GridBeans: Supporting e-Science and Grid Applications*, in: *Second IEEE International Conference on e-Science and Grid Computing, 2006 (e-Science '06)*, pp. 45–52, 2006.
4. R. Kluszczynski and P. Bala, “Supporting NAMD Application on the Grid Using GPE”, in: *Parallel Processing and Applied Mathematics*, R. Wyrzykowski, J. Dongarra, K. Karczewski, and J. Wasniewski, (Eds.), vol. 4967 of *Lecture Notes in Computer Science*, pp. 762–769. Springer, 2008.
5. J. Yu and R. Buyya, *A Taxonomy of Workflow Management Systems for Grid Computing*, *J. Grid Comp.*, **3**, 171–200, 2006.
6. B. Demuth, B. Schuller, S. Holl, J. Daivandy, A. Giesler, V. Huber, and S. Sild, *The UNICORE Rich Client: Facilitating the Automated Execution of Scientific Workflows*, in: *2010 IEEE Sixth International Conference on e-Science (e-Science)*, pp. 238 – 245, 2010.
7. S. Sild, U. Maran, M. Romberg, B. Schuller, and E. Benfenati, “OpenMolGRID: Using Automated Workflows in GRID Computing Environment”, in: *Advances in Grid Computing - EGC 2005*, Peter Sloot, Alfons Hoekstra, Thierry Priol, Alexander Reinefeld, and Marian Bubak, (Eds.), vol. 3470 of *Lecture Notes in Computer Science*, pp. 464–473. Springer, 2005.
8. S. Sild, U. Maran, A. Lomaka, and M. Karelson, *Open Computing Grid for Molecular Science and Engineering*, *J. Chem. Inf. Modeling*, **46**, 953–959, 2006.
9. B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao, *Scientific workflow management and the Kepler system*, *Concurrency and Computation Practice and Experience*, **18**, 1039–1065, 2006.
10. Pipeline Pilot, Accelrys, www.accelrys.com.
11. Xiaoyu Yang, Richard P. Bruin, and Martin T. Dove, *Developing an End-to-End Scientific Workflow: A Case Study Using a Comprehensive Workflow Platform in e-Science*, *Computing in Science and Engineering*, **12**, 52–61, 2010.
12. J. M. Garcia H. Perez-Sanchez, I. Kondov, K. Klenin, and W. Wenzel, *A Pipeline Pilot based SOAP implementation of FlexScreen for High-Throughput Virtual Screening*, in: *Proceedings of 3rd International Workshop on Science Gateways for Life Sciences (IWSG-Life 2011)*, 2011.
13. R. H. Fogh, W. Boucher, J. M. C. Ionides, W. F. Vranken, T. J. Stevens, and E. D. Laue, *MEMOPS: Data modelling and automatic code generation*, *J. Integr. Bioinform.*, **7**, 123–145, 2010.
14. G. Manduchi, F. Iannone, F. Imbeaux, G. Huysmans, J.B. Lister, B. Guillerminet,

- P. Strand, L.-G. Eriksson, and M. Romanelli, *A universal access layer for the Integrated Tokamak Modelling Task Force*, Fusion Engineering and Design, **83**, 462 – 466, 2008.
15. K. Benedyczak, M. Lewandowski, A. Nowinski, and P. Bala, “Unicore virtual organizations system”, in: Parallel Processing and Applied Mathematics, Roman Wyrzykowski, Jack Dongarra, Konrad Karczewski, and Jerzy Wasniewski, (Eds.), vol. 6068 of *Lecture Notes in Computer Science*, pp. 155–164. Springer, 2010.
 16. J. J. Kwiatkowski, J. Nelson, H. Li, J. L. Bredas, W. Wenzel, and C. Lennartz, *Simulating charge transport in tris(8-hydroxyquinoline) aluminium (Alq3)*, Phys. Chem. Chem. Phys., **10**, 1852–1858, 2008.
 17. J. J. P. Stewart, MOPAC2009, 2008, Stewart Computational Chemistry, Colorado Springs, CO, USA.
 18. J. J. P. Stewart, *Optimization of parameters for semiempirical methods V: Modification of NDDO approximations and application to 70 elements*, J. Molec. Modeling, **13**, 1173–1213, 2007.
 19. A. Verma, A. Schug, K. H. Lee, and W. Wenzel, *Basin hopping simulations for all-atom protein folding*, J. Chem. Phys., **124**, 044515, 2006.
 20. J. J. Kwiatkowski, J. M. Frost, and J. Nelson, *The Effect of Morphology on Electron Field-Effect Mobility in Disordered C60 Thin Films*, Nano Lett., **9**, 1085–1090, 2009.
 21. Elmer finite element software homepage, <http://www.csc.fi/elmer>.

Serpens Suite for Kepler Workflow Orchestration System

Marcin Płóciennik¹, Michał Owskiak¹, Tomasz Żok¹, Antonio Gómez-Iglesias², and Francisco Castejón²

¹ Poznan Supercomputing and Networking Center,
E-mail: {marcinp,michalo,tzok}@man.poznan.pl

² Centro de Investigaciones Energéticas, Medioambientales y Tecnológicas,
E-mail: {antonio.gomez,francisco.castejon}@ciemat.es

This paper presents the overview of the Serpens suite for the Kepler workflow orchestration system. The Serpens suite provides the Kepler extension for enabling distributed Grid execution. Its modules leverage and integrate various, already existing, middleware components and enable work with middleware stacks like UNICORE and gLite. This research has been driven by Nuclear Fusion applications' requirements. The driving idea was to enhance the modelling capabilities for ITER sized plasma research by providing access to grid and HPC resources. The solution described is very general, this way it could be integrated into the official Kepler release and can be used in non-plasma based use cases as well. We highlight features and important concepts through example workflows and results of their execution.

1 Introduction

Scientific communities have nowadays the possibility of accessing a large variety of computing resources. Advanced sciences like Nuclear Fusion that strongly depends on simulation and modelling, are nowadays benefitting from the rapid advancement of computing hardware and increased accessibility to computational tools and infrastructures. As computational capabilities are getting increasingly more powerful, efficient use of the available infrastructure is becoming more and more complex from users perspective. Clearly, scientists prefer to focus on their research and not on issues related to platforms and tools required to perform it. As a result, there are projects focused on developing or deploying scientific workflow management systems that enable the optimisation of the process of efficient usage and access to resources. The goal of the scientific workflows is to automate repetitive tasks. They can greatly support researchers in activities such as data access, analysis, job submissions and visualisation.

This paper presents the concept of the Serpens suite for the Kepler workflow orchestration system¹. Using Kepler's graphical interface and components, scientists can create executable scientific workflows, which are flexible tools for accessing scientific data (streaming sensor data, medical and satellite images, simulation output, observational data, etc.) and executing complex analyses on them. Kepler is shipped with a searchable library containing many ready to use processing components ('actors'). These can be easily customised, connected, and then run (as a workflow) from a desktop-like environment. Actors can perform calculations, automatic data management, or execution of external applications. Kepler workflows can be decomposed into smaller parts that allow complex tasks to be divided into simpler ones. This feature provides workflow designers with the ability to build re-usable, modular sub-workflows.

The Serpens suite, presented in this paper, provides a set of Kepler actors and workflows enabling distributed Grid execution. Its modules leverage and integrate various, al-

ready existing, middleware components and make it possible to work with UNICORE and gLite² directly from Kepler. There are two modules related to UNICORE: Serpens UNICORE module and VineToolkit module (The Vine Toolkit is an adaptor to various middleware stacks. It provides access to target infrastructure via an abstract, Java based API. Among others it abstracts from the UNICORE access layer). All these tools and their inter-dependencies are detailed later in this article.

This research has been started as a part of the EU funded project EUFORIA (EU Fusion for ITER Applications). EUFORIA was enhancing the modelling capabilities for ITER sized plasmas by providing the access grid and HPC resources and by adaptation and integration of a set of critical applications for edge and core transport modelling. One of EUFORIA's goals was to promote a novel approach to dynamic coupling and integration of physical codes executed within heterogeneous platforms. Integration was carried out via a workflow engine - Kepler. One of the aims of the project was to extend both, workflow orchestration system and middleware in such a way, that grid and HPC support could be transparently incorporated into existing physical applications.

The article is organised as follows. In Section 2, we analyse the state of the art, while in Section 3 we present the use cases and their requirements. Afterwards, we describe the concepts behind the access layer for computing resources in Section 4; finally, in Section 5 we evaluate the results and software developed. As a conclusion, we present future directions of work.

2 Related Work

Since even only mentioning all possible workflow systems would exceed the limit of this publication we rather focus on several examples of workflow frameworks that target at supporting distributed infrastructures like Grids. We refer to cases in which workflows may require the scheduling of high-performance computing (HPC) resources such as local clusters or remote (Grid, HPC or cloud computing) resources. There are several systems that provide a convenient way of building and executing these workflows. We would like to mention the largest systems like Pegasus³, Taverna⁴, Triana⁵, P-GRADE⁶, but also more specific solutions like GridAnt, UNICORE Workflow System⁷.

Pegasus³ is a framework for mapping scientific workflows onto distributed resources including Grid and Cloud-based systems. Pegasus has been used in a number of scientific domains including astronomy, bioinformatics, earthquake science, gravitational wave physics, ocean science, limnology, and others. It has been used to run workflows ranging from just a few computational tasks up to 1 million. Pegasus is focusing on middlewares like Condor, Globus, or Amazon EC2. **Taverna**⁴ is an open source and domain-independent Workflow Management System – a suite of tools used to design and execute scientific workflows and aid in silicon experimentation. Taverna can access a large number of services in the fields of bioinformatics, astronomy, chemoinformatics, health informatics and others. Taverna can invoke generic WSDL-style Web services (*WSDL*: Web Service Definition Language). The example supported grid middleware is KnowARC, GridSAM and gLite². The **P-GRADE**⁶ Grid Portal is a web based, service rich environment for the development, execution, and monitoring of workflows and workflow based parameter studies on various grid platforms. Workflows and workflow based parameter studies defined in the P-GRADE Portal are portable between grid platforms. It is focused on interoperability

with Globus Toolkit 2, Globus Toolkit 4, LCG and gLite grid middlewares. **Triana**⁵ is a workflow-based graphical problem solving environment independent of any particular problem domain. Triana workflows have incorporated a range of distributed components such as Grid jobs or Web services. **GridAnt** is an extension of the Apache Ant build tool residing in the Globus COG kit. GridAnt allows for the construction of client side workflow for Globus Toolkit applications. It allows for the specification of precondition and parallel tasks in much the same way as the Ant build tool. **UNICORE Workflows**⁷ are one of the most important features of UNICORE middleware. The UNICORE Rich Client offers the users to design workflows in a graphical way. The UNICORE Workflow Engine translates a workflow description into a set of single jobs which are run in the specific order.

There is no clear general criteria to say that one of the systems is better than others, since the frameworks are usually very complex. They usually target different aspects and are built starting from specific applications requirements. For the fusion community a detailed evaluation of existing technologies resulted in choosing Kepler as the main platform, since Kepler as data-flow driven workflow system with its directors concepts was the most convenient solution for managing and integrating fusion codes and ontologies.

3 Use Cases Description

The solution presented in this paper was dictated by the requirements of the nuclear fusion applications (EUFORIA, EGI-Inspire). One of the key factors was to provide applications with universal access to both HPC and Grid resources. Another requirement was to provide the user with the ability of submitting multiple tasks within a workflow based environment – to provide a way to submit multiple grid/HPC jobs. More advanced use cases assumed usage of different resources within one workflow at the same time. Scientists wanted to build workflows with different computational requirements. They needed to mix sequential codes with parallel ones. In this section we present several use cases and their specific requirements. The presented applications have been implemented using the Serpens suite. Details regarding each particular use-case are presented in Section 5.

3.1 VMEC-DKES

VMEC⁸ (Variational Moment Equilibrium Code) is a three-dimensional MHD equilibrium solver based on nested magnetic surfaces. It is used to calculate the MHD equilibrium of a given configuration.

The configuration space is defined by the control variables, which are the Fourier harmonics of the cylindrical coordinates R and Z describing the shape of the outermost magnetic flux surface. To reduce the twists and bends of the coils eventually required to produce the optimal surface, the local flux surface curvature is bounded. The alignment of the B_{min} , B_{max} , and trapped J contours with ψ is performed at three or more radial flux surfaces and, for J , at four values of λ .

VMEC has become the *de facto* standard code for calculating 3-D equilibrium. It is used in nearly all of the laboratories that house stellarator devices.

DKES (Drift Kinetic Equation Solver) calculates the neoclassical diffusion coefficient as a function of collisionality. DKES uses as input an equilibrium previously calculated by

VMEC, and also another input file with specific information not needed by VMEC. Thus, the coupling of both applications is easy to implement. It is also a *de facto* standard code in the stellarator community.

Both applications are serial and have a large number of configuration parameters. Several parameter scans can be performed using just VMEC, which will generate several output files containing MHD equilibria. Each of these outputs can be the input for DKES. Implementing another parameter scan using the DKES input (not the equilibrium), we can run a large number of different tests using the same equilibrium file. Thus, this use case implies the execution of a parameter scan in the grid. Each execution, will then require another parameter scan.

3.2 VMEC-Mercier-COBRA-Visualisation

This use case is focused on the characterisation of the transport of particles in confined plasmas. As previously mentioned, VMEC calculates the MHD equilibrium of a given configuration. Once we have achieved this equilibrium, we can proceed with further characteristics of plasmas.

The so called Mercier⁹ modes are local instabilities characterised by being radially localised but toroidally and poloidally extended, i.e., by $k_{\perp} \gg k_{\parallel}$.

The Mercier stability criterion is presently implemented in the VMEC code (refer to the following section). The last versions of VMEC code estimates the Mercier stability criterion for the effective radius 0 to 1 (the radius of the plasma is reduced to that format). Since the area of interest is in the range $[0.8 - 1]$, only those values are taken into account. Stable Mercier configurations are those satisfying the criterion in that range.

Besides Mercier stability criterion, the Ballooning mode stability is also imposed. Ballooning modes are characterised by being radially, toroidally, and poloidally localised, thus $k_{\perp} \sim k_{\parallel}$. Ballooning modes appear in zones of unfavourable curvature. The COBRA (Code for Ballooning Rapid Analysis) code¹⁰ performs this stability analysis.

These are serial applications which are executed in the grid. On the other hand, the generation of the 3D visualisation file of the results obtained with those applications is executed on HPC resources. This is a shared-memory parallel application (using OpenMP) which uses the output of the previous applications generates a Silo¹¹ visualisation file which can be displayed later using VisIt visualisation tool¹². Thus, the output of VMEC and COBRA is sent from the grid resources to HPC resources, where it will be used by the parallel application to create this Silo file which can be later used in any machine running VisIt.

Figure 1 shows the output of the visualisation application as displayed with VisIt. This visualisation plays an important role in the understanding of the confinement capabilities of the fusion reactor.

A single execution of this use case consists of the evaluation of a configuration of a given fusion device. However, this configuration is characterised by a large number of configuration parameters. Different values for these parameters will provide different configurations which may present severe differences among each other. Thus, a parameter scan is performed by changing the values of several of these parameters.

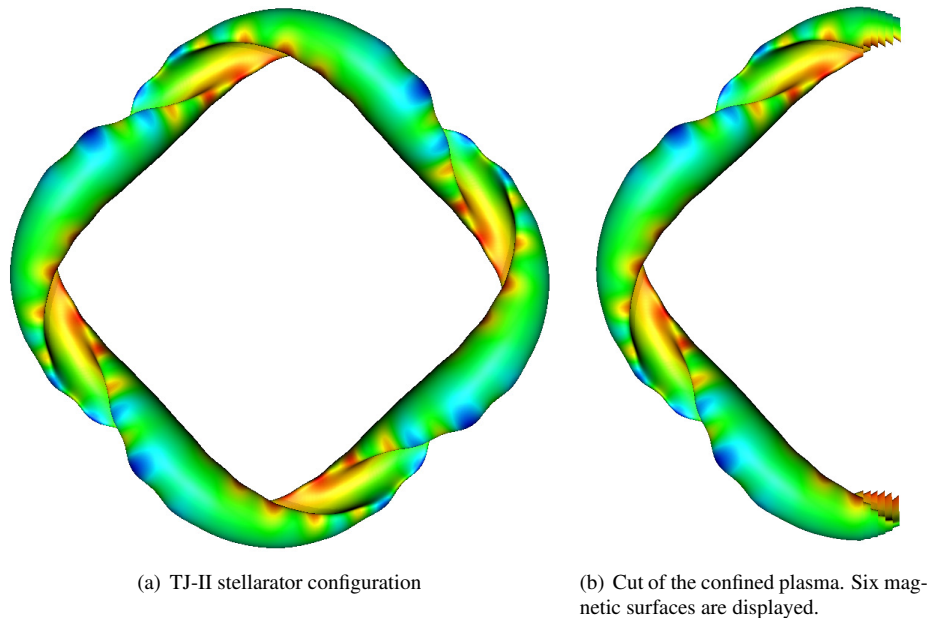


Figure 1. Top view of the plasma confined in a fusion device.

4 Architecture Details

Kepler is a scientific workflow system built upon the open source Ptolemy II framework¹³. The project was initiated in 2003 as a collaboration of UC Davis, UC Santa Barbara and UC San Diego. Over the years many contributors from various scientific and engineering areas have supported Kepler both in development time and expertise. The team, behind development of Kepler's core (they also lead the whole development and integration process) is funded on NSF basis. The development decisions took into account the vision of a generic and uniform architecture as well as domain-specific scientific extensions.

4.1 Workflow Directors

Each Kepler workflow requires a director. It is a special object which orchestrates and schedules the execution of workflow elements. There are several predefined directors, each suitable for a different purpose. The Synchronous Dataflow (SDF) is a simple yet very efficient director. It achieves good performance by prescheduling workflow execution. It requires that the order of execution is inferable from the workflow structure itself. This in practice excludes conditional branching and looping. For these purposes, the Dynamic Dataflow (DDF) director can be used. It is still a single-threaded execution model, however the schedule is updated in each iteration. This allows conditional branches to be executed depending on the boolean value of the condition clause. Yet another type of director is even better suited for parallel execution - we mention here Process Networks (PN). In this model, each workflow element is run in a separate thread waiting for its input

ports to be filled with data. This behaviour is very useful with workflow models having either concurrent or parallel execution scheme. There are numerous directors in Kepler, not discussed in this paper, better suited for particular use cases. They are described in details within Kepler's user manual¹³.

4.2 Workflow Actors

The aforementioned workflow element, actor, is orchestrated by the director. It means that an actor can not perform its execution unless it is told to do so. There are two types of actors in Kepler, the basic and composite ones. A basic actor is an object characterised by its set of input and output ports, internal parameters and some action incorporated within it. This is the main computation element of the workflow. In an actor-oriented design, one analyses how data flow is organised. Which data go into the actor, how they are processed and which data are produced as a result. Some actors are just responsible for producing data, others are simple data consumers, but in most cases, actors perform both actions at the same time. Director, actors and relations between actors constitute the workflow. The second type of actor, the composite one, is in fact a workflow that is embedded into another workflow. This way, the complexity of a workflow can be hidden by abstracting commonly used parts. These parts are expressed as composite actor. This approach promotes modularity and reuse of already existing workflows.

4.3 Suites and Modules

In version 2 of Kepler, a new release scheme was introduced. Functionality is divided between modules which are gathered in suites. A separate module consists of source codes for actors it publishes, external libraries, licensing information and other Kepler-related metadata. A suite is an ordered list of modules. User may decide to install a single module or a full suite which will ensure that all intermodular dependencies are met. Each module is versioned and resources created in Kepler are saved with these version numbers in metadata. This prevents users from using workflows or composite actors in a non-compatible Kepler installation.

4.3.1 Serpens Suite

The Serpens suite is an extension to Kepler that employs the mentioned approach. It provides standard activities related to remote job management: submission, monitoring, data management, and so on as a set of actors, both basic and composite. By combining a meaningful sequence of such actors, client applications can create a complete workflow in a very flexible and convenient manner. Kepler is responsible for user interaction and managing multisite, distributed, and heterogeneous workflows. The heterogeneity here, implies that a single workflow, spanning multi-step activities, contains Grid/HTC and HPC application executions.

4.3.2 gLite Module

The first module contains actors and workflows to work with grid environments using gLite middleware. It allows creating VOMS proxies, transferring data using the Globus GridFTP

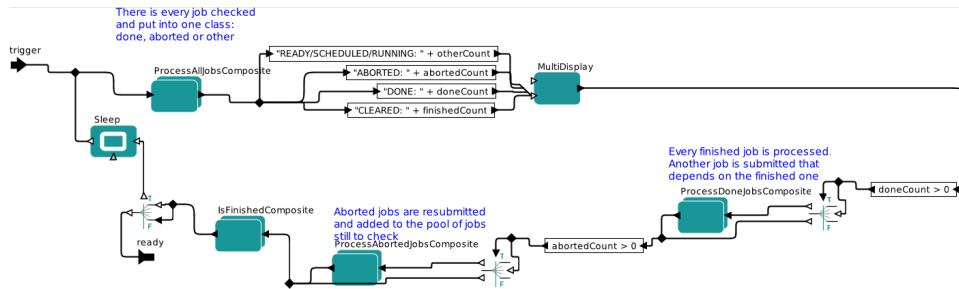


Figure 2. A main loop of gLite template workflow. It manages up to thousands of grid jobs making sure to resubmit them upon failure and to safely download output files. It is a part of the gLite module within the Serpens suite.

protocol, submitting, and managing jobs. The accompanying template workflow is a full use case implementation ready to be modified according to specific needs of the grid application. A main loop of the workflow which manages multiple grid jobs is presented as Figure 2.

The access to gLite resources is done using a dedicated Roaming Access Server (RAS). It is a layer between Kepler actors and the grid infrastructure. It is designed to hide the complexity of configuration, grid security and operations execution by publishing a set of well-defined web services. This makes the creation of basic Kepler actors much easier and manageable, while on the other hand makes the grid administration more centralised and more easily configurable.

Besides gLite operations, the template workflow incorporates also various failure detection and prevention mechanisms. Jobs are monitored by their status or directly by the logging and bookkeeping events. All data transfers are checked, especially if they concern output retrieval. It may happen that despite the job finished successfully, the output files get lost due to various networking, security or general infrastructure issues. These situations are handled by Serpens actors. Besides these dynamic operations, we also incorporate several static information sources to avoid using faulty infrastructure elements in the first place. This means to ignore unresponsive storage, erroneous workload managers or computation elements which often lead to jobs failure. This strategy led to significant success ratio improvement of the jobs we manage via Serpens actors and workflows versus jobs submitted normally. It allows submitting up to thousands of jobs, managing them in spite of infrastructure problems occurring naturally or randomly, and keeping track of the overall progress even between separate Kepler sessions. It comes as a generic single application submitter, although a set of parameters fully describe its functionality and allows tuning it in a lot of ways.

4.3.3 UNICORE Module

The second module is responsible for supporting the UNICORE middleware. Actors contained within are based on a UNICORE Commandline Client (UCC) and mimic behaviour of this application. User can request information from the UNICORE registry, use available storages and sites, submit and manage jobs. This module contains also several exam-

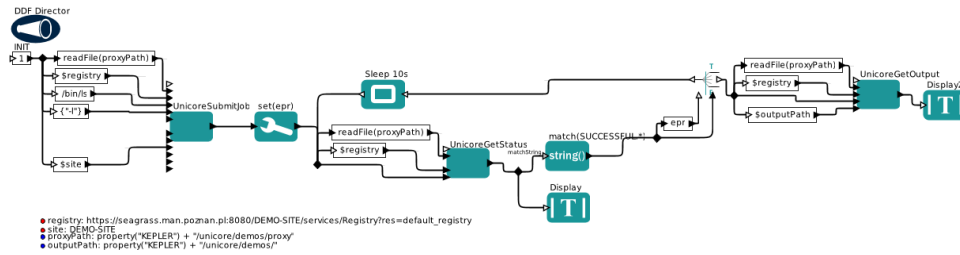


Figure 3. A full use case workflow for UNICORE. It submits a job with input ports, waits until it is successfully finished and retrieves output files. It is a part of the UNICORE module within the Serpens suite.

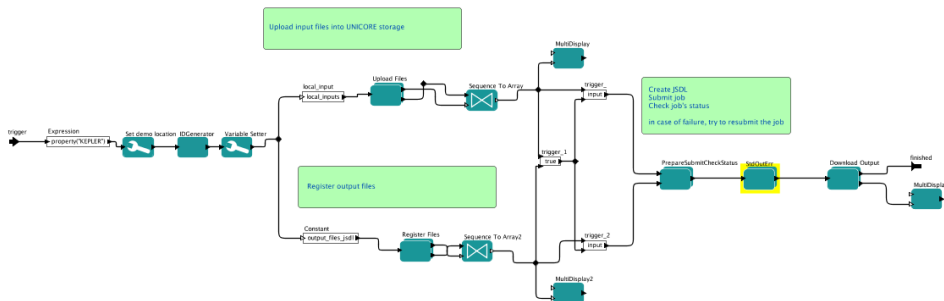


Figure 4. A full use case workflow for UNICORE using Vine Toolkit. It uploads input files and registers outputs. Then a JSDL document is generated, job submitted and managed until it is successfully finished and output files are downloaded. It is a part of Vine Toolkit module within Serpens suite.

ple workflows presenting clearly the usage of each separate actor. There is also a full use case implementation which highlights the way to submit a job with input files, wait until it is successfully finished and retrieve its output. This is presented in Figure 3. Kepler mechanism of composite actors enables to get this ready workflow as a single entity of a larger workflow, which makes the UNICORE module even more usable for a wider context.

4.3.4 Vine Toolkit Module

The last module in the Serpens suite is a set of actors working with Vine Toolkit. It is a software package which unifies access to various grid middleware infrastructures. Using the same set of actions users can work with remote storages of different kinds and remote job management schemes in a simple and uniform way. In particular, among others the UNICORE, gLite or QosCosGrid middleware technologies are supported. The Vine Toolkit module for Kepler contains actors which represent basic actions provided by this technology. There are also example workflows and as with previous modules – a template workflow implementing a full job submission use case. It is presented as Figure 4.

Table 1. Timing of different actions happening in VMEC-DKES use case.

Action name	Time [min]
Input files upload	0.5
Single VMEC run	20
Single DKES run	10
Final outputs download	1

5 Solution Evaluation

A number of applications have been already developed and deployed with usage of the Serpens suite. These applications are mainly coming from the Fusion community as discussed in the section of the use case analysis. Initial set of the workflows has been deployed in the EUFORIA project. As already mentioned, the EUFORIA project enhanced modelling capabilities for ITER through the joint use of HTC and HPC resources together with the fusion modelling community by adaptation, optimisation and integration of a set of critical applications for edge and core transport modelling as well as turbulence simulations. Most of these application codes allow construction of the complex, combined workflows that produce advanced physics results. We would like to describe here the implementation of workflows for the fusion codes like VMEC-DKES and VMEC-Mercier-COBRA which were described in details in the use cases section.

5.1 VMEC-DKES

The VMEC application calculates an MHD equilibrium for a given parameter set. This allows executing a parameter scan and running multiple and parallel instances of VMEC with different initial parameters. For each equilibrium found, a DKES parameter scan can be run. The user prepares a set of differing initial parameter values for DKES and each one of them will be run with each of the calculated equilibria. This double parameter scan can easily yield thousands of separate jobs, thus a grid environment suits this use case requirements the best.

In our evaluation, we have prepared 14 parameter sets for VMEC and 180 for DKES. Together it sums up to 2,520 jobs which is a great test suite for efficiency, performance and failure prevention mechanisms available in the workflow solutions inside Serpens for Kepler. The timings (average time) of particular actions of this use case are shown on Table 1. The jobs were run on different clusters available, however the typical configuration met was like the following:

- Scientific Linux 5.4,
- Intel®Xeon 3.2 GHz, Quad-core,
- 16 GB of physical memory.

The workflow solution was designed to make a full use of grid computations. Despite the time for a single VMEC run being constant, the heterogeneous nature of grid makes it that the real time of job execution (from submission to output retrieval) was differing

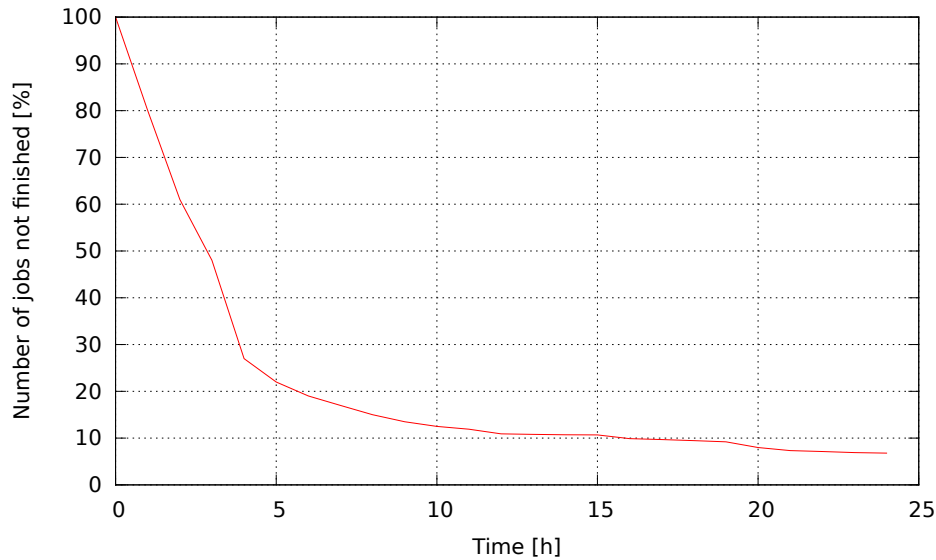


Figure 5. A relation between the number of jobs not finished and time of workflow execution in a heterogeneous grid environment.

relatively much between the 14 jobs submitted in the first phase. Thus, as soon as any VMEC job finished a 180 DKES parameter scan based on its equilibrium was immediately submitted.

The job management and failure prevention mechanism is a complex system of composite Kepler actors for bookkeeping, monitoring and eventually job resubmission. Thus we decided to split this functionality for two applications VMEC and DKES into two related workflows. The first one is responsible for successful VMEC job execution and DKES job submission. The second is loading all the DKES jobs produced and manages them until successful finish. Both workflows run in parallel at the same time without interfering with each others behaviour. This allows managing VMEC and DKES jobs at the same time.

In our evaluation tests we put much effort in checking the overall stability and expected user satisfaction. Submission of such a high number of jobs produces lots of output data. Even with a very stable infrastructure with a resubmission ratio being on an extremely low level, the number of faulty jobs will be positive at least in the long term. Our proposed workflow solution is not only a visual layer for the user to prepare his computational experiments. The fault tolerance and prevention mechanism plays a huge role in the workflow inner processing. Our evaluation tests proved that the whole use case was eventually finished successfully despite many resubmissions, queue stalls or infrastructure problems. By successful we mean that in the end, the real output for each of the 2,520 jobs was retrieved and was stored on user disk in full. From the workflow start to such defined end, it took 59 hours.

We have run all the jobs on fusion community grid which is a typical heterogeneous

environment. Computational and storage nodes have very different configurations (even different CPU architecture), are located all over the world, and have non-consistent network throughput between each other. To make the full test, we did not decide on any particular cluster, but chose to use all the infrastructure elements available which met some minimal requirements for VMEC and DKES applications. During the tests we observed that the relation between the number of jobs not finished and time of workflow execution resembles the power law function as is shown on Figure 5.

5.2 VMEC-Mercier-COBRA-Visualisation

The second use case is an evaluation of a concept of mixed middleware and resources in a single workflow. The VMEC-Mercier-COBRA part was run on the grid, as the parameter scan best suited this type of distributed environment. The visualisation part, as described in the use cases section, incorporates a parallel implementation to which the HPC resources apply better. We have successfully merged this different concepts into a single workflow thanks to general scope of the actors in Serpens suite.

The concept of composite actors allowed us to embed a complete UNICORE template workflow as a single entity inside the gLite template. This way, not only did we run jobs on different types of distributed environments, but we have in fact used totally different middlewares to access them. The workflow itself was a manager of this whole process, keeping all the necessary steps and phases in correct order and in correct state.

The VMEC-Mercier-COBRA as a parameter scan job produced many outputs in different order. Because the workflow managed the use case on different middlewares at once, we designed it to process HTC and HPC jobs separately. This allowed us to avoid undesired situations in which one type of job blocks the other.

The workflow was successfully evaluated not only in the technical terms described above, but especially by fusion physics community. The resulting visualisation shown in the use cases section in Figure 1 was a result of our workflow solution.

6 Future Plans

Serpens has been presented during different hands-on tutorials, demonstrations and lectures, e.g.: EGI_User_Forum, SC2009 and SC2010, as a part of the demo for the results of the EUFORIA-DEISA-EGEE pilot project. Serpens has been committed, recently, as a part of the main Kepler repository (BSD license). It is officially released with the newest Kepler version. This opens new possibilities for reaching new communities and users and working on fulfilling new requirements and use cases. Initial use cases are coming from the Nuclear Fusion, but since the provided solution is general, use cases from other communities are already under development and will be deployed in the future. Also since the middleware is changing rapidly, and new emerging infrastructures are appearing, Serpens is planned to be extended in order to support new scenarios. In particular the support for cloud technologies is foreseen.

7 Summary

In this paper we have presented how workflow based computation can be utilised Grid and HPC related resources. We have shown the state of art in the area of workflow managing

systems by describing some of the existing solutions. Kepler is one of these systems and the Serpens related development is strictly bound to it. After the environment is defined, we have presented real life use cases that show how a workflow based approach can be utilised within field of physics. These use cases present applications that benefit from execution within both Kepler and Grid based environments. Workflow managers allow code owners to abstract from the computation layer, while Grid resources provide them with efficient computational resources. These use cases are the basis to describe which kind of assumptions have been made before actual development started. These use cases led us to formulate the architecture design as described in section 4. After detailing the design of the software we moved to module based organisation of the Serpens project and described how is it related to Kepler and its modules. In essence, Serpens module was used during scientific research and helped to produce real life results. These results are also presented and discussed within this article. Of course, even though Serpens is already incorporated into Kepler's repository there is still open space left for further development. We discussed it as well, suggesting directions of further development.

Acknowledgements

The research leading to these results has received funding from the FP7 (FP7/2007-2013) under grant agreement n°211804 (EUFORIA). Support for user communities using Kepler scenarios is done under EGI_InSPIRE project, under grant agreement n°261323. Part of the service developments are supported in PL-Grid project, co-funded by the European Regional Development Fund as part of the Innovative Economy program.

References

1. I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher and S. Mock. *Kepler: an extensible system for design and execution of scientific workflows*. *16th International Conference on Scientific and Statistical Database Management*, pp. 423–424 (2004).
2. E. Laure, S. M. Fisher, A. Frohner, C. Grandi, P. Kunszt, A. Krenek, O. Mulmo, F. Pacini, F. Prelz, J. White, M. Barroso, P. Buncic, F. Hemmer, A. Di Meglio, A. Edlund *Programming the Grid with gLite Computational Methods in Science and Technology* **12**(1), pp. 33-45 (2006).
3. E. Deelman, G. Singh, Mei-Hui Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G.B. Berriman, J. Good, A. Laity, J.C. Jacob, D.S. Katz *Scientific Programming Journal*, **13**(3), pp. 219-237 (2005).
4. D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M. Pocock, P. Li, and T. Oinn *Taverna: a tool for building and running workflows of services*. *Nucleic Acids Research*, **34**, iss. Web Server issue, pp. 729-732 (2006).
5. I. Taylor, M. Shields, I. Wang, A. Harrison. *The Triana Workflow Environment: Architecture and Applications Workflows for e-Science*, Springer, pp. 320-339 (2007).
6. Z. Farkas, P. Kacsuk *P-GRADE Portal: A generic workflow system to support user communities* *Future Generation Computer Systems* **27**(5), pp. 454-465 (2011).
7. S. Gudenkauf, W. Hasselbring, A. Höing, G. Scherp and O. Kao *Workflow Service Extensions for UNICORE 6 - Utilising a Standard WS-BPEL Engine for Grid Service*

- Orchestration*. Proceedings of 4th UNICORE Summit 2008 in conjunction with EuroPar 2008, Las Palmas de Gran Canaria, Spain, pp. 103-112 (2008).
8. S.P. Hirshman and G.H. Neilson *External inductance of an axisymmetric plasma*. *Physics of Fluids* **29**(3), pp. 790–793 (1986).
 9. C. C. Hegna and N. Nakajima. *On the stability of Mercier and ballooning modes in stellarator configurations*. *Physics of Plasmas*, **5**(1336), pp. 1336–1344 (1998).
 10. R. Sanchez, S. P. Hirshman, J. C. Whitson, and A. S. Ware. *COBRA: an optimized code for fast analysis of ideal ballooning stability of three-dimensional magnetic equilibria*. *Journal of Computational Physics*, **161**(2), pp. 576–588 (2000).
 11. Lawrence Livermore National Laboratory. *Silo website (June 2011)*.
<https://wci.llnl.gov/codes/silo/>.
 12. H. Childs, S. A. amd E. Deines, T. Fogal, C. Garth, J. Meredith, Prabhat, D. Pugmire, O. Rübel, A. Sanderson, G. Weber, and B. Whitlock. *VisIt: a production tool for visualizing and analyzing large data*. U.S. Department of Energy. Office of Science. Scientific Discovery through Advanced Computing (2009).
 13. Edward A. Lee *Disciplined Heterogeneous Modeling* Proceedings of the ACM/IEEE 13th International Conference on Model Driven Engineering, Languages, and Systems, Springer-Verlag, pp. 273-287, (2010).
 14. Bertram Ludäscher and Ilkay Altintas and Shawn Bowers and Julian Cummings and Terence Critchlow and Ewa Deelman and David De Roure and Juliana Freire and Carole Goble and Matthew Jones and Scott Klasky and Timothy McPhillips and Norbert Podhorszki and Claudio Silva and Ian Taylor and Mladen Vouk *Scientific Data Management: Challenges, Technology, and Deployment Computational Science Series, Chapter 13: Scientific Process Automation and Workflow Management*, Chapman and Hall/CRC, (2009).
 15. Bertram Ludäscher and Mathias Weske and Timothy McPhillips and Shawn Bowers (2009) *Scientific Workflows: Business as Usual? 7th Intl. Conf. on Business Process Management (BPM), LNCS 5701*.
 16. Luis Cabellos, Isabel Campos, Enol Fernández-del-Castillo, Michał Owsiak, Bartek Palak and Marcin Płóciennik *Scientific workflow orchestration interoperating HTC and HPC resources Computer Physics Communications*, **182** (4), pp. 890–897 (2011).

Molecular Dynamics Science Gateway with Vine Toolkit Providing UNICORE Middleware Support

Piotr Dziubecki, Piotr Grabowski, Tomasz Kuczynski, Krzysztof Kurowski, Dawid Szejnfeld, Dominik Tarnawczyk, and Malgorzata Wolniewicz

Poznan Supercomputing and Networking Center,
Noskowskiego 12/14, 61-704, Poznan, Poland

E-mail: {deepres, piotrg, docentt, krzysztof.kurowski, dejw, dominikt, goslaw}@man.poznan.pl

Science Gateway acts as a mediate system between scientists and their computational tools in a form of a web portal. It provides space for communities, collaboration and data sharing and visualisation in a comprehensive and efficient manner. The main objective of such solution is to allow users to access computational resources, process and analyse their data and get the results in a uniform and user friendly way. In this article we present a solution for the Molecular Dynamics field with a use of NAMD application. Particularly our Gateway focuses on the construction of a particle model, preparation of input data for simulation, geometry optimisation and Molecular Dynamics simulation ending with the data analysis and visualisation. Another example of Science Gateway from the area of nanotechnology science will be also described in this paper.

1 Introduction

The advanced web-based graphic and multimedia oriented user interfaces (GUIs) designed for scientists and engineers could change the way users collaborate, share computing experiments and data, and work together to solve day-to-day problems. Moreover, future science and engineering gateways will influence the way users will access not only their data, but also control and monitor their demanding computing simulations using the Internet. To allow users to interact remotely with future supercomputers and large-scale computing environments in a more interactive and visual manner we present a tool called Vine Toolkit that has been successfully used as a core web platform for various gateways^{1,2}. Vine Toolkit is a modular, extensible and easy-to-use tool as well as a high-level Application Programming Interface (API) for various applications, visualisation components and building blocks to allow interoperability between many different HPC and grid technologies like UNICORE 6. It supports Adobe Flex and BlazeDS technologies³ to allow developers to create advanced, rich web applications similar to many stand-alone GUIs. Additionally, Vine Toolkit has been integrated with well-known open source web frameworks such as Liferay⁴ and Gridsphere⁵. In this paper, we briefly describe new technological solutions relevant for advanced scientific portals driven by example scientific needs.

2 Vine Toolkit as a Base for a Science Gateway

Starting from the top of the Vine Toolkit software stack, it provides an efficient and robust user interface framework based on the Adobe Flex and BlazeDs software. Vine allows the integration of the rich internet application standard directly to a browser and enables applications to act and look exactly as their stand-alone versions. Thus, it is possible to

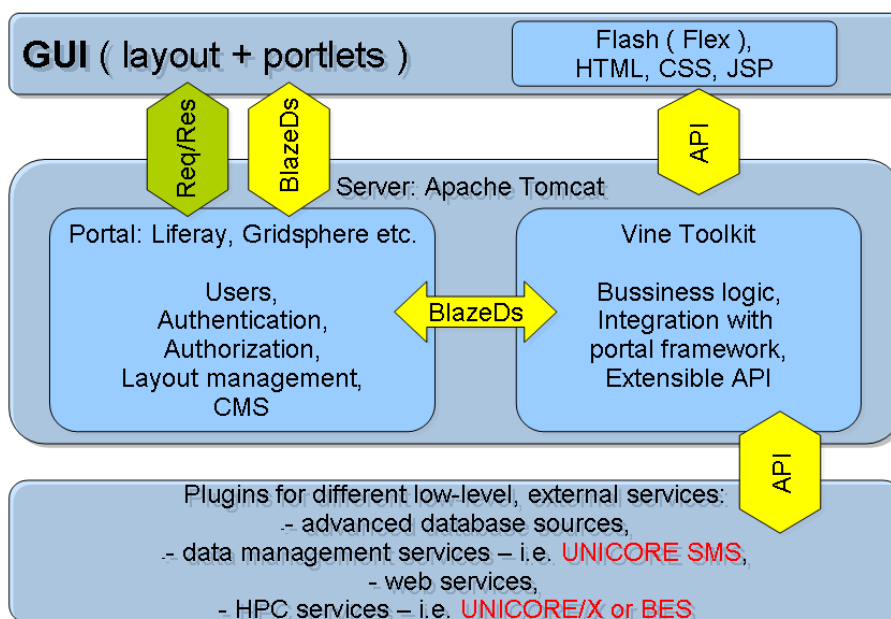


Figure 1. Portal with Vine Toolkit generic schema.

create advanced portal applications like Science Gateways where developers can create web-based versions of many legacy applications and their GUIs. One of the key new requirements for the Vine Toolkit regarding the integration with existing portal frameworks was to enable web application developers to create rich and advanced user interfaces as quickly as possible. Initially, we tried to use JS/AJAX⁶-based frameworks within Vine Toolkit. Various problems related to the software portability in different web browsers encouraged us to migrate to other frameworks for the development and deployment of cross-platform rich Internet applications. Eventually, Adobe Flex was chosen for developing rich and advanced user interfaces in Vine Toolkit. The main reason was the fact that at the time of the project's inception, Microsoft Silverlight was far behind Flex in terms of functionality⁷. Also, the licensing favoured Flex which is open source. Obviously, the presentation layer is a front end to various components and services provided by Vine Toolkit, but it is getting more and more important once advanced web applications are available. Thanks to a pluggable Vine architecture it is possible to extend its base functionality in a uniform way. For instance, at the beginning Vine Toolkit offered only a support for the Globus Toolkit middleware. Currently, it is possible to use the majority of leading middleware stacks: UNICORE 6, GRIA, gLite, QosCosGrid middleware and many other well-known standards, such as OGF JSDL, OGF OGSA-BES or OGF-HPC Profile⁸⁻¹⁰. Technically speaking, a new service in Vine can be added by creating a separate project and implementing a set of predefined APIs. Then, after a proper configuration, it can be used transparently by the end user without any additional changes in the application code. Finally, Vine offers various deployment configurations including standalone mode, web

service mode and more importantly a ready to use integration with portal environments and portlet containers, e.g. Gridsphere or Liferay^{5,4}. Therefore, with a single software stack it is possible to build a complex solution consisting of services, a portal and a set of user-customised applications at once available as a web gateway. Vine was designed to work with well-known JSR-168 open standard and its reference implementation and the Tomcat web application container¹¹. Since version 1.1 Vine Toolkit also supports Liferay JSR-286 enterprise portal¹². Consequently, Vine Toolkit gives its users a great opportunity for creating and delivering production-quality web environments as it covers major web-based development aspects, especially for scientific and computing portals.

2.1 UNICORE 6 Plugin

The UNICORE 6 plugin is based on the UCC client jar libraries. We found some problems related to the multi user access in the portal environment regarding resolving storage locations. It turned out that client jars use ehcache to remember last resolved storage locations. In case of any consequent access by another user, errors related to the access denied problem occurred. To avoid this we replaced suitable classes in the plugin class loader to eliminate cache and to make the storage resolving process totally dynamic. We decided to use x509 proxy certificates as most of today grid middleware use this kind of security credentials. It implies a requirement for the supported UNICORE installations - the UNICORE Gateway has to have the proxy extension installed (currently unfortunately an unofficial extension). It is also possible to configure a common portal certificate and use the default security scheme in UNICORE but then all portal users act as single user on the middleware side and the user accounting has to be done on the portal side what could be allowed in some scenarios if desired. We start to think about supporting SAML assertions in the future with providing a tool for end users like Java Web Start or Java Applet to generate and sign security assertions which will be uploaded to portal and added to the user portal session in order to use it while communicating with the UNICORE server. Thanks to use of the UCC client jars we can use both access modes like BES or UNICORE/X. Configuration of the Vine resource allows also to point out the default storage resource in order to automatically generate target storage locations in job descriptions if desired - for example it is possible to export all outputs to some particular UNICORE storage.

3 Molecular Dynamics Science Gateway

Vine Toolkit has been initially designed as a tool providing services facilitating the access to the HPC resources. After that stage, satisfied with the result we started to add GUI components like Credential Manager or File Browser to help users build their portals and more advanced applications. Recently we have entered into the next phase of our tool lifecycle - now we are building the complete solutions upon the existing framework, incorporating services and components into one entity - the Science Gateway. NAMD Client belongs to the second generation of web components based on the Vine Toolkit framework. First applications created with a use of Vine like Nano Client have been designed to solve specific problems in the area of nanotechnology (like finding the total energy, charge density and electronic structure of systems made of electrons and nuclei within Density Functional Theory). On the other hand we wanted to extend our base of reusable concepts - building

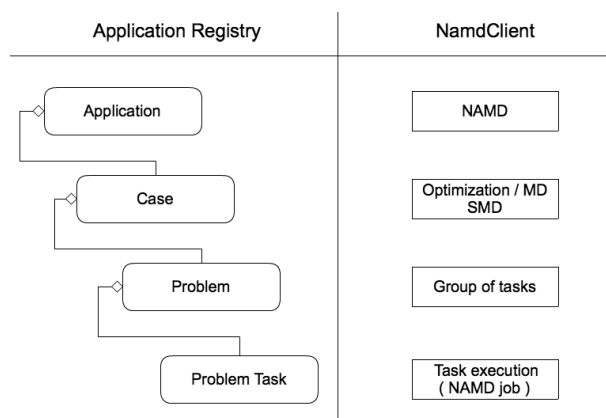


Figure 2. Application registry mapping diagram.

blocks which could be used later to create different applications as well. Having that in mind, we based our software on a set of components, which could be reused in many different contexts. For the NAMD Client we gathered user requirements and created a general view of the new application stack. The whole system is meant to solve molecular dynamics (MD) simulation problems¹³. Scientists would like to have access to the portal where they can define a problem by creating/editing the polypeptide model with the use of a graphical tool or importing data from a local repository as well as from the remote protein data banks. Some input files are created as a result of that process. Next step is to prepare and configure the geometry optimisation (optional) and finally end up with the MD / Steered MD computations. NAMD Client is designed and tested to work on the UNICORE6 middleware architecture. We have prepared a typical scenario for the task execution, it consist of several steps:

1. User upload his data into the temporary directory on the Portal File System (local file system on the server).
2. NAMD configuration file is prepared and tuned by the user.
3. User starts new task.
4. All input files are copied to the destination SMS file service, StageIn section is performed by Vine.
5. NAMD task is executed.
6. StageOut section is performed by Vine, all files from the given task are copied to the remote data file system.
7. User with the use of File Manager component checks and downloads data from the given task.

On each stage there is a need for efficient data Organization - creating a problems library with a set of features facilitating database browsing and finding the desired information.

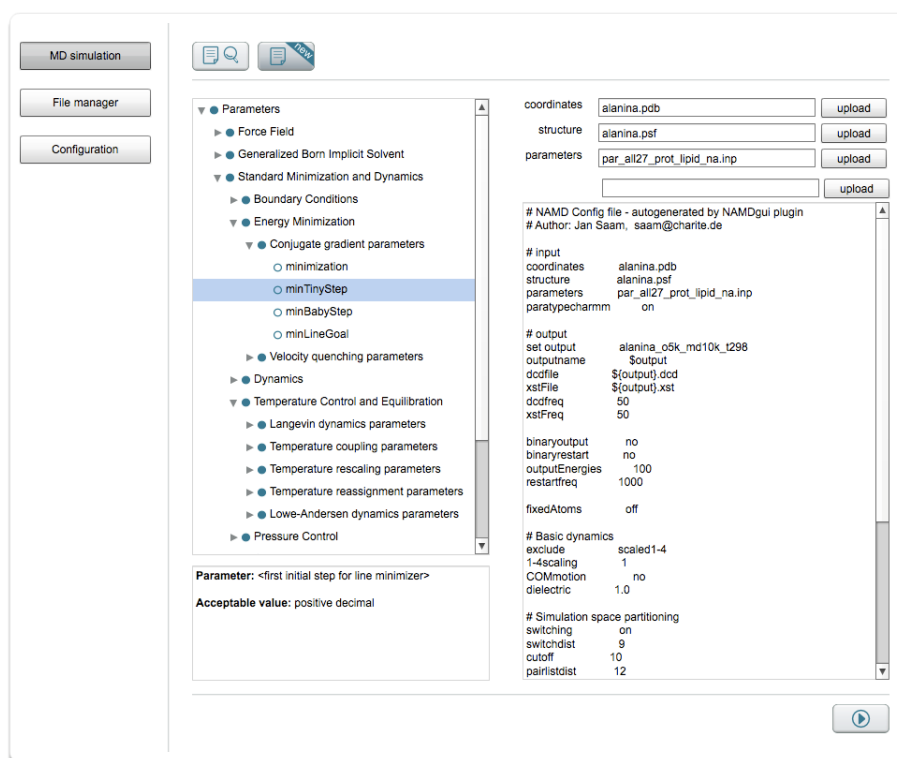


Figure 3. NAMD Client application view.

For these applications there is Application Registry service and UI component used also in NanoClient which will be our base system responsible for the data management. The Figure 2 depicts NAMD Client specific application/case mapping in the Application Registry structure.

Utilisation of the mentioned service lets user work with data in a convenient way especially when dealing with a massive amount of data objects. The next step is to edit/configure a new case. In order to do so we provide form based components with a graphical representation of the configuration file. “Advanced” mode allowing inline modification of the text input files is also available. The final stage is the job submission which is the main added value. It could be a single job or a set of jobs, depending on how many variants user would like to examine. It is worth to mention that parameterised tasks are grouped under a “Problem” category and thus it is easier to analyse and compare the results. Job submission and monitoring are also in a form of reusable code, so they are used without any modifications in order to schedule and observe the task execution process. The last element of the proposed solution is a simple tool allowing for the analysis of computed results. We are working on the log scrapping mechanisms, which let us visualise the partial result in real-time. That feature will provide users with an instant feed-back and he will be able, for instance, to terminate tasks not matching the experiment’s requirements. Cur-

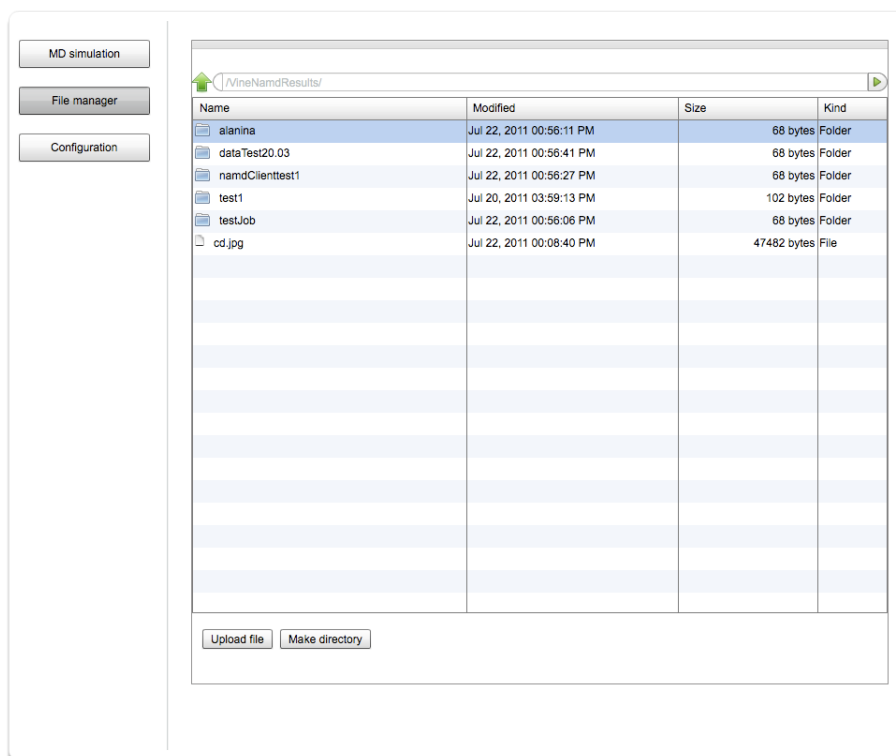


Figure 4. NAMD Client file browser view.

rently we are focusing on the delivering of the core functionality - integration Application Registry, task definition interface with a Job Submission and Monitoring. Later on we will complete the whole system with an editor of polypeptide model and more visualisation tools. Reusing components gives us one more benefit, we are capable of creating a new application, which will be able to incorporate the functionality from the Nano Client and NAMD Client projects. Therefore users could use both components within the context of the same software instance. For the end user it will be just a matter of choosing different task type from the application registry. In parallel to the development task the solution is tested with real middleware, the NAMD package and computational resources. During the prototype test phase the NAMD Client application is tested with UNICORE 6 plugin and UNICORE 6 installation on PL-Grid test infrastructure in Poznan Supercomputing and Networking Center.

4 Nano-Science Gateway

The most advanced part of Nanotechnology Science Gateway is the ABINIT portal client. The ABINIT simulation software package allows users to solve problems like: finding the total energy, charge density and electronic structure of systems made of electrons and

nuclei within Density Functional Theory (DFT), using pseudo-potentials and a planewave basis^{14,15}. ABINIT also includes options to optimise the geometry according to the DFT forces and stresses, or to perform molecular dynamics simulations using these forces, or to generate dynamical matrices, Born effective charges, and dielectric tensors. Excited states can be computed within the Time-Dependent Density Functional Theory (for molecules), or within Many-Body Perturbation Theory. Despite of its usefulness in computations, being a command-line tool, ABINIT requires from the user not only domain-specific knowledge but also some computer science related accomplishments and ABINIT in/out data structures familiarity. In order to hide the complexity and provide a web-based collaborative access to ABINIT, we created many new rich web applications using Vine Toolkit and Adobe Flex. Consequently, we are able to support the transparent web access to sequential and parallel execution of DFT codes deployed on HPC computing clusters available for users in the PL-Grid infrastructure. By providing basic and advanced modes we are able to support both experts and beginners during their simulation studies. Continuous meetings with end users were a great opportunity to gather new requirements, e.g. tools to parse input/output parameters in the form-like panel for ABINIT and other DFT code, e.g. Quantum Espresso. Some parameters that were not reflected in the parameter input form could be now easily added using a new rich editor in the advanced mode. One should note that inputs needed for ABINIT job submission are pseudo-potentials files. Using the built-in Vine Toolkit file manager users can easily access, copy and assign appropriate files. It is also possible to store users' files by taking the advantage of Vine Toolkit files repository as well as advanced access control mechanisms to share them according to defined policies.

After preparing a set of parameters and selecting steering parameters for DFT calculations, the user can add another set of parameters to be solved in parallel. When the whole experiment is ready (typically, it consists of several parameter sets) the user can send the experiment to a grid middleware controlling remote computing clusters. In this approach, we used the QosCosGrid middleware stack together with a metascheduler services called QCG Broker. Therefore, we were able to make the underlying HPC resources fully transparent for end-users, so they could focus only on domain-specific interfaces and problems instead of struggling with computing infrastructure details, such as available processing power, deployed libraries or memory limits. After the job submission the user can monitor all his/her simulations by simply checking the progress bars. An additional monitoring tool is a chart presenting the relative difference between subsequent computation iterations. If the user notes that the difference does not converge during the experiment, he/she may decide to cancel this task in order to save the computing time. It is also possible to cancel the whole experiment - in this case all pending tasks (for corresponding parameter sets) are cancelled.

Immediately after each job completion, the user can see generated results in the portal. As it was mentioned above, all the ABINIT output files are stored in a Vine repository or can be transferred to remote file servers, e.g. GridFTP or more advanced Data Management Services, e.g. iRODS or DMS. The calculated functions of the total density of electronic states (DOS) line charts can be also displayed as visualisation results. It is possible to view multiple series from different tasks on the same chart. The chart can be dynamically cropped and zoomed and special values like the Fermi energy can be also marked on the chart (if it was calculated in given case). For those applications we used Vine extensions based on Adobe Flex. Figure 5 shows the example experiment and it's visualisation results.

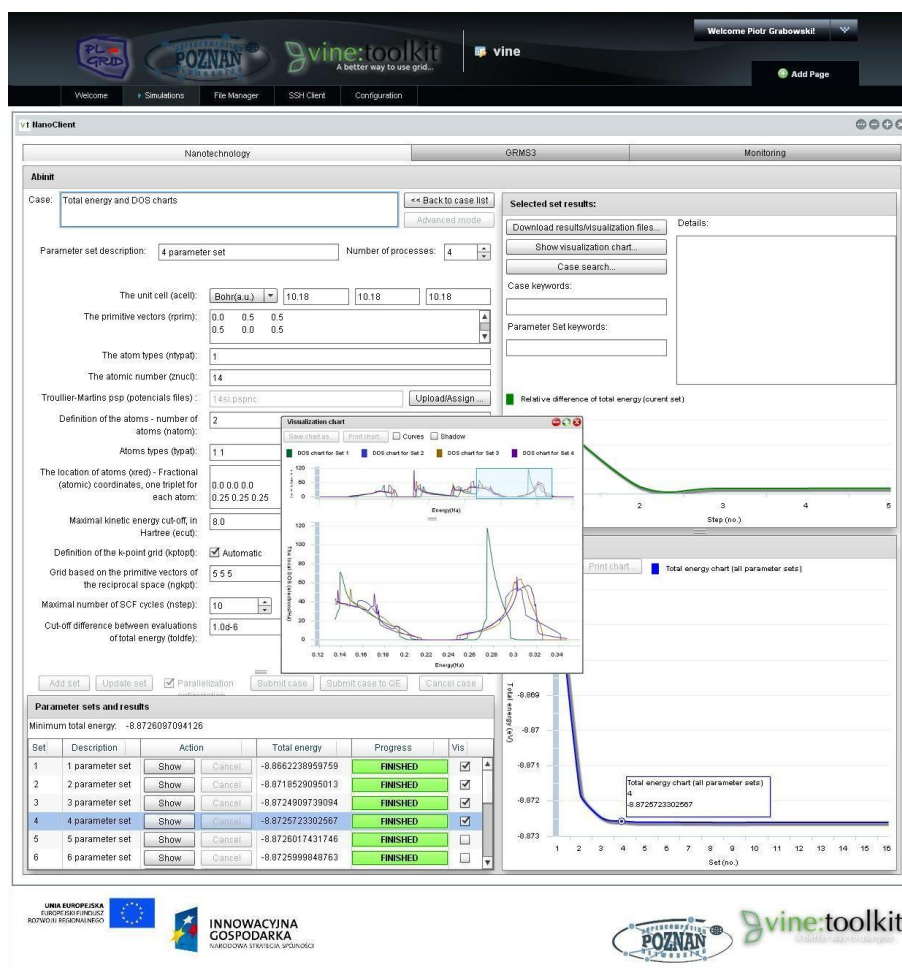


Figure 5. Nano-Science Gateway - ABINIT web client: Total energy calculation and calculated functions of the total density of electronic states (DOS) line charts.

All parameter sets and corresponding computing jobs together with their results are stored as cases for further use. This allows users to start or re-submit long-term calculations from the portal, observe their progress after some time (possibly with another browser and session), view the results of completed experiments and reuse the previous experiments altering some parameters. To enable quick search of a certain example all parameter sets and previous cases can be annotated with keywords by users. Thus, it is possible to look for similar experiment descriptions and results using these tags by other users working on similar data sets. In this case, a special module for web searching is used with two data sources available: Google and ScienceDirect.

5 Conclusions and Future Work

In this paper, we briefly presented our approach to build Science Gateways using our Vine Toolkit integrated with the Adobe Flex software development kit. The presented web-based gateway supports various nanotechnology scientists in execution and management of large-scale simulations on computational grids using molecular dynamics code in NAMD application or DFT code in ABINIT or Quantum Espresso. The described framework offers an effective way to start, monitor and control scientific applications via web based user-friendly graphical interfaces. Currently Vine Toolkit is being enhanced and tested under the national Polish Grid Infrastructure (PL-Grid) project and is planned to be deployed in production environments. The future works on the Vine Toolkit based Science Gateways is divided into two parts: various extensions to the existing NAMD, ABINIT, and Quantum Espresso clients and support for other applications and packages like MOPAC. Many existing users identified several new applications that can be easily added to the gateway, e.g. NWChem or BLAST/CLUSTALW2, and their integration will be a next step. Then, we would like to extend the gateway to support all major applications used for large-scale parallel simulation in nanotechnology and molecular chemistry/biology, thus giving scientists possibility to make their advanced calculations in one collaborative web space.

Acknowledgements

The authors would like to thank all people whose work allowed us to develop aforementioned applications and provide the Nano-Science Gateway for the scientific community. The special thanks goes to people at the Institute of Physics, Poznan University of Technology who provided a lot of useful feedback to user interfaces and generated main requirements. This work has been funded by the PL-Grid project (contract number: POIG.02.03.00-00-007/08-00, website: www.plgrid.pl)¹⁶. The PL-Grid project is co-funded by the European Regional Development Fund as part of the Innovative Economy program.

Keywords

Science Gateway, Web2.0, ABINIT, Vine Toolkit, Liferay, Adobe Flex, Material Science, Molecular Dynamics, UNICORE6, NAMD, Nanotechnology.

References

1. Russell, M., Dziubecki, P., Grabowski, P., Krysinski, M., Kuczynski, T., Szejnfeld, D., Tarnawczyk, D., Wolniewicz, M., Nabrzyski, J.: *The Vine Toolkit: A Java Framework for Developing Grid Applications*. Parallel Processing and Applied Mathematics. Lecture Notes in Computer Science, 4967, 331-340, 2008.
2. Szejnfeld, D., Dziubecki, P., Kopta, P., Krysinski, M., Kuczynski, T., Kurowski, K., Ludwiczak, B., Piontek, T., Tarnawczyk, D., Wolniewicz, M., Domagalski, P., Nabrzyski, J., Witkowski, K.: *Vine Toolkit - Towards Portal Based Production Solutions for Scientific and Engineering Communities with Grid-Enabled Resources Support*. Scalable Computing: Practice and Experience, 11(2), 161-172, 2011.

3. Adobe, Flex/BlazeDs, <http://www.adobe.com/products/flex/>.
4. Liferay, <http://www.liferay.com/>.
5. Gridsphere, <http://www.gridsphere.org/>.
6. Ullman, Ch., Dykes, L.: *Beginning Ajax*. Paperback, ISBN 978-0-470-10675-4, 2007.
7. Microsoft, Silverlight, <http://www.silverlight.net/>.
8. Streit, A., Bala, P., Beck-Ratzka, A., Benedyczak, K., Bergmann, S., Breu, R., Daivandy, J. M., Demuth, B., Eifer, A., Giesler, A., Hagemeyer, B., Holl, S., Huber, V., Lamla, N., Mallmann, D., Memon, A. S., Memon, M. S., Rambadt, M., Riedel, M., Romberg, M., Schuller, B., Schlauch, T., Schreiber, A., Soddemann, T., Ziegler, W.: *UNICORE 6 - Recent and Future Advancements*. JUEL-4319, ISSN 0944-2952, 2010.
9. Kurowski, K., Piontek, T., Kopta, P., Mamoński, M., Bosak, B.: *Parallel Large Scale Simulations in the PL-Grid Environment.*, COMPUTATIONAL METHODS IN SCIENCE AND TECHNOLOGY Special Issue, 47-56, 2010.
10. Kurowski, K., de Back, W., Dubitzky, W., Gulyás, L., Kampis, G., Mamonski, M., Szemes, G., Swain, M.: *Complex System Simulations with QosCosGrid.*, Lecture Notes in Computer Science, Volume 5544/2009, 387-396, DOI: 10.1007/978-3-642-01970-8_38, 2009.
11. JSR-168, <http://jcp.org/en/jsr/detail?id=168>.
12. JSR-286 <http://jcp.org/en/jsr/detail?id=286>.
13. Phillips, J. C., Braun, R., Wang, W., Gumbart, J., Tajkhorshid, E., Villa, E., Chipot, C., Skeel, R. D., Kalé, L. and Schulten, K., *Scalable molecular dynamics with NAMD*. Journal of Computational Chemistry, 26: 1781-1802. doi: 10.1002/jcc.20289, 2005.
14. Gonze, X., Amadon, B., Anglade, P.-M., Beuken, J.-M., Bottin, F., Boulanger, P., Bruneval, F., Caliste, ' D., Caracas, R., Cote, M., Deutsch, T., Genovese, L., Ghosez, Ph., Giantomassi, M., Goedecker, S., Hamann, D.R., Hermet, P., Jollet, F., Jomard, G., Leroux, S., Mancini, M., Mazevet, S., Oliveira, M.J.T., Onida, G., Pouillon, Y., Rangel, T., Rignanese, G.-M., Sangalli, D., Shaltaf, R., Torrent, M., Verstraete, M.J., Zerah, G., Zwanziger, J.W.: *ABINIT : First-principles approach of materials and nanosystem properties*. Computer Phys. Commun. 180, 2582-2615, 2009.
15. Gonze, X., Rignanese, G.-M., Verstraete, M., Beuken, J.-M., Pouillon, Y., Caracas, R., Jollet, F., Torrent, M., Zerah, G., Mikami, M., Ghosez, Ph., Veithen, M., Raty, J.-Y., Olevano, V., Bruneval, F., Reining, L., Godby, R., Onida, G., Hamann, D.R., Allan, D.C.: *A brief introduction to the ABINIT software package*. Zeit. Kristallogr. 220, 558-562, 2005.
16. PL-Grid Project, <http://www.plgrid.pl/en>.

Advancing Cutting-Edge Biological Research with a High-Throughput UNICORE Workflow

Richard Grunzke, Ralph Müller-Pfefferkorn, Ulf Markwardt, and Matthias Müller

Technische Universität Dresden
Center for Information Services and High Performance Computing
Dresden, Germany
E-mail: {richard.grunzke,ralph.mueller-pfefferkorn,ulf.markwardt,matthias.mueller}@tu-dresden.de

In biological research automatic high-throughput microscopes produce large amounts of pictures in a short time interval, which are to be analysed with image processing software. This results in thousands of computing jobs at once as the images can be processed in parallel. In this paper a workflow for such high throughput computing is presented. It was implemented using UNICORE for job management and iRODS to manage the data. The workflow is fully integrated into the biologists working environment.

1 Biological Research and High Throughput Computing

1.1 Automatic Microscopy for Biology

The Max Planck Institute of Cell Biology and Genetics (MPI-CBG) in Dresden conducts research on the molecular mechanisms of absorption and transportation of molecules in cells. In this research parts of cells are highlighted to examine the underlying cellular processes. Genomes are prepared and the resulting reactions create light emissions. These emissions are observed in pictures (see Figure 1), taken by automatic microscopes, and analysed by the image analysis software Motion Tracking, which is developed at MPI-CBG. High ranking publications, for example in Nature magazine¹, are the result.

In their research the biologists arrange experiments in wells on a plate. A well is one of up to 384 compartments on a plate. Each of them contains a slightly different sub-experiment, ranging from different specific genomes to control experiments. Usually, experiments involve many plates. A genome-wide screen, for example, can involve more than 200 plates. After the preparation which can take several months several images at different depths and at ten different positions in each well are taken by an automatic microscope (see Figure 1). Thus, millions of images are acquired that need to be analysed. With file sizes up to 10 MB per image the total size can mount up to several TB for a single experiment. In the future, the biologists plan to record three-dimensional and time-resolved movies, which will require up to several GByte per well.

1.2 Supporting Biological Research with a Grid Infrastructure

To improve and speed up the analysis of the images a high throughput workflow² for image processing and data management was developed and deployed in a joined effort of the Center for Information Services and High Performance Computing (ZIH) of the Technische Universität Dresden. The following preconditions and demands led to the decision to use Grid infrastructures for this workflow.

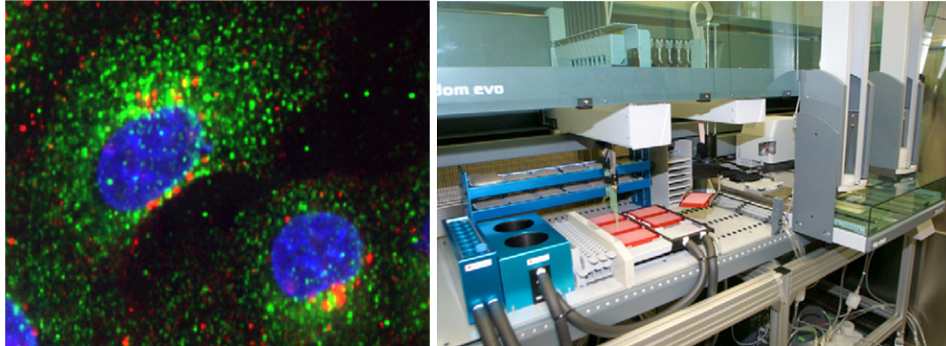


Figure 1. Left: Picture taken by automatic microscope showing highlighted parts of cells, ©Zerial Lab, MPI-CBG; Right: Automatic microscope; ©MPI-CBG.

Computing Demands: The image analysis jobs consist of many short and independent single-core jobs running in farming mode. Each job takes about 10 to 30 minutes to finish. Typically batches of tens of thousands of jobs are submitted at once which exceed the computational capabilities at MPI-CBG. As the research possibilities are only limited by the available computational power the demand is ever growing. These characteristics make it ideal for Grid Computing.

Storage Demands: The storage demands are substantial. For example, a large data set consists of 18 TB in 8 million files. Currently, a total of about 100 TB is stored at MPI-CBG. With the bandwidth between ZIH and MPI-CBG not even reaching 1 GBit/s the transfer of files to the ZIH for analysis became the bottleneck. Storing the data at ZIH would make use of the internal bandwidth of 10 GBit/s. The data is organised in directory structures with metadata encoded in the directory name. Additional metadata is stored in an extra file for each image. Thus, a storage system with good performance and metadata support would be beneficial.

User Demands: The image analyses biologists conduct are controlled by the software Motion Tracking which includes a graphical user interface. Motion Tracking runs on Windows workstations as that is the main platform of the research group. Most biologists do not have a deep interest in computing infrastructures - they want it to “just work” so that they can focus on their research. Keeping that in mind the goal was to make the use of the Grid infrastructure as transparently as possible and to require only minimal knowledge of the underlying technologies and interaction from the users. This meant to fully integrate the Grid infrastructure into their work environment and graphical user interface.

2 Architecture

2.1 Computing Grid

Based on the demands outlined above the UNICORE 6 middleware³ was chosen for the job submission. It is easily integral into heterogeneous systems as it provides platform independent Java components. This was very important since the workstations at MPI-

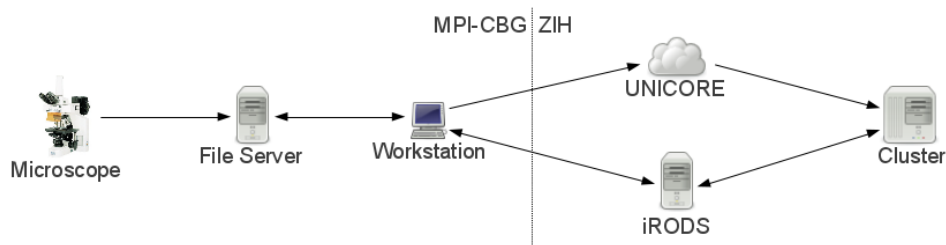


Figure 2. Sketch of the systems hardware architecture.

CBG are running various versions of Windows. UNICORE is easy to deploy and maintain, mature and offers great support and continuous development.

UNICORE was setup in the following way: To access the Grid the UNICORE command line client (UCC) was installed on the Windows workstations. A central registry at ZIH lists all the available services. To process the large number of submissions five UNICORE/X servers were deployed at ZIH to handle the TSIs of the participating clusters. Each server is a virtual machine with four cores with 2.6 GHz each and at least 4 GB RAM. One XUADB each serves the UNICORE/X instances with user information.

2.2 Data Storage

iRODS⁴ is a mature Grid data management system which allows efficient access to possibly widely distributed data. It provides a global namespace, meaning that data, even if spread across different storage resources, is accessible through a unique identifier consisting of path and file name. Additionally, the data can be augmented with metadata, which is stored in a database. The server components enable the integration of different storage resources. The access to data is efficient due to its parallel data transfer capabilities.

At ZIH an iRODS server was deployed on a machine with 16 cores with 2.6 GHz each, 32 GB RAM, 10 GBit/s network connection and 40 TB storage capacity. The initial intention to store all data on it could not be maintained as the amount of data grew much faster than expected. Thus, it is used as a data cache at ZIH to allow high bandwidth access to the data from the clusters. When the storage capacity on the iRODS server reaches the threshold, files that were not accessed the longest will be deleted to free up space. On a Grid cluster iRODS command line clients (icommands) are used to access iRODS. Figure 2 gives an overall view on the system architecture used.

2.3 The User Environment

2.3.1 Integration into the User Environment

As mentioned above the researchers use Windows workstations to conduct their work. The image analysis software Motion Tracking, developed at MPI-CBG, steers the whole workflow. This includes many tasks like managing a large number of images, analysing a few of them manually to find suitable parameters, and executing the analysis on all images

on remote systems. The automatic microscopes are connected to file servers where the original images are stored. These are directly accessed by Motion Tracking for analysis.

To support the use of UNICORE and iRODS the functionality of Motion Tracking had to be extended.

The UNICORE command line client UCC was deployed on the workstations as a Windows service. At the beginning of every batch of analysis tasks Motion Tracking executes “ucc connect” to connect to the UNICORE services and to initialise server processes. Afterwards, “net start uccbatch” is executed to make sure the UCC is running.

For each job Motion Tracking creates a UNICORE job description file using templates. Key words get replaced to create the customised job files. The job description file template is a generic file which specifies imports, an executable, arguments, and exports. Imports are the files to be uploaded to the USpace of the cluster. The imports include a script executed by the Motion Tracking command line client, the parameter file, and a run script. The run script is defined to be executed. The run script template is the template which transfer files from and to iRODS and execute the Motion Tracking binaries on clusters to compute the results. When jobs are created the templates are filled with the real parameters to form the actual job description files and run scripts.

iRODS support was integrated via the use of the iRODS libraries. They allow to access the iRODS server to transfer files.

2.3.2 The Directory Structure on the Workstation

To support the biologists in easily submitting jobs and transferring data from their Windows workstations a directory structure in Windows was set up to hold the needed tools and directories. The Java Development Kit (JDK) in version 1.6 or newer needs to be installed. The structure was tested under Windows XP, Windows XP 64 and Windows 7/64. *C:\unicore* is the root directory of the directory structure to support UNICORE and iRODS. It contains the sub-directories bin, conf, input, jobs, misc, output and update.

bin-Directory

The bin directory holds the following executables and scripts:

- instsrv.exe is used to install the uccbatchservice.exe and register it as a Windows service. It is included in the Windows Server 2003 Resource Kit Tools.
- killucc.bat is the script to exit the running UCC batch program:

```
1 @echo off
2 FOR /F "tokens=1" %%i IN ('c:\unicore\bin\showucc.bat')
3 DO TASKKILL /F /PID %%i
```

- runucc.bat is the script to start the UCC batch program:

```
1 @echo off
2 cd c:\unicore\input
3 ucc.bat batch -X -f -W c:\unicore\conf\siteWeights --max 3500
```

```

4         --u30000 --maxNewJobs 20 --threads 20
5         -i c:\unicore\jobs -o c:\unicore\output
6 net stop uccbath

```

It is only executed by the Windows service.

- showucc.bat is a script to show the running UNICORE Command line Client processes:

```

1 @echo off
2 java -classpath c:\unicore\bin\tools.jar sun.tools.jps.Jps -V |
3     find /i "UCC"

```

It is used by killucc.bat.

- tools.jar is needed by showucc.bat to list the UCC batch programs. The library is part of the Java JDK.
- uccbathservice.exe is the UCC Windows service executable. The Windows service was developed to execute runucc.bat. It assures that only one instance of the UCC is running in the system. This is necessary as biologists can run several instances of Motion Tracking on one workstation.
- update.bat uses UCC to upload files to a computing system in the Grid. This is applied e.g. to update the Motion Tracking software on the Grid systems. It avoids the batch system by executing the update job directly on the login node. Such, the update does not have to compete with jobs in the batch system. This mechanism is only needed on systems where the Motion Tracking binaries are installed by the user and not by the administrators. update.bat contains the following:

```

1 @echo off
2 del c:\unicore\bin\logit
3 echo connect -v > c:\unicore\bin\shellit
4 cd c:\unicore\update\
5 for /f %i in ('dir /b /s /A-D "') do call :ProcessDir "%i"
6     unicore_emilia
7 for /f %i in ('dir /b /s /A-D "') do call :ProcessDir "%i"
8     unicore_deimos_1
9 c:\unicore\bin\ucc\bin\ucc.bat shell < c:\unicore\bin\shellit
10     2>> c:\unicore\bin\logit
11 cd c:\unicore\bin
12 goto :eof
13 :ProcessDir
14 @echo off
15 set test=%~pnx1
16 set test=%test:\unicore\update\=%
17 set test=%test:\=/%
18 echo put-file -s %test% -t u6://%2/Home/%test%
19     >> c:\unicore\bin\shellit

```


- The ucc subdirectory is the UCC installation directory. It has to be added to the PATH variable of Windows.
- UNICORE_Rich_Client contains the UNICORE Rich Client URC which is used to create the user's keystore for authentication during installation.

input-Directory

The input directory is the directory where Motion Tracking copies the files to that are to be uploaded by the UCC to the UNICORE 6 job directory on the target system.

jobs-Directory

The jobs directory is observed by the UCC. Every job description file that is copied into it will be processed and a job will be created, which will be sent through the UNICORE 6 middleware to the Grid.

output-Directory

The output directory is the place to store the job output files downloaded from the UNICORE 6 job directory.

conf-Directory

The conf directory contains the following files:

- the preferences file containing the configuration information used for the UNICORE-client, including the password.
- the keystore of the user with the Grid certificate
- the siteWeights file, which determines the ratio with which the jobs get submitted to the different UNICORE 6 sites. For the ZIH machines it looks like this:

```

1  unicore_deimos_1 = 650
2  unicore_deimos_2 = 650
3  unicore_deimos_3 = 650
4  unicore_deimos_4 = 650
5  unicore_emilia = 550
6  UCC_DEFAULT_SITE_WEIGHT = 0

```

- irodsEnv contains configuration information used to access iRODS

misc-Directory

This directory contains install, configuration, update and test jobs.

update-Directory

The content of the update directory is uploaded by the update.bat script.

3 The Logical Workflow

3.1 Steps of the Workflow

From the user's perspective the logical workflow includes the following steps (see figure 3 for an illustration and the relation of the workflow to the system architecture).

1. Images are acquired by the automatic microscope in huge numbers.
2. The images are stored on a file server in distinct data sets.
3. Then the biologists select files and load them with the image analysis software Motion Tracking. The templates for the job description files and run scripts are filled by Motion Tracking including the paths of the input files in iRODS.
4. The image for each job is synchronised to the iRODS server, meaning it only gets transferred if it doesn't exist already. The files are stored in the same data set directory structure in iRODS as the original data set on the file server. The irsync method of the iRODS Windows libraries is used.
5. Then, after the image for the individual job is synced, the created job file is copied to the jobs directory. UCC, running in batch mode, continuously submits the jobs as they appear there.
6. A UNICORE/X manages and forwards the job to the cluster.
7. The job is scheduled on the destination cluster and the run script gets executed.
8. The run script downloads the job files via the icommands from the iRODS server. It authenticates itself via a proxy certificate created by UNICORE in the job directory on the cluster (USpace).

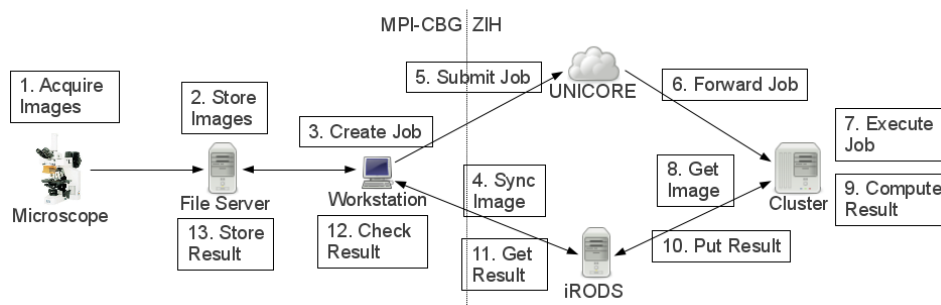


Figure 3. Sketch of the Workflow.

9. The run script executes the Linux command line version of Motion Tracking at ZIH HPC systems to analyse the image and create the result files.
10. After the analysis is finished the results are put back into the iRODS server.
11. Motion Tracking regularly checks for news results and fetches them via the iRODS irsync method.
12. Motion Tracking checks the results for correctness.
13. Motion Tracking saves correct results by integrating them into the data set on the file server.

3.2 Levels of Authentication

Several levels of authentication are used in the workflow.

When the UCC on the workstation connects to UNICORE 6 the user is authenticated with his/her Grid certificate stored in a keystore. It is secured with a password which is saved in a preferences file.

For the file transfers to iRODS via Motion Tracking the user authenticates himself/herself with his/her iRODS login/password combination. This is saved in the iRODS configuration file.

When a USpace for a job is created on a Grid system a proxy certificate, derived from the user certificate, is automatically generated by UNICORE and placed in the USpace. It is used by the iRODS command line tools to authenticate against the iRODS server. This allows the job to transfer files without the need for the user to enter a password or to send the password along with the job to the cluster.

4 Summary and Outlook

The main advantage of the Grid workflow described above is its seamless integration into the work environment of the biologists. From the users point of view not much has changed – the same familiar graphical user interface is used. But the capabilities have been increased dramatically. An analysis can be done in a much shorter time as the submission of jobs is less hindered by the available bandwidth and more computing resources are usable.

From the developers point of view both the integration of UNICORE 6 and iRODS into the existing environment were relatively smooth. The cooperation especially with the UNICORE developers was very good. When the biological workflow was extended characteristics of how analysis jobs and data are handled were kept in mind and a number of issues had to be solved.

A drawback of UNICORE was the sub-optimal support for High Throughput Computing – namely the submission of a large number of jobs at once. The submission was slow due to high loads caused by the UCC on the client and UNICORE/X on the server side. These problems were mitigated by more powerful clients and the deployment of several UNICORE/X instances for a single cluster to handle the submission of a huge number of jobs more efficiently.

The space-based approach^{5,6} was developed to significantly reduce the impact of these problems by providing a central job cache. Just before the UNICORE Summit 2011 the approach was officially included in the UNICORE distribution with a new version. The approach is planned to be integrated into the Motion Tracking workflow in the near future.

References

1. Collinet, C. et al., *Systems survey of endocytosis by multiparametric image analysis*; Nature, 464(7286), pp 243–249, Macmillan Publishers Limited, March 2010.
2. Grunzke, R., *Design of a Data Management Workflow in a High Throughput Grid Environment*, Diploma thesis, ZIH Internal Report, ZIH-R-0904, 2009.
3. Streit, A. et al., *UNICORE 6 - Recent and Future Advancements*, in *Annales des Télécommunications*, 65:11-12, pp 757-762, DOI: <http://dx.doi.org/10.1007/s12243-010-0195-x>, 2010.
4. Rajasekar, A. et al., *iRODS Primer: Integrated Rule-Oriented Data System*, Synthesis Lectures on Information Concepts, Retrieval, and Services, DOI: <http://dx.doi.org/10.2200/S00233ED1V01Y200912ICR012>, Morgan & Claypool Publishers, 2010.
5. Schuller, B.; Schumacher, M., *Space-Based Approach to High-Throughput Computations in UNICORE 6 Grids*, in: *Proceedings of 4th UNICORE Summit 2008 in Springer LNCS 5415, Euro-Par 2008 Workshops: Parallel Processing*, pp 75 - 83.
6. Grunzke, R.; Schuller, B., *Secure High-Throughput Computing Using UNICORE XML Spaces*, in: *Proceedings of 6th UNICORE Summit 2010 in IAS Series Volume 5*, pp 27 - 35.

A Web Framework for Workflow Submission and Monitoring via UNICORE 6 based on Distributable Scientific Workflow Templates

Sandra Bergmann¹, Bastian Demuth¹, and Volker Sander²

¹ Jülich Supercomputing Centre,
Research Centre Jülich, 52425 Jülich, Germany
E-mail: {s.bergmann, b.demuth}@fz-juelich.de

² FH Aachen University of Applied Science,
52066 Aachen, Germany
E-mail: v.sander@fh-aachen.de

The UNICORE workflow system is capable of executing scientific workflows which are described by XML documents. These workflow descriptions are composed of job descriptions that define the remote execution of scientific software programs, as well as transitions, loops, and conditions that steer the control flow during execution. The UNICORE Rich Client (URC) is a graphical client which makes the workflow system's functionality available to the user. It features a graphical workflow editor for creating and modifying workflow descriptions. In the scope of this work, a novel export function has been developed, allowing for the generation of UNICORE workflow templates from the URC's workflow editor. Workflow templates are workflow descriptions that include place-holders for selected input parameters. These place-holders can later be substituted with different parameter values in order to obtain workflows describing slightly different in-silico experiments. The process of re-parameterising workflow templates should require much less experience than creating workflows from scratch with the URC's workflow editor. In order to create a comfortable graphical user interface for managing and submitting workflow templates, a Web framework has been implemented based on Apache Wicket, an open source Java project for flexible and efficient Web application development. Exported workflow templates can be easily uploaded to this Web application. The missing input parameters can be filled in with automatically generated Web forms implemented in Groovy. The resulting workflow descriptions can be submitted to the UNICORE workflow system and the execution status can be monitored via the Web application. Upon successful execution of embedded jobs, output files can be downloaded directly from the Web-based user interface.

1 Introduction and Motivation

Workflows in UNICORE consist of activities representing Grid jobs or control structures such as loops or if-statements and transitions representing the flow of control. They are modelled and visualised as compound directed graphs with nodes corresponding to Grid jobs or other atomic activities and subgraphs corresponding to control structures. Activities are characterised through input and output parameters, e.g. Grid jobs describe the remote execution of scientific applications that require specific arguments, loops are defined by specifying an iteration variable and a terminal condition.

With the aid of the workflow editor in the URC³, users can create complex workflows and specify their attributes. However, tapping the editor's full potential requires a lot of experience. This work targets those users who do not want to create workflows from scratch and deal with this complexity but re-use predefined workflows and adapt some values for their experiments.

This is the underlying model: An expert user with sufficient experience and knowledge creates a workflow via the URC’s workflow editor and exports the corresponding workflow template. This template can then be shared with end users who parameterise and re-submit it via the developed Web application. For supplying parameters, simple Web forms are used. These forms can be created automatically and support extensive parameter validation by evaluating available metadata describing the workflow parameters.

2 Export Function

The expert user creates workflow templates via the newly developed export function in the URC. The export function is embedded in a deployable Eclipse plugin and contains four easy steps (Figure 1), which will be explained in what follows.

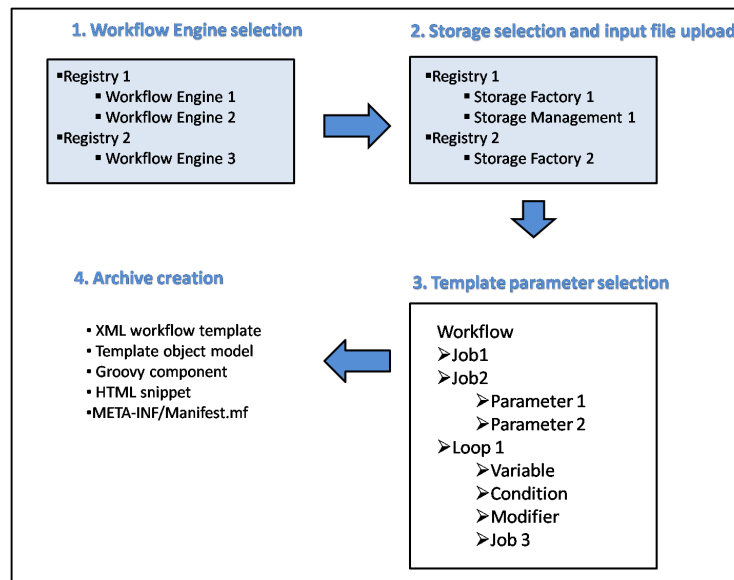


Figure 1. Export process.

Workflow engine selection and Storage selection for input file upload

The first step of the export function is the selection of a workflow engine, which is chosen by the expert user. The selected workflow engine will be used for executing all workflows based on this workflow template. By making this choice during template creation, the end user does not have to deal with such execution details. In fact, the existence of a whole set of Grid services involved in workflow execution is hidden as it should not concern the end user at all. The second step deals with selecting a Grid storage for uploading and safe keeping of those workflow input files that reside on the expert user’s hard disk.

Template parameter selection

Next, all parameters which should be substituted by place-holders are selected by the expert user. Currently primitive job parameters, files, file sets, workflow variables, variable modifiers and terminal conditions are possible template parameters.

Archive creation

The last step is the creation of a compressed workflow template archive. This Java archive (‘.jar’ file) contains the XML workflow description, a template model file and two additional files describing the outer appearance of the Web form in the Web application. These Web form files, a Groovy and an HTML description, are used to present the selected template parameters in a simple user interface. An expert user with sufficient programming skill can modify these files in order to obtain customised forms specific to the given workflow.

3 Web Application

The design of the Web application is based on the current UNICORE Web client development. Main functions that operate on workflow templates include upload, manipulation and deletion of templates; functions dealing with workflow execution include submission, output fetching and deletion of running workflows.

3.1 Wicket and Groovy

Apache Wicket¹ is a Web application framework allowing an efficient and easy development of Web applications based on Java and HTML. It merges stateless HTTP with stateful Java programming, thus being able to deliver complex interactive Web pages. The first version was released in 2004.

Wicket is based on the widely known model-view-controller principle trying to separate a page’s business logic and graphical representation layers (the model contains the data and business logic, the view corresponds to the graphical representation, the controller layer creates the view layer from the model and mediates between the two). By using asynchronous JavaScript and XML (Ajax²) a high level of interactivity and immediate responsiveness can be realised.

For Wicket container components such as pages and panels, an HTML description is necessary in addition to the Java class describing its behaviour. In our system, the Web forms for workflow template visualisation are realised with Wicket panel components, hence the necessity of creating a Java class and an HTML description file. In order to avoid Java source code compilation while exporting the template, Groovy⁵ files are used which can be compiled into Java bytecode at runtime.

Groovy is a scripting language for Java influenced by programming languages like Python, Ruby, and Smalltalk. The syntax is similar to Java but has been simplified to fit the needs of a scripting language.

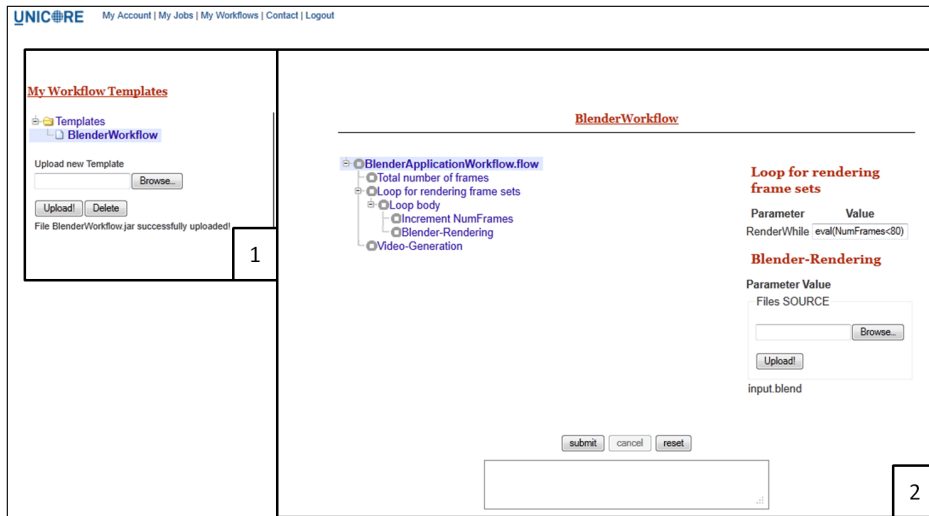


Figure 2. Visualisation of uploaded workflow templates.

3.2 Functionality

3.2.1 Workflow Template Management Operations

Figure 2 shows the the most important functions for workflow templates. In the upper left corner (1) previously uploaded workflow templates are shown. New templates can be uploaded and unneeded templates can be deleted. When a template is selected, the associated parameterisation Web form is displayed at the centre of the page (2).

The workflow structure is represented by a tree, where nodes correspond to atomic workflow activities, control structures, or parameters. Atomic activity nodes can only have parameter nodes as their children, representing their in- and output parameters. In contrast, control structure nodes can have child nodes representing their parameters, atomic activities, or other control structures. There are controls for modifying parameter values to the right of the parameterisation form. If no specific node is selected, controls for all editable template parameters are shown. If an activity node is selected, only parameters of the corresponding atomic activity or structure are shown. Depending on their type, parameters can be modified via text fields, combo boxes, or controls for file upload. All user input is validated against parameter metadata and clear error messages are displayed when a parameter value does not meet the requirements, thus preventing users from submitting Grid jobs with invalid arguments. The screenshot displays an example workflow for rendering a 3D scene with the well-known raytracing application Blender. With this workflow, equally sized sets of frames can be rendered in parallel on different computers on the Grid. Once all frames have been rendered, the resulting image files are combined into a single animation file in Windows Media Video (.wmv) format. The template contains two editable parameters, an input file called “SOURCE” corresponding to the 3D scene to be rendered and a terminal condition for the parallel for-each-loop called “RenderWhile”.

With the latter parameter, the total number of rendered images can be determined. In order to make workflow input files accessible to the UNICORE workflow system, a UNICORE Grid storage is deployed on the same host as the Web application. Files that are uploaded through the parameterisation form of the Web application can be downloaded via this Grid storage.

3.2.2 Workflow Processing Operations

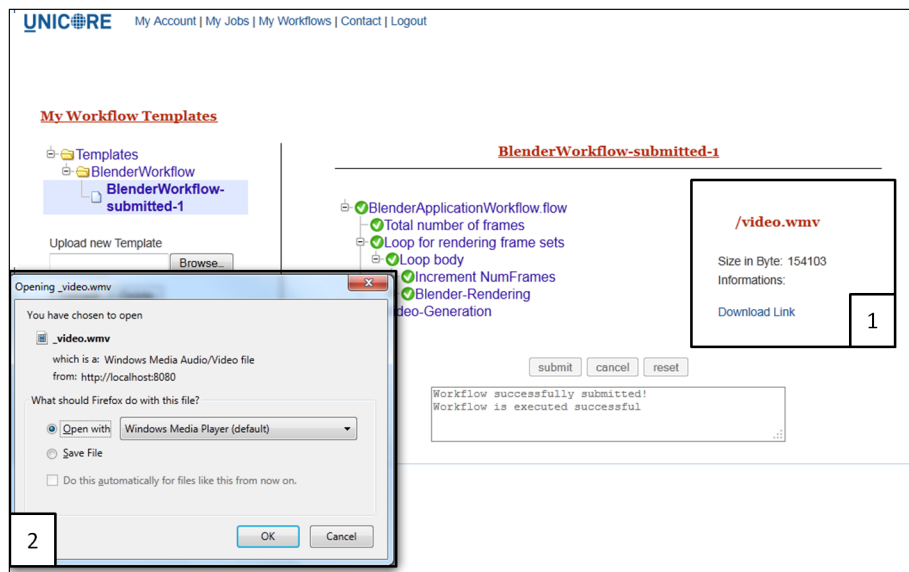


Figure 3. Monitoring of a submitted workflow and output fetching.

After finalising the workflow template, the end user can submit the workflow to the previously selected workflow engine for execution. The current execution state is visualised with small coloured icons (yellow means “running”, green means “finished” and red means “failed”) and resulting output files can be downloaded to the user’s local disk. Figure 3 shows a submitted workflow called “BlenderWorkflow-submitted-1” which was executed successfully. The main output file named “video.wmv” is selected (1) by the end user who wants to download and open this file (2).

4 Concluding Remarks

Based on the workflow editor of the UNICORE Rich Client and the current UNICORE Web client framework, a new Web application for sharing, editing and executing workflow templates has been created.

It offers a rich set of features and provides end users with a high level and highly interactive interface that hides all low level implementation details of the UNICORE workflow system. Expert users can prepare and share workflow templates with minimal effort and end users benefit from advanced features such as automatic creation of suitable parameter controls and input validation. In addition to extensive validation, tooltips display the expert user's parameter descriptions for clarifying the semantics of each and every parameter. There is, as always, room for future improvements: The visualisation of workflow templates in the Web application is currently achieved using a tree viewer as shown in Figure 2 and Figure 3. A more suitable graph-based workflow representation can be realised by deploying off-the-shelf JavaScript graph drawing toolkits, e.g. the JavaScript InfoVis Toolkit (JIT⁴).

In the future, expert users should be enabled to directly upload workflow templates from the URC to the Web application. Currently the expert user stores the archive on his local disk and distributes it to other users who can upload it to the Web application.

The Web application manages workflow templates on a per-user basis. Extending the current template repository mechanism, it should be possible to share templates via the Web application interface, so that other users could obtain workflow template copies, manipulate them and submit the finalised workflows to the workflow system. Taking up ideas from social networking platforms and other Web 2.0 services, end users might be enabled to comment on and evaluate workflow templates from other users. It should be possible to sort templates and to search for templates by name, category or other attributes.

References

1. M. Dashorst, *Wicket in Action*, (Manning Publications Co., 2009).
2. J. J. Garrett, *Ajax A new Approach to Web Applications*, 2005 <http://adaptivepath.com/ideas/essays/archives/000385.php>.
3. UNICORE Team, *UNICORE Rich Client user manual*, 2011, <http://www.unicore.eu/documentation/manuals/unicore6/files/urc/>.
4. JavaScript InfoVis Toolkit, <http://thejit.org/>.
5. G. L. D. Koenig, P. King and J. Skeet, *Groovy in Action, Second Edition*, (Manning Publications Co., 2011).

Integration of UNICORE 6 into the Gatlet Portal Framework

Torsten Antoni¹, Andreas Bender¹, Bastian Boegel², and Stefan Bozic¹

¹ Steinbuch Centre for Computing (SCC)
Karlsruhe Institute of Technology (KIT)
76128 Karlsruhe, Germany
E-mail: {torsten.antoni, andreas.bender, stefan.bozic}@kit.edu

² Ulm University
89069 Ulm, Germany
E-mail: bastian.boegel@uni-ulm.de

UNICORE 6 is one of the major Grid middlewares in the European Middleware Initiative (EMI). Due to the fact that the number of available web interfaces for UNICORE 6 is still limited, this paper shows a solution how the middleware has been integrated into Gatlet - a Grid portal framework. With the integration of UNICORE into Gatlet, portal users can submit jobs to sites, monitor the job status or browse storage resources directly from a web browser. This approach has the advantage that the user doesn't have to install any client software. Furthermore a portal is accessible from any computer that has a web browser installed. Application developers can implement Gatlet based portlets for their software, therefore it is easy to bring the application from the Grid to the user's browser.

1 Gatlet Overview

The aim of the Gatlet¹ project is to provide a tool for easy access to major Grid middlewares and storage resources via web browser. Currently the framework supports access to Globus 4, gLite 3.1, GridFTP and SRM resources. With a portal built on this framework a Grid user can submit and monitor jobs and manage files on storage resources. Grid infrastructures are mapped in the portal to entities like sites, clusters, corresponding middlewares and VOs. Gatlet is shipped with several portlets (e.g. for Job Submission, Monitoring, Resource Management). The Gatlet framework is Open Source and written completely in Java. The name Gatlet is a combination of GAT (Grid Application Toolkit)² and Portlet. GAT provides access to several major Grid middlewares and storage resources through an API. It is easy to extend GAT with new middleware types or versions. GridSphere is used as portlet container for the portlets shipped with Gatlet and respectively custom portlets for scientific applications. GridSphere allows users to authenticate against the portal with X.509 certificates. Grid security issues are handled by Gatlet via MyProxy³. The framework uses a database to store data for submitted jobs, Grid resources and application data. Through the use of JPA (Java Persistence API) all common vendors of relational databases are supported. The Gatlet service API enables access to this database and GAT. The integration of UNICORE 6 into the existing architecture made it necessary to write a new GAT adaptor and to support the Security Assertion Markup Language (SAML) in GAT and Gatlet. Figure 1 gives an overview of the architecture of a Gatlet based portal with UNICORE 6 support. The following chapters describe the involved components in more detail.

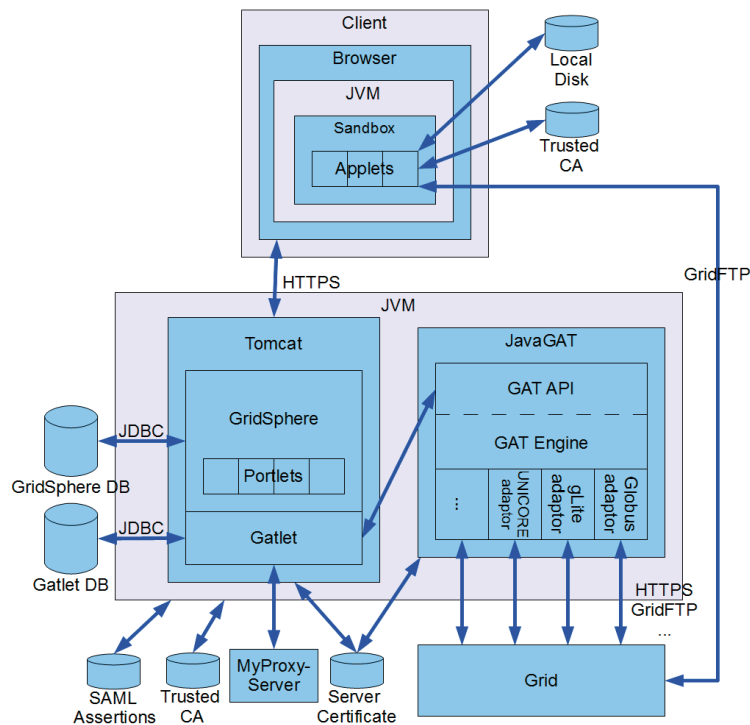


Figure 1. Architecture overview of a portal build with GridSphere, JavaGAT⁶ and Gatlet.

2 UNICORE 6 Security

The security of UNICORE 6 is based on open standards⁴. X.509 certificates issued by a trusted Certification Authority (CA) are used for clients and servers and all communication between them is encrypted via SSL. Furthermore a UNICORE site must provide a gateway as single entry point for service requests. The gateway is responsible for forwarding all requests to a target site. This has the advantage that a site's firewall has a single entry point. Beside that, the gateway is responsible for authentication and authorization of a service request. The gateway checks if the client-authenticated SSL connection is valid. UNICORE is using SAML⁵ for Trusted Delegation. A UNICORE server publishes its identity information (DN) to a Service Registry. The Service Registry can be queried by a client for available server DNs. With this information a client can issue a Trusted Delegation to a server. With this Trusted Delegation the server has the right to act in the name of the client e.g. for accessing storage resources or involve further computing resources to fulfil a job's requirements. A Trusted Delegation is expressed by SAML which defines a standard for exchanging authentication and authorization data between security domains. Using SAML is a secure approach for handling Trusted Delegation because it contains only public information like the user certificate respectively the server's DN.

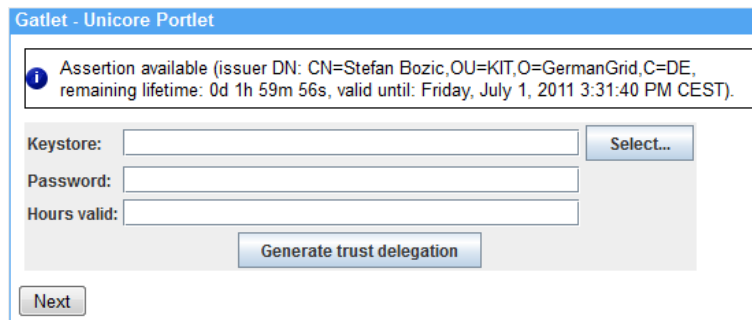


Figure 2. Screenshot of the ETDApplet embedded in a Gatlet portlet.

The basis for supporting UNICORE 6 in Gatlet is the integration of a SAML based Trusted Delegation creation process. This is because the portal must be able to access UNICORE services in the name of the user. Due to the fact that a user must not store his Grid credential (including the private key) on the portal, the creation of the Trusted Delegation must take place at the user side. The Jülich Supercomputing Centre has developed the ETDApplet that creates SAML based Trusted Delegation based on the user's Grid certificate. The applet can be embedded into a portlet assumed that the user has Java Virtual Machine installed on his device. Another requirement is that the portal server must have a valid host certificate that is accepted by UNICORE services. The subject of the client issued SAML assertion will be the DN of the portal's certificate. With the Trusted Delegation the portal is able to use UNICORE services in the name of the user. Figure 2 shows the Gatlet UNICORE Portlet where the ETDApplet has been included. The applet creates the Trusted Delegation and will automatically upload it to the portal server. All SAML Assertions are stored temporarily on the portal's file system and will be accessed by a security context of JavaGAT when using the UNICORE adaptor which is described in detail in the next section.

3 JavaGAT - UNICORE 6 Adaptor

JavaGAT⁶ is responsible for accessing Grid resources in the Gatlet framework. For each supported Grid middleware and protocol a JavaGAT adaptor exists. An adaptor has its own classloader which allows the toolkit to work with different Grid middlewares at the same time. The modular concept of JavaGAT allows developers to extend the toolkit easily with new adaptors. The new UNICORE 6 adaptor that has been developed for the integration is using the UNICORE 6 implementation of the High Level API (HiLA)⁷ to access UNICORE resources. HiLA is an abstract API for accessing Grid resources of different middlewares in a consistent manner. The API is based on the key concept that every Grid resource can be referenced by a URI. Examples for UNICORE 6 HiLA URIs:

```
unicore6:/sites/Alpha-Site/storages/home
unicore6:/sites/Alpha-Site/tasks/79258434...f160cb28c
```

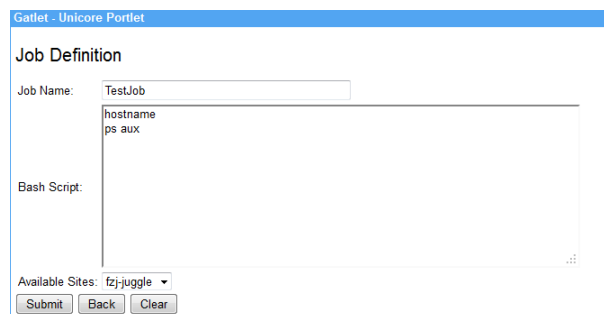
For programmatical usage an URI is mapped to a Location that has a method that can locate the Resource behind this Location. Resources can be instances of Task, Storage or Site which offers a lot of functionality to developers for interacting with UNICORE resources.

```
Location loc = new Location("unicore6:/stefan.bozic@sites/");
Resource res = loc.locate();
```

By using the trust delegation support of HiLA, the adaptor accepts service requests from multiple users at the same time, which is indispensable for the work in a portal. All security relevant preferences will be configured in `~/hila/unicore6.properties`. Because the adaptor is acting on behalf of a user when requesting UNICORE 6 services, it needs a Trusted Delegation to fulfil the task. After the user has created a valid Trusted Delegation with the ETDApplet, the assertion is uploaded to the portal and stored as a file under `~/hila/saml-assertions`. The file name is the same as the user's id in the portal to define a consistent mapping. Once an assertion is stored in this folder, the UNICORE adaptor has the ability to request UNICORE 6 services for the user until the assertion has expired. Note that the keystore definition for HiLA must include the host certificate of the portal, because all SAML assertions have been issued to the portal.

4 Gatlet UNICORE

Gatlet is shipped with a portlet (Figure 3) that allows users to submit script jobs to a UNICORE site. At first the user must create a trust delegation. Then he has to define a job name, a bash script and a target site for his job. Gatlet will validate and process the user input and pass it to JavaGAT. Finally the UNICORE adaptor will submit the job to the target UNICORE site. When the job has been submitted, a UNICORE job id will be returned.



The screenshot shows a web form titled "Gatlet - Unicore Portlet" with a "Job Definition" section. It contains a "Job Name" field with the value "TestJob", a "Bash Script" text area containing "hostname" and "ps aux", and a dropdown menu for "Available Sites" set to "fzj-juggle". At the bottom are "Submit", "Back", and "Clear" buttons.

Figure 3. Screenshot of the UNICORE Script Portlet.

The id and the job parameters are stored in the Gatlet database. With these information it's possible to monitor and control the job. Based on the open and abstract API of Gatlet, UNICORE jobs fit seamlessly into the Gatlet Monitoring Portlet (Figure 4) which can

display job instances of various middlewares like UNICORE, Globus or gLite. The portlet allows users to refresh, stop or view details of UNICORE jobs. Furthermore the working directory of each job can be browsed.

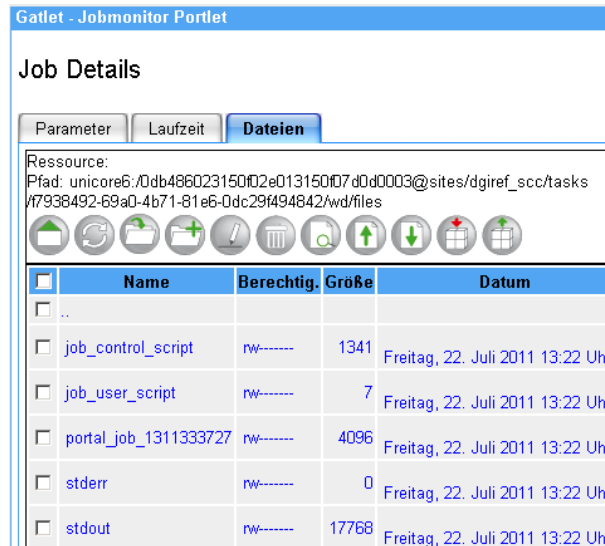


Figure 4. Screenshot of the Job Monitor Portlet which shows the working directory of a job.

5 Conclusion

UNICORE 6 has been integrated successfully into Gatlet. One of the major issues was the support of Trusted Delegation in several components of the architecture. Trusted Delegation builds the base for allowing a portal to access UNICORE services on behalf of a user. The ETDApplet creates such Trusted Delegations in form of SAML tokens and submits them to the portal. A new portlet allows portal users to submit script jobs to UNICORE sites. These jobs can be monitored in the portal with Gatlet Monitor Portlet.

References

1. Gatlet <http://www.gatlet.de>
2. Gabrielle Allen et al. *The Grid Application Toolkit: Towards Generic and Easy Application Programming Interfaces for the Grid* Proceedings of the IEEE Volume 93, Issue 3, Pages 534-550, March 2005.
3. J. Novotny, S. Tuecke, V. Welch. *An Online Credential Repository for the Grid: MyProxy* Proceedings of the Tenth International Symposium on High Performance Distributed Computing (HPDC-10), IEEE Press, August 2001.

4. Krzysztof Benedyczak, Piotr Bala, Sven van den Berghe, Roger Menday and Bernd Schuller *Key aspects of the UNICORE 6 security model* Future Generation Computer Systems Volume 27, Issue 2, Pages 195-201, February 2011.
5. S. Cantor (Ed.) *Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0* OASIS Standard, 15 March 2005.
6. JavaGAT <http://www.cs.vu.nl/ibis/javagat.html>.
7. HiLA User Guide <http://www.unicore.eu/documentation/manuals/unicore6/files/hila-2.2/hila.html>.

User Defined Runtime Environments Using UNICORE

Björn Hagemeyer and Kiran Javaid

Jülich Supercomputing Centre,
Research Centre Jülich, 52425 Jülich, Germany
E-mail: {b.hagemeyer,k.javaid}@fz-juelich.de

Vitalisation plays a major role in IT services today. It allows for reducing the number of physical machines in use by consolidating services onto virtual machines. Computing centres benefit from vitalisation by being able to dynamically relocate virtual machines to different physical machines and separating software from hardware maintenance.

Users can exploit the mobility of virtual machines by easily reusing them in new vitalised environments. Thus, users only need the possibility to run virtual machines in order to be able to take their runtime environments with them. This is the approach behind the user defined runtime environments described in this paper.

1 Introduction

The central concept behind user-defined runtime environments is the availability of software in a well-defined environment. Particular versions of libraries, directory layout etc. are such that the application in question can directly execute without further configuration. Thus, not all software that users may want to run needs to be available and maintained by the computing centres. Software maintenance and updates are done in the virtual machine images, e. g. at the virtual Organization (VO)¹ or research community level. This reduces the amount of software that the administrators of a computing centre have to install and maintain and leaves the installation of domain specific software to the software experts.

A virtual machine (VM) image is an image of a machine's hard drive. Whereas several formats to store these images exist, we consider the generic case of a binary copy of the drive's contents, also referred to as *raw* image. VM images are one of the building blocks of our solution. In addition to the actual VM images, metadata about them will be stored. This metadata contains information about the installed applications and the requirements to be met by the VM manager (VMM) like *processor architecture*, *main memory*, and the available *disk space*. Application information is stored in the same way as the IDB entries for static UNICORE sites. This information allows for a flexible search for particular machine images providing features requested by a user, typically applications.

The approach described here allows for Software-as-a-Service scenarios² where the physical location of execution is unknown beforehand. With VM images provided by VM image repositories, these images can be run in just about any computing centre matching the requirements of the image. Software-as-a-Service would be implemented as a higher level service on top of the ability to explicitly run particular VM images, allowing for fully automated management of the VM and the submission of jobs to it.

We will discuss some related work in section 2. In section 3, we will give an overview over the architecture and how it fits in with the already existing UNICORE architecture, followed by more details about the implementation in section 4. The approach proposed in this paper supports Software-as-a-Service scenarios, which will be outlined in section 5.

2 Related Work

Dharanikota et. al³ have started a previous attempt to introduce virtualisation to UNICORE, which has been abandoned. Their work was based on Xen⁴ as the Virtual Machine Manager (VMM). At the time, this restriction may have been valid, as not many VMMs were available, at least on the LINUX platform. Times have changed, and with the advent of a number of other solutions, it is absolutely necessary to support more than a single one of these. Moreover, this initial approach introduced an entirely new approach to the Target System Interface (TSI), which was based on Java RMI communication. We propose to leave as many of the existing UNICORE components as possible unchanged. The dynamic configuration of the TSI, which is necessary in the proposed architecture can be achieved by other means than rewriting the component (cf. section 3.4).

“The gLite Standard Worker Node is a set of clients required to run jobs sent by the gLite Computing Element via the Local Resource Management System.”⁵ These clients are installed in the compute nodes of a cluster, in order to be able to send gLite⁶ middleware jobs to them. The Worker Nodes on Demand Service (WNoDeS)⁷ adds on top of this approach by providing predefined VM images including these tools. Thus, the images need to be deployed and can be used to run gLite jobs in environments, where image deployment is possible. Such clusters then only need to deal with the management software required to run the images. WNoDeS provides a good integration with a batch system, such that the execution of virtual machines is scheduled alongside other, ordinary compute jobs.

The globus alliance’s⁸ Virtual Workspaces⁹ is an incubator project that develops Nimbus¹⁰. “Nimbus is an EC2-interface compatible remote service that allows authorised clients to deploy environments (virtual machines) described by meta-data on a specified resource quota. The environments that can be deployed in this way range from single VMs to coordinated virtual clusters and more complex constructs.”⁹

3 Fitting the Model

UNICORE is based on an architecture that fits the use cases we describe here quite well. The Target System Factory (TSF) is the unique entry point to a site of computational resources, and the Target System (TSS for Target System Service) is created by the factory for each user. However, in contemporary UNICORE, all TSSs at a site represent the same physical resources, i. e. same cluster.

It is but a small step from this model to extend the TSS such that each TSS resource represents its own compute resource or virtual machine. The overall UNICORE architecture remains exactly the same. Changes are required in the interfaces of the TSF and TSS and the respective implementations. Clients will need to be able to make use of the altered interfaces.

Figure 1 displays an overview of the architecture and the interplay among the various components. As in the existing UNICORE model, a client communicates with the TSF and TSS services. In addition, it will communicate with a VM image repository to discover VM images to deploy. Once a VM image matching a user’s needs has been found, the client requests the creation of a TSS from the TSF using the identified VM image. Only Target System Factories announcing the capability to deploy VM images can be used for this. Mixed operation of different configurations of UNICORE/X is possible within an

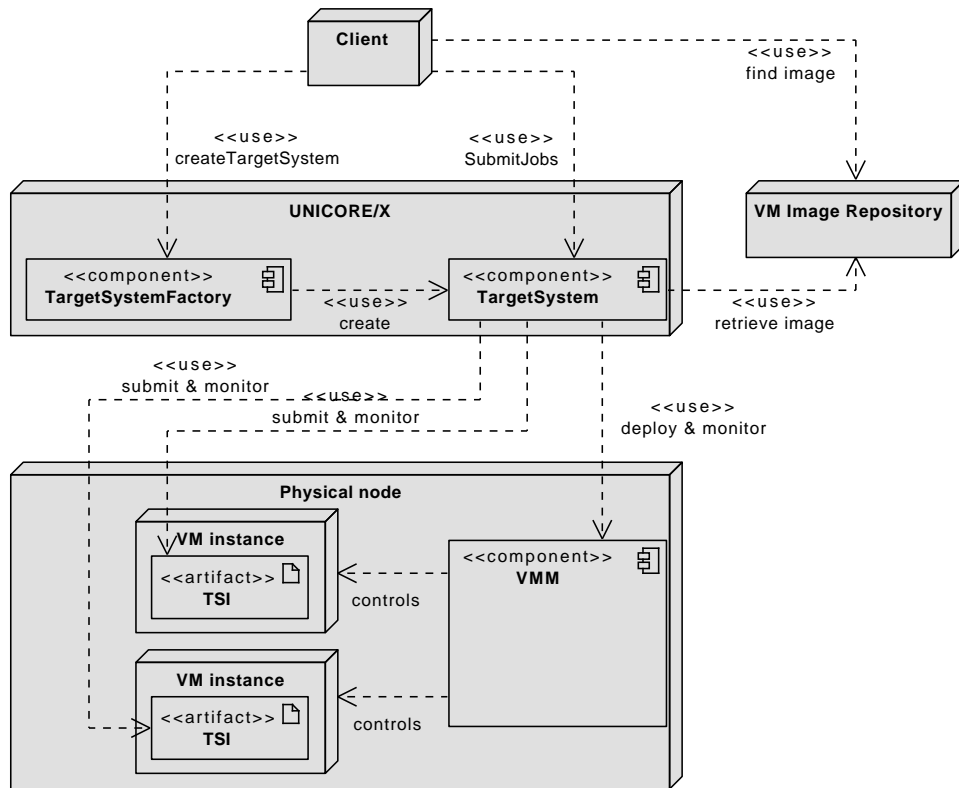


Figure 1. UNICORE Architecture supporting user defined runtime environments.

infrastructure, such that traditional and virtualised UNICORE services can be used concurrently.

Once the request to deploy a VM image has been sent to the TSF, it will immediately create a WS-Resource representing the new TSS. The TSS implementation takes care of the deployment of the VM image and the tracking of its state until the TSI inside the VM instance becomes available. From this point onward, everything behaves as in the existing UNICORE implementation of the TSS. Thus, job submission and monitoring, as well as storage actions, are done without requiring any additional functionality, i. e. clients can be used unchanged.

3.1 TargetSystemFactory

As already stated, in traditional UNICORE the implementation of a Target System Factory (TSF) is only capable of creating TSSs that all represent the same computing resource. While this suits the model of a single High Performance Computing (HPC) resource perfectly, it does not serve all users' needs when it comes to providing user or application specific environments.

In such cases, the TSF should rather represent a computing centre where user defined



Figure 2. Comparison of the existing (left) and proposed (right) UNICORE architectures. In the new model, each TSS represents its own XNJS.

environments can be deployed, leading to differing TSS created by the factory according to the characteristics of the environment, i. e. the VM image.

3.2 TargetSystem

The Target System Service (TSS) is the actual representation of a computational resource. It is this service, to which the user submits jobs for execution. The TSS service is a WSRF¹¹ service meaning that it is stateful and the state is associated with resource ids that a caller needs to provide in order to retrieve the correct state. Each TSS is represented by its own resource id.

Currently, each of the TSS resources represents the same computational resource. With the model described in this paper, this will change and a TSS will represent an instantiated virtual machine with its unique characteristics. The model is extensible to also cater for entire clusters of virtual machines, as this is analogous to physical clusters already represented in the current UNICORE model.

For the new model, the TSS needs to be extended in various ways. First of all, virtualised nodes require a certain amount of time for their setup. On the other hand, the Web Services (WS) call from the client to the TSF should return as quickly as possible. So far, it has been the case that once the WS call returned, the TSS could be used right away. In order to inform the client about whether the TSS is ready to accept job submissions, an additional state property has been added to its properties. Only once the TSS goes into state *Ready* will the client be able to submit jobs to it.

3.3 XNJS

The TSS is a facade to the Extended Network Job Supervisor (XNJS), the execution engine of UNICORE. With all TSS instances representing the same computational resource, only a singleton XNJS is required inside a UNICORE/X server. This is different in the proposed model, and thus an individual instance of the XNJS with differing configuration in each case is required in order to access each of the VMs.

Whereas this type of setup had been foreseen in earlier versions of UNICORE, it has never been used so far. Different TSS offer different applications (user defined environments), and thus the configuration of the Incarnation Database (IDB) needs to be different, too. The existence of a 1-1 relationship among XNJS and IDB as well as among XNJS and TSI is another reason to use different XNJS instances for different machines. This is shown in Figure 2.

Once the implementation of the proposed architecture has sufficiently advanced, scalability issues will have to be evaluated. After all, the XNJS is an active component with a

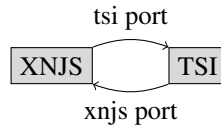


Figure 3. Connections between XNJS an TSI.

number of parallel threads. Multiple XNJS instances could easily lead to a stall of the UNICORE/X machine and thus reasonable limits will have to be determined to avoid this. These limits will be put on the number of threads available within each XNJS and the number of XNJS instances and thus the number of VM a single UNICORE/X server can handle.

3.4 TSI

The Target System Interface (TSI) remains as is without further changes. However, there are a number of configuration actions necessary to link the TSI as deployed in the virtual instance, with the corresponding XNJS. The TSI protocol is defined such that there are two connections required between the XNJS and the TSI. One connection from the XNJS to the TSI and another connection from the TSI back to the XNJS. The ports for these connections are usually configured in a static manner, such that they need to be known prior to starting any of the two components. Figure 3 illustrates this.

It would be possible to restrict the TSI port to which the XNJS connects on the virtual instance to just a single port. After all, each TSI binds to this port on a different (virtual) machine, such that there are no clashes.

This approach does not satisfy the requirements of the XNJS port, to which the TSI connects. The XNJS is structured such that an incoming connection on a particular port is associated with a particular TSI. Multiple TSIs connecting to the same port will not be possible. Thus, the XNJS port for the TSI connection will need to be dynamic and will need to be communicated to the TSI before starting it.

This customisation can be done if we are able to contextualise¹² the running instance, e. g. at boot time. The information about which XNJS the TSI is supposed to connect to, will be passed into the TSI configuration during the startup process of the virtual instance. Many cloud middlewares support this contextualisation process by different means^{13,14}. This mechanism can be crafted manually when using only vitalisation, as proposed in this paper.

3.5 Matching of Requirements

The full set of properties of a target system in the new model is made up of properties from various, interdependent sources. This is shown in Figure 4. The properties are taken from the JSDL specification's Resources elements¹⁵. The experienced reader will notice that cluster properties, i. e. all *Total** resources, have been omitted. This is due to the fact that the currently targeted implementation will only support the instantiation of single virtual

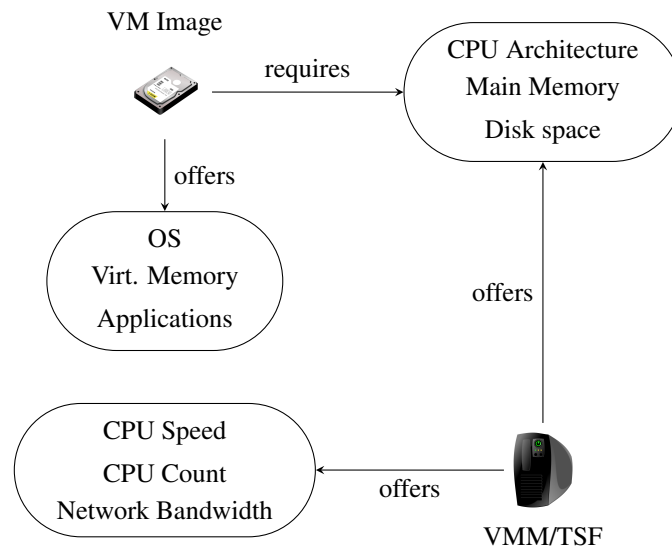


Figure 4. The full set of properties of the TSS is made up of properties from various, interdependent sources.

nodes. Hence, cluster properties would have no meaning. Once the general model of virtualised TSS has been implemented it will be extended to support entire virtual clusters (cf. section 6.2).

The properties can be regarded in the following manner. The user is interested in running a particular *Application* or *OS*. These are offered by a VM image. Thus, the VM image a user is interested in belongs to a subset of all images, where the images in this subset offer what he desires.

Each of these images have certain requirements to be met by the virtualised environment in order to run the image. Virtualised environments represented by a TSF have to offer these properties. This second step of matching the image requirements with the TSFs offerings further restricts the set of possible images to use. The final selection of an actual virtualised environment could be done based on the further offerings of the TSF, e.g. *CPU Speed* or *CPU Count*.

4 Implementation

The architecture described above is currently implemented as part of a master thesis due in Fall of 2011. In the following, we would like to point out some implementation details that are less related to the overall architecture.

4.1 Abstraction of the VMM

With the multitude of Virtual Machine Managers (VMM) available today, every computing centre uses its own approach and VMM. In order to avoid a vendor lock-in depending on

a single one of the solutions, we will use an abstraction layer enabling us to run the virtual instances on a variety of VMMs. The library we'll use is called libVirt¹⁶. libVirt is capable to access a variety of contemporary VMMs. Additionally, it provides (remote) management of virtual machines, virtual networks and storage. The client API is available for LINUX, Solaris and Windows platforms. Furthermore, it supports the Java programming language among others, which is important for an easy integration with the TSS.

An alternative approach would be the use of an abstraction layer over Cloud middlewares, which is available as jclouds, presenting "cloud-agnostic abstractions, with stable implementations of ComputeService and BlobStore."¹⁷ We foresee an update of our work to be using cloud infrastructures in the future. For the time being, we restrict ourselves to the local and direct use of virtualisation, expecting no extensive architectural changes to cater for the use of cloud environments.

4.2 VM Image Repository

The VM image repository can be seen as a storage with additional properties on files. These properties are metadata describing the contents of files for easy retrieval and selection of appropriate files. The functionality is already available with the UNICORE Metadata Management Framework (MMF)¹⁸.

We have defined a profile of using the MMF to describe the contents of VM images. The meta data about the image serves several purposes.

First of all, it is used to discover VM images containing certain applications or setups, i. e. user defined environments. For this purpose, an Incarnation Database (IDB) document is part of the meta data, capturing what the image provides.

Secondly, each VM image has a set of requirements associated with it in order to be executable in a certain environment. These requirements are also captured as part of the metadata.

Lastly, each VM image contains a TSI, which the XNJS needs to connect to. As the TSI port can differ among VM images, the port number is described as part of the meta data. It is expected that more configuration data will be added to this section of the image description as the development continues.

5 XaaS

"Cloud computing is emerging as a model in support of 'everything-as-a-service' (XaaS)."¹⁹ We can categorise the work of this paper within the landscape of these views of services.

First of all, the TSF is located at the Infrastructure-as-a-Service (IaaS) layer. It provides access to actual hardware available in a computing centre. Users are enabled to run specifically crafted virtual machine images on this infrastructure.

The virtual machine images can then be seen as platforms providing an operating system, runtime environment, and possibly applications to be run in this environment. They are thus located in the Platform-as-a-Service (PaaS) layer.

Lastly, Software-as-a-Service (SaaS) can be offered in dynamic ways. The discovery of virtual machine images providing an application and the discovery of sites running these images can be automated. Thus, a user would only have to submit a request for a particular

application to be run on specific data and not have to deal with the setup of the runtime environment.

The different service models are stacked on top of each other. SaaS is stacked on top of PaaS, and PaaS on top of IaaS. As stated in², the stacking is not strict such that higher levels can access all services of any lower level.

6 Summary and Future Work

In this paper, we have described a refinement of the UNICORE architecture to allow for the provision of virtualised environments, in which VM images accessible through UNICORE clients can run. This model enables users to provide their own, domain specific VM images being able to run users' jobs. At the same time, it removes some of the burden of software administration from computing centres' staff, as only virtualisation software needs to be administered. Users benefit from this solution by being able to run the software of their choice administered by software experts from their problem domain.

The new approach matches the existing UNICORE model quite well, most of the changes relate to small interface changes in the TSF and TSS. The setup of the VM instances by the TSS will be the biggest development effort of this endeavour.

The solution is part of the way to UNICORE offering Software-as-a-Service by means of a higher-level service responsible for the orchestration of the chosen software, a VM image offering this software and a virtualised environment where this image can be run.

6.1 Extension to Cloud Computing

We have so far restricted our model to only be used in local virtualised environments. However, not much is keeping us from running the VM images on Cloud resources backed by a cloud middleware. Then, the borders between local execution, e. g. in a private cloud, and remote execution, e. g. in a public cloud, would vanish. In turn, users could use whatever amount of computational resources whenever they like and still submit their jobs through one of the UNICORE clients.

On the other hand, the use of public clouds is subject to payments. So the use of public clouds will probably have to be made explicit to users, if they are to come up for the costs of running their VM instances and jobs in the public cloud. This would require another, client side, extension to the proposed model, letting the user decide about whether they are willing to pay the price for quick execution in public clouds. A selection of which public cloud to used could also be part of this extension.

6.2 Cluster VMs

The currently ongoing implementation intends to provide access to single VM nodes. In terms of High Performance Computing (HPC), one of the core domains of the UNICORE Grid middleware, users are interested in entire clusters of compute nodes. Future work will have to investigate the possibility of easily providing virtualised clusters. General work in this direction showed promising results²⁰. Some of the open source Cloud middleware already support this schema. The solution proposed in this paper just needs to use the functionality provided, as the model is already capable of running computational jobs on a cluster, no matter if physical or virtual.

References

1. Ian Foster, Carl Kesselman, and Steven Tuecke, *The anatomy of the Grid: Enabling scalable virtual organization*; The International Journal of High Performance Computing Applications, 15(3):200 - 222, Fall 2001.
2. Christian Baun, Marcel Kunze, Jens Nimis, Stefan Tai, Christian Baun, Marcel Kunze, Jens Nimis, and Stefan Tai, *Cloud-architektur*; In Cloud Computing, Informatik im Fokus, pages 27 - 41. Springer Berlin Heidelberg, 2011. 10.1007/978-3-642-18436-9_3 (in German).
3. Sai Srinivas Dharanikota and Ralf Ratering, *Manageable dynamic execution environments on the grid using virtual machines*; In Roman Wyrzykowski, Jack Dongarra, Norbert Meyer, and JerzyWasniewski, editors, Parallel Processing and Applied Mathematics, volume 3911 of Lecture Notes in Computer Science, pages 643 - 650. Springer Berlin / Heidelberg, 2006.
4. Xen. URL: <http://www.xen.org>. Last accessed: 2011-06-10.
5. glite-wn. URL: <http://glite.cern.ch/glite-WN/>. Last accessed: 2011-07-22.
6. gLite - Lightweight Middleware for Grid Computing. URL: <http://glite.cern.ch/>. Last accessed: 2011-07-22.
7. WNoDeS. URL: <http://web.infn.it/wnodes/index.php/wnodes>. Last accessed: 2011-06-10.
8. The globus alliance. URL: <http://www.globus.org/>. Last accessed: 2011-08-01.
9. Globus virtual workspaces. URL: http://dev.globus.org/wiki/Incubator/Virtual_Workspaces. Last accessed: 2011-08-01.
10. Nimbus. URL: <http://www.nimbusproject.org>. Last accessed: 2011-08-01.
11. Steve Graham, Anish Karmarkar, Jeff Mischkinsky, Ian Robinson, and Igor Sedukhin. *Web services resource 1.2 (ws-resource)*. URL: http://docs.oasis-open.org/wsrif/wsrif-ws_resource-1.2-spec-os.pdf, April 2006. Last accessed: 2011-07-22.
12. Katarzyna Keahey and Tim Freeman. *Contextualization: Providing one-click virtual clusters*. In eScience, 2008. eScience '08. IEEE Fourth International Conference on, pages 301 - 308, December 2008.
13. Contextualizing Virtual Machines 2.0. URL: <http://opennebula.org/documentation:archives:rel2.0:cong>. Last accessed: 2011-08-09.
14. Nimbus cluster guide. URL: <http://www.nimbusproject.org/docs/current/clouds/clusters2.html>. Last accessed: 2011-08-09.
15. Ali Anjomshoaa, Fred Brisard, Michel Drescher, Donal Fellows, An Ly, Stephen McGough, Darren Pulsipher, and Adreas Savva. *Job submission description language (jsdl) specification, version 1.0*. URL: <http://www.gridforum.org/documents/GFD.56.pdf>, November 2005. Last accessed: 2011-08-11.
16. libvirt: The virtualization API. URL: <http://libvirt.org/>. Last accessed: 2011-06-09.
17. jclouds - multi-cloud library. URL: <http://code.google.com/p/jclouds/>. Last accessed: 2011-08-01.

18. Waqas Noor and Bernd Schuller. *MMF: A Flexible Framework for Metadata Management in UNICORE*. In Achim Streit, Mathilde Romberg, and Daniel Mallmann, editors, UNICORE Summit 2010, Proceedings, volume 5 of IAS Series, pages 51 - 60. Forschungszentrum Jülich, 2010.
19. Alexander Lenk, Markus Klems, Jens Nimis, Stefan Tai, and Thomas Sandholm. *What's inside the cloud? an architectural map of the cloud landscape*. In Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing, CLOUD 2009, pages 23 - 31, Washington, DC, USA, 2009. IEEE Computer Society.
20. Single click starts a 10,000-core cyclecloud cluster for \$1060/hr.
URL: <http://blog.cyclecomputing.com/2011/04/single-click-starts-a-10000-core-cyclecloud-cluster-for-1060-hr.html>, April 2011. Last accessed: 2011-07-22.

Desktop Grids Opening up to UNICORE

Matthias Keller¹, Jozsef Kovacs², and André Brinkmann¹

¹ Paderborn Center for Parallel Computing,
Fürstenallee 11, 33102 Paderborn, Germany,
E-mail: mkeller@upb.de

² MTA SZTAKI Computer and Automation Research Institute Hungarian Academy of Sciences
P.O.Box 63, 1528 Budapest, Hungary
E-mail: smith@sztaki.hu

Research communities from high energy physics to humanities utilised grid infrastructures to support and accelerate their research. Their computations can be executed by different grid technologies: Grids of cluster systems, like the German D-Grid, grids of supercomputers, like the Distributed European Infrastructure for Supercomputing Applications (DEISA), or desktop grids consolidated in the International Desktop Grid Federation (IDGF). UNICORE is one of the three grid middleware environments supported by the European Middleware Initiative (EMI) for managing a set of cluster or supercomputers, but desktop grids are currently unsupported. This work fills this gap enabling UNICORE to support all three kinds of grid technologies of the European Grid Infrastructure (EGI). This unified interface enables European scientists to access web services, portals, and applications on all grid technologies in the same way.

1 Introduction

The European Grid Infrastructure (EGI) creates and provides an eScience infrastructure for European research communities. Those communities, reaching from high energy physics to humanities, can choose from different grid technologies to execute their computations: Grids of cluster computers, like the German D-Grid, supercomputers, like the Distributed European Infrastructure for Supercomputing Applications (DEISA), or desktop grids consolidated in the International Desktop Grid Federation (IDGF). Each of the technologies has different properties concerning access limitations, amount and types of available resources, and privacy properties. UNICORE¹ supports both clusters and supercomputers, which are also known as service grids (SGs). Extending UNICORE to support desktop grids (DG) is the final step to enable UNICORE to operate with all available grid resources. This extension enables European eScience communities, especially UNICORE VOs, to access all available grid resources from one middleware.

Desktop grids collect idle resources from loosely coupled desktop computers. Each DG client only has to install a small client software to run computations. DGs exploit that compute capacities of modern computers are rarely exhausted while performing daily tasks, like reading mails and websites or writing texts. The DG client software now allows the utilisation of idle capacities for scientific computations. The desktop computers form either an institutional/private/campus or a voluntary/public desktop grid. The first scenario aggregates idle resources out of computer pools of an university or out of employees' computers of a company. The second scenario motivates the general public to install DG client software on their home computers and therewith to donate compute resources for science similar to popular projects like SETI@home².

Since there is no high-speed network between the desktop computers, only a subset of applications is applicable to them, like parameter studies (PS). Applications like MPI parallel applications, which require such a high-speed interconnect should still be executed on service grids. However, PS currently running on service grids can be outsourced to DGs, leaving more resources available for researchers in need of highly interconnected nodes. The power of the DG is the potential to collect and utilise millions of desktop computers owned by citizens. A typical public desktop grid can collect hundreds of thousands or even millions of computers from the volunteers worldwide, depending on how appealing the supported applications are for the citizens. In case of a campus desktop grid, the typical size is several thousands since they are collected within an institute/company.

The aim of the European Desktop Grid Initiative (EDGI) is to deploy desktop grid (DG) and cloud services for EGI user communities that are heavy users of distributed computing infrastructures (DCIs) and require an extremely large multi-national e-infrastructure. In order to achieve this goal, software components of ARC, gLite, UNICORE, BOINC, XWHEP, ATTICS, 3G Bridge, OpenNebula, Eucalyptus and OpenStack will be integrated into a platform, which can move jobs from service grids to desktop grids. Therefore, EDGI extends ARC, gLite, and UNICORE grids with gateways to volunteer and institutional DG systems. EDGI also integrates a bridge between service grids and cloud environments to be able to support QoS for the DG environments. This enables EDGI to explore new service provision models in order to ensure a harmonised transition from service grids to desktop grids. EDGI will also provide a workflow-oriented science gateway to enable user communities to access the EDGI infrastructure more easily. EDGI establishes the IDGF Organization to coordinate DG-related activities in Europe, both for solving technical issues as well as to attract volunteer DG resource donors by disseminating results of the EDGI and EGI projects. IDGF and EDGI will work in strong collaboration with EGI, EMI, NorduGrid, the UNICORE Forum, and interested NGIs.

The rest of the paper is organised as follows. Section 2 gives a short overview of the EDGI infrastructure. Section 3 describes the technical challenges of extending UNICORE as a UNiform Interface to Computing REsources to support multiple DGs. Finally, section 4 concludes the work described in this paper.

2 Overview of the EDGI Infrastructure

EDGI is a project supported by the FP7 Capacities Programme. This section will give a short overview of the key components of the EDGI infrastructure³, which is being developed and maintained during the project. The aim of this infrastructure (shown in Figure 1) is to provide a gateway to tens of thousands of desktop grid resources for those users who are familiar with gLite, ARC, or UNICORE type service grids and do not intend to move their environment to another type of grid. Adding a huge number of desktop grid resources can significantly increase the overall capacity of a service grid system.

Desktop grid systems can be built based on institutional or volunteer resources. In the former case, an institution (e.g., a university) is maintaining the resources, while in the latter case, the resources are provided by volunteer home PC owners worldwide. In DG systems, applications must be ported (i.e., modified according to the requirements of a DG software like BOINC) and validated (to make sure they do not cause any harm on the resource) before being deployed on the server. After the deployment, which includes reg-

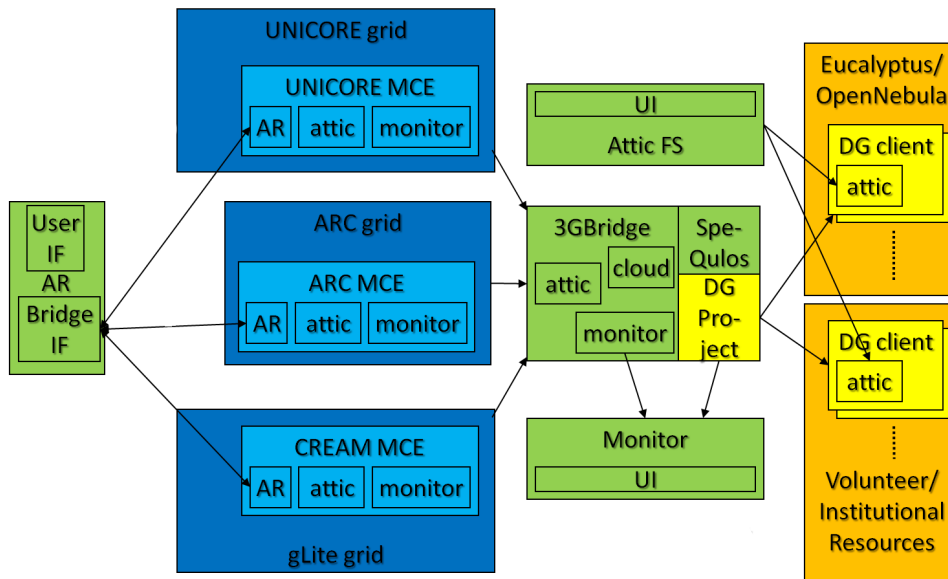


Figure 1. The EDGI software infrastructure.

istering the application with all of its binaries, workunits can be submitted. In DG systems, volunteers trust the applications provided by the attached DG when letting the application run on their PCs. To store the validated applications centrally, a new *Application Repository* (denoted by “AR” in Figure 1) has been introduced. The most important information a user can find related to an application are a detailed description, binaries, example input files, supported service grids and desktop grids. Users can select an application and submit it to his/her well-known service grid as usual.

The specially prepared *Modified Computing Element* (MCE inside gLite, ARC, or UNICORE in Figure 1) is responsible for forwarding the job to a desktop grid. Before doing that, it checks whether the application originates from the AR (through the “Bridge IF”) and then whether the application is supported (i.e., registered) on the target desktop grid. Access to a desktop grid is provided by the *Generic Grid-Grid bridge* (3G Bridge)⁴ component (depicted in the middle of Figure 1) running on the server of the target desktop grid system. The 3G Bridge is responsible for inserting the job as a workunit into the desktop grid system (BOINC or XtremWeb), for keeping track of its progress, and for reporting the status and results back to the service grid computing element.

The EDGI infrastructure is also able to utilise computing resources (shown in the upper-right corner of Figure 1) from cloud infrastructures by instantiating virtual machines on OpenStack or on OpenNebula to speed up computation when actual resources are considered unable to provide enough capacity. The dynamic scheduling of cloud resources is part of the newly developed SpeQuloS⁵ software, which resides on the DG server. There is also an Attic P2P file system⁶ used for storing huge input files for jobs executed many times to decrease the load on the desktop grid server. Monitoring⁷ in EDGI collects information about service grid and desktop grid components and provides a web-based portal to overview the usage/utilisation of the overall infrastructure.

3 Implementation

This section describes the extension of UNICORE to access desktop grid resources. After starting with a brief UNICORE description, the implementation is described and critical technical details are outlined.

3.1 UNICORE Ecosystem

The following section describes the distributed architecture of UNICORE. UNICORE is divided into three layers: client, service, and system. Users and their applications can access UNICORE services via different client tools. The UNICORE services, associated with the service layer, are loosely coupled Java Web services. They have to be registered in at least one well-known *Service Registry*. The concrete resources, clusters, or storage systems are UNICORE's *target systems* and are part of the system layer. The services on top of these target systems provide a system independent UNiform Interface to (COmputing) REsources (UNICORE). The remainder of this section will describe the different services as shown in Figure 2.

UNICORE/X is the core component of a UNICORE site. It hosts the web services for job and storage management:

- The *Target System Service (TSS)* is created by a factory (TSF) and provides access to native resources. It uses the back end component, the eXtended Network Job Supervisor (XNJS), sometimes referred to as the job management and execution engine of UNICORE. For job submission a *Job Management Service* is created and provided.
- The *Job Management Service (JMS)* handles the control flow of a single job: submit, abort, pause, and monitor. To execute these requests the XNJS is used.
- The *Storage Management Service (SMS)* offers unified access to arbitrary storage systems. The main functionality is implemented inside the back end XNJS component.
- The *File Transfer Service (FTS)* enables concurrent file transfer operations between generic SMSs or by supporting protocols like HTTP, OGSA Byte-IO, GridFTP, UDP, and UFTP.

These services are known as UNICORE atomic services (UAS) and are indirectly accessible through a *Gateway* that ensures security. Information about available services are stored in a special *Registry Service*.

The *Gateway* component is the single entry point to a UNICORE site. It authenticates all incoming requests and forwards them to their intended destination services. Replies are sent back to the clients. As a result, only one port is needed to access a site. To authenticate users the Gateway checks the signed user certificates. These are stored in the XUADB together with a set of attributes, which are used by UNICORE/X to authorise users.

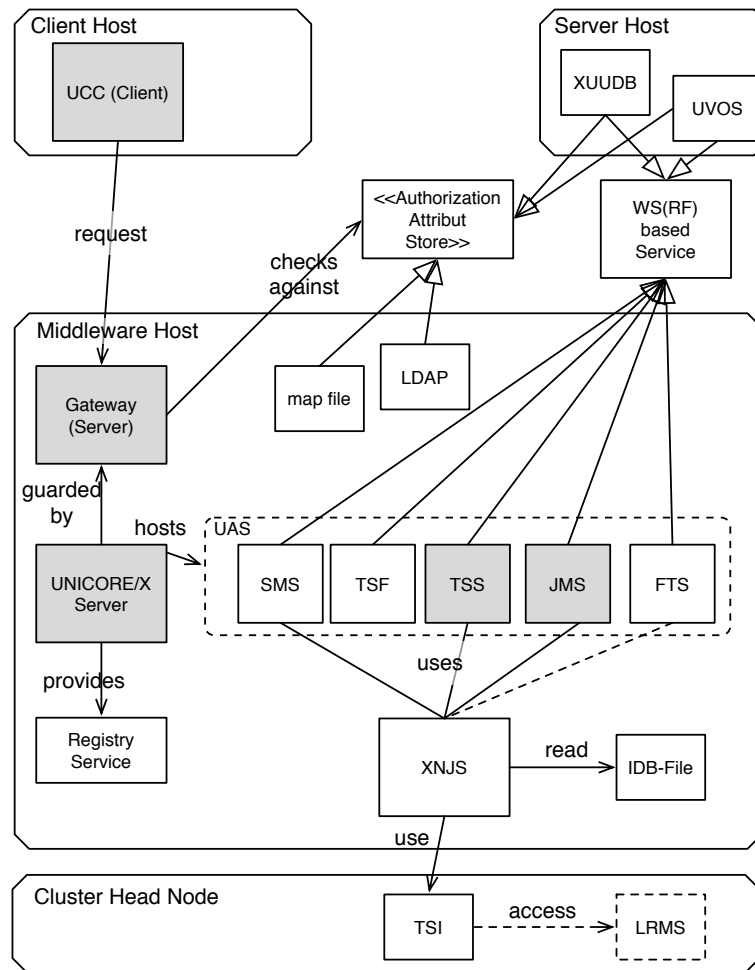


Figure 2. Modules and components of UNICORE.

The *XUADB* is a user database mapping user credentials (certificates, distinguished names) to access privileges for each service and to a set of attributes, like Unix logins, roles, or projects.

The *Registry Service* provides a central point of registration for different services of a UNICORE installation. At least one registry is needed to run an installation but different can be provided, e.g., each site can have a local registry to separately list local services.

The *eXtended Network Job Supervisor (XNJS)*, also called UNICORE engine, contains most of the core functionality of a UNICORE site, which is used by most UASs. An internal component is the *Target System Interface (TSI)*, which enables a unified access to target systems for resource and storage handling. The XNJS handles the control flow

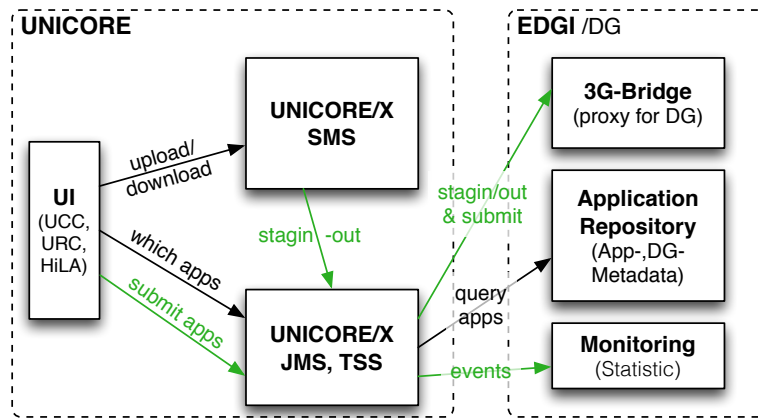


Figure 3. Architecture and Interaction of UNICORE and EDGI components.

of a job, the staging to and from local storage systems and submitting to local resource management systems. This is done through the TSI, which implements the access to native management and storage systems.

The Target System Interface (TSI) implements access to local systems. In a typical grid environment, a resource management software (RMS) like the Sun Grid Engine schedules a job and a shared file system provides access to the working directory for the job.

Currently, two types of TSI implementations exist: a Java and a Perl implementation. The Java implementation is mainly used for test purposes. It executes a shell process and stores data locally. The Perl implementation runs a set of Perl scripts with different implementations for accessing Torque, SGE, OpenCCS, or PBS.

3.2 Architecture and Implementation

This section describes the changes to services described in the previous section required to enable DG job submission. Figure 3 shows UNICORE and EDGI components and their interactions. With the User Interface (UI), a user can upload his input data to an SMS, query a UNICORE/X server for available applications, and submit jobs using those applications and input data. The UNICORE/X server (specifically the XNJS through the TSI) will download the input data from the SMS and submit the job to the 3G Bridge, which forwards the job to the target DG. After the job terminates on a DG client, the UNICORE/X server downloads the output data from the DG and stores it on the given target storage. Later, the user can download the results from the target UNICORE storage.

Typically, a UNICORE/X site represents a single cluster. The extension keeps this architecture by wrapping a single DG installation. Therewith, querying each UNICORE/X for application or resource information results in individual data for each DG. Additionally to submit a job, a user can choose a DG installation. Wrapping a target system is usually done by implementing a TSI. Rather than using a Perl script and executing command line applications, we decided to implement a Java based TSI, which is less error prone. This

also includes a full re-implementation of the TSI for accessing remote components like the application repository and the 3G Bridge, which is SOAP-based.

The remainder of this section describes critical points of the implementation in detail: the special data flow, processing information from the application repository, communication with external components, URL-passthrough realisation, and installation.

The staging process is slightly different from a standard UNICORE system, because data has to be transferred additionally to and from the DG head node. The control flow during a job submission is shown in Figure 4. Figure 5 shows related interactions from an architectural point of view. The process is initialised by a user request for job submission (a). Through the TSI, the back end component XNJS initiates a stage-in process (b,c). This creates a working directory for the job and downloads every input file into this directory (c). In contrast to the stage-out process, this stage-in process is unchanged and works by using existing code supporting every UNICORE file transfer protocol.

Afterwards, the XNJS triggers a job submission (d). The new TSI transforms the UNICORE job request into a 3G Bridge request (e). This request cannot contain input data, which is probably hundreds of megabytes large. Thus, the 3G Bridge expects an alternative way to fetch the input data, e.g., via HTTP (f). The 3G Bridge is typically collocated with the DG management software on the DG head node, so the 3G Bridge moves these files to the correct place. To enable fetching input files, the filespace-directory, in which UNICORE creates working directories, is exposed via a HTTP-server. To inform the 3G Bridge about the download location, the input file location of the request (e) is replaced by the corresponding remote URL.

If nothing unexpected happens, the status will change from PENDING to RUNNING and FINISHED. The new TSI observes this by polling the 3G Bridge (h,i). If FINISHED is reached, the 3G Bridge provides a list of output data with HTTP URLs. This data is downloaded to the job's working directory (j). This functionality is hooked in prior to UNICORE's normal stage-out process to enable a fully supported file transfer. Thus, the XNJS believes the computation is not finished until the output file downloading from the remote HTTP server (j) is finished. After every file transfer is finished (j), the job and its files are deleted on the DG site (k), and the normal XNJS functionality continues: stage-out after job computation (l, m).

This implementation preserves UNICORE/X stage functionality and supports different target and source storage systems. It also cleans the 3G Bridge or the DG head node up. Thus overall storage capacity is minimised. The job data is stored (for later download) in only one place, at the UNICORE side. To download data from the 3G Bridge site, the implementation reuses UNICORE's File Transfer Services to have stable and concurrent download capabilities.

The Incarnation Database (IDB) is basically an xml-file read by the XNJS. The IDB is used to describe native or local properties of the target system, for which the UASs provide a uniform interface. The IDB has roughly three parts describing the kind of local resources, CPU architecture, cluster size, the available applications, and specific execution environments used by applications, e.g., MPI. Based on a JSDL-extension, the IDB can specify how to invoke POSIX-like applications. A viable solution is to introduce a new JSDL-extension for DG-like applications and fill them with data and properties from the application repository. This solution is postponed after the first version is running stable

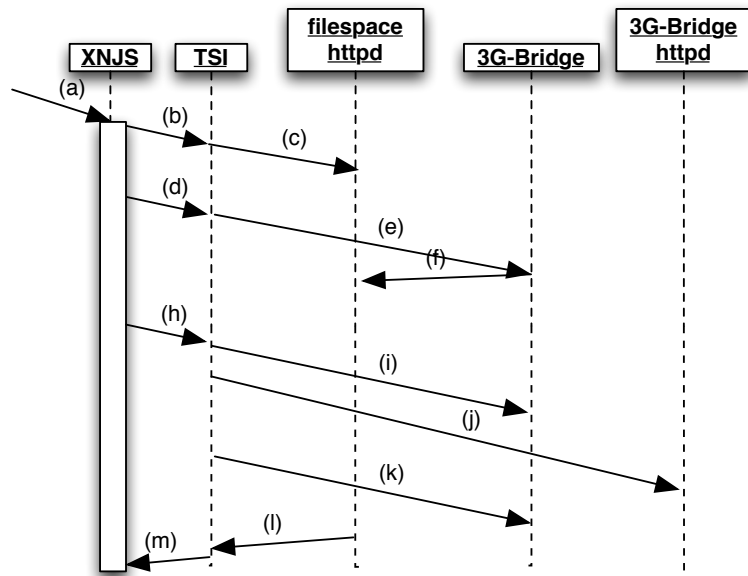


Figure 4. Job control flow between UNICORE and EDGI components.

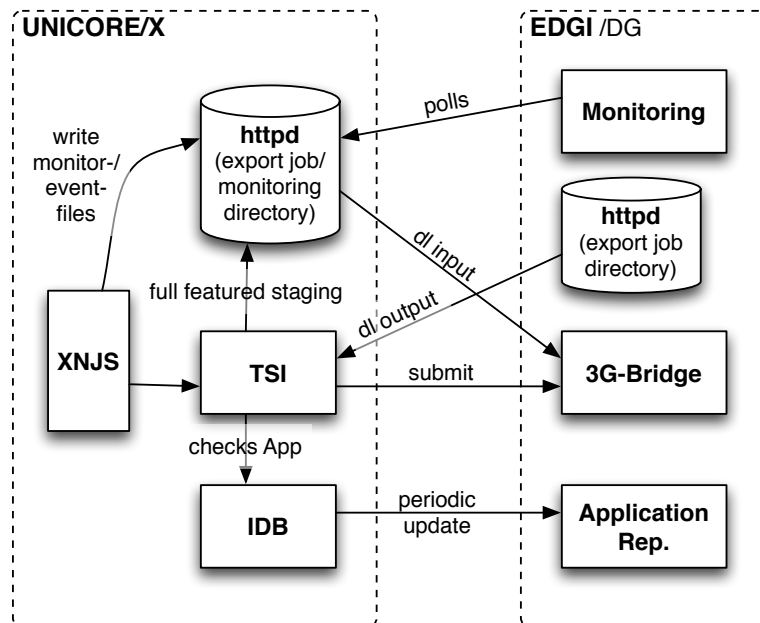


Figure 5. Interaction of components during job submission.

because it introduces additional efforts of creating the specification and verifying it. The current solution updates the IDB at UNICORE/X start time with information from the application repository, which relates to the currently accessed DG. This can also be triggered from command line, deactivated in the XNJS configuration file, and continuously updated if an interval is specified (default are daily updates). This eases the administrative effort because newly available applications for DGs are automatically available for UNICORE users without involving manual actions. This manual step is still needed for submission via gLite and ARC.

An additional benefit of utilising the IDB system is that information about the accessed DG instance are available from inside the UNICORE system. This not only enables the UNICORE users to access the new type of resources as usual but also frees them to look up DG application specific information from the application repository (another website) and to write a correct job description for submission.

Communication with external components is wrapped inside border classes. The communication is written in pure java so that no command line calls for clients are involved. This makes the implementation more stable and reliable and exceptions can be handled better. For communication with the 3G Bridge, UNICORE's SOAP communication capabilities are reused and the Application Repository is accessed via HTTP requests. A third border class handles the EDGI internal monitoring system, but does not need to communicate with external components, because the monitoring system follows a pull model for observed nodes. For this, a directory is exposed by an HTTP server and the border class writes monitoring events in files, handling the special format for the monitoring internally.

The benefit of encapsulating the handling of external components is that changes to the interface are most probably restricted to local changes within these border classes. Thus the remaining code stays unchanged. This was observed by updating the monitoring format and SOAP-interface for a new 3G Bridge version.

URL-passthrough is a concept realised in EDGI for gLite to bypass local staging processes. Publicly available data can directly be downloaded by DG clients. Otherwise, those data have to be downloaded to a UNICORE/X side, then to a DG head node, and from there DG clients could fetch them. This would result in a waste of bandwidth and potentially cause network bottlenecks on the UNICORE/X site or the DG head node. For bypassing, the URL has to contain certain data and stick to the following format: `bypass://[protocoll]://[host,path to file]:[md5]:[size]`.

Installing a UNICORE site with this extension involves the following two steps. First, a default UNICORE/X server has to be installed. For this more difficult part, manuals are provided by the UNICORE team^a. Additionally, an optional java-based graphical installer is available^b. Afterwards, the UNICORE extension for desktop grids has to be installed as a plugin. Therefore, you need to place a jar-file in a directory and reconfigure the UNICORE/X configuration file. Both, the jar-file and the detailed install instructions are provided in a separate directory as a package for each version at the publicly accessible UNICORE repository^c.

^a<http://www.unicore.eu/documentation/manuals/unicore6/>

^b<http://www.unicore.eu/download/unicore6/>

^c<http://unicore.svn.sourceforge.net/viewvc/unicore/unicorex/trunk/>

4 Conclusion

We have described the motivation for and the implementation of a UNICORE extension, which directly supports desktop grids from within this service grid middleware. The extension will have a release version, which will be integrated in a productive environment. The stability of the extension is ensured by several unit tests and also by EDGI's work package to test the developed infrastructure. Thus, an active cycle between test engineers and developers is established.

Additional concurrent project efforts port a lot of applications to run on desktop grids, so that the UNICORE community can choose from a rich set of applications to submit to new resources. These new resources will boost research, not only for those who use the additional desktop grid resources, but also by leaving more capacity for the rest of the users. In summary, the release quality, the broad application spectrum, and newly available resources make this extension a valuable contribution to the UNICORE community.

Acknowledgements

The research leading to these results is funded by the European Union Seventh Framework Program (FP7/2007-2013) under grant agreement no 261556.

References

1. A. Streit et.al. *Unicore 6 - recent and future advancements*. Berichte des Forschungszentrums Jülich, JUEL-4319, Forschungszentrum Jülich Zentralbibliothek, February 2010.
2. D.P. Anderson. *Boinc: A system for public resource computing and storage*. Proceedings of the 5th IEEE/ACM International GRID Workshop, pages 1 - 7, 2004.
3. P. Kacsuk, J. Kovacs, Z. Farkas, A. Marosi, and Z. Balaton. *Towards a powerful european dci based on desktop grids*. Journal of Grid Computing, 9:219 - 239, 2011. 10.1007/s10723-011-9186-z.
4. Z. Farkas, P. Kacsuk, Z. Balaton, and G. Gombás. *Interoperability of boinc and egee*. Future Generation Computer Systems, 26(8):1092 - 1103, 2010.
5. S. Delamare, G. Fedak, D. Kondo, and O. Lodygensky. *Hybrid distributed computing infrastructure experiments in grid5000: Supporting qos in desktop grids with cloud resources*. In Proc. of Grid500 Spring School, 2011.
6. I. Kelley and I. Taylor. *Bridging the data management gap between service and desktop grids*. Data Management, pages 13 - 26, 2008.
7. F. Araujo, D. Santiago, D. Ferreira, J. Farinha, L. Moura Silva, P. Domingues, E. Urbah, O. Lodygensky, A. Marosi, G. Gombas, Z. Balaton, Z. Farkas, and P. Kacsuk. *Monitoring the edges project infrastructure*. In 3rd Workshop on Desktop Grids and Volunteer Computing Systems (PCGrid 2009), Rome, Italy, May 2009.

NHiLA - Bridging the Gap Between .NET and UNICORE

Daniel Krott¹, Michael Gerhards¹, Sascha Skorupa¹, and Volker Sander¹

FH Aachen, University of Applied Sciences,
52428 Jülich, Germany

E-mail: {D.Krott, M.Gerhards, Skorupa, V.Sander}@fh-aachen.de

A main topic of the HiX4AGWS project is the distribution of Grid jobs to the UNICORE middleware. A new way of job distribution has been evolved there in addition to the existing push-based approach of the UNICORE developers. This approach is called actor-driven pull-based, because the actors (e.g. Grid resources) are not passive any more. The actors pull the jobs from the task repository for processing. Currently there are clients written in Java for accessing the UNICORE middleware, so the actors can only be written in Java, too. There is no support for .NET written actors. The UNICORE Grid middleware is accessible by a service layer using web service based interfaces, which are either provided according to the OGSA standard or by the proprietary UNICORE Atomic Services (UAS). These web service interfaces are used by the components of the client layer of the UNICORE architecture that enable users to access Grid resources, e.g. by submitting Grid jobs or by starting Grid file transfers. Currently, the client layer consists of three different realisations: The UNICORE Rich Client (URC), the UNICORE command line Client (UCC), and the Java based High Level API (HiLA). Although UNICORE provides loosely coupled WSDL interfaces for its services, to our knowledge, all available UNICORE clients have been developed in Java so far. Motivated by the demand to offer .NET applications with its powerful functionalities to access UNICORE services and to use the pull-based actor-driven approach for .NET actors, we decided to develop a solution for the .NET framework, which is introduced in this paper.

1 Introduction

Current e-Science infrastructures provide support for complex scientific processes that consist of orchestrated resources such as pure computational devices, data repositories, scientific instruments or applications. Grid environments are the common technical approach used to build these e-Science infrastructures on a middleware platform by a proper service layer. In order to support the orchestration of scientific tasks, many Grid middleware platforms offer a workflow system either as an integrative part or as an enactment service on top of the middleware. So far, all popular Grid workflow systems use an approach that follows push patterns¹. Here, a software agent, e.g. the workflow engine, actively exercises control about the progress of a workflow by pushing the individual tasks to selected resources according to the dependencies, provided by the workflow description. Consequently, in either case it is the workflow systems that takes the initiative and causes the distribution process of newly created workflow tasks to occur. The alternative solution would be a pull-based actor-driven approach, where the commitment to undertake a specific task is initiated by the resource itself rather than the system. The concept of using pull patterns to bind resources to tasks has been ported to existing Grid technologies in the scope of the ongoing project HiX4AGWS². In particular, the UNICORE Grid middleware has been extended by establishing a corresponding task repository to allow an actor-driven execution of tasks³. Of course the performance and the scalability of this pull-based approach strongly depend on the amount of available resources and applications that are able

to execute particular task instances kept within the task repository. Because of this, it is an inevitable requirement to avoid technical barriers that might occur for example because of the programming language in which an actor is implemented. While so far UNICORE especially focused the Java world, all available clients that are able to access the UNICORE service layer are implemented in Java. There are two possible ways to realise access to UNICORE services from .NET applications, so that potential actors for the task repository are no longer restricted to Java (Figure 1). The first option is to bridge the gap between Java and C# with IKVM.NET⁴. IKVM.NET is a Java Virtual Machine for the .NET runtime system. It is providing a .NET implementation of the Java-libraries and self-made Java-libraries can be translated to .NET libraries with the ikvmc-translator. The translation with this compiler from Java classes and jars into .NET assemblies is the static way. There is also a dynamic way to use Java classes and jars for running Java application on the .NET runtime environment. In this case the Java byte code is translated into the C intermediate language (CIL), which is a front-end for the C-programming language to compile programs into a subset of C⁵. Thus, everything that is needed for the .NET runtime environment is available and no further steps are necessary⁶. The other way is to write a native API for .NET, which is similar to the existing Java HiLA. We decided to realise a native .NET solution, called .NET High Level API, because not all functionalities from Java are supported by IKVM.NET (e.g. RMI) and some security properties are cumbersome to configure like keystores and policy files.

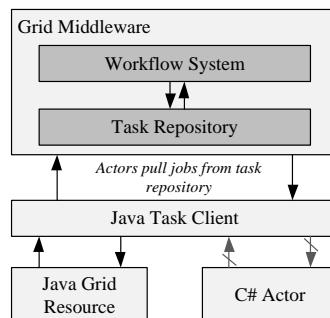


Figure 1. Actor-driven approach for job distribution according to HiX4AGWS.

Motivated by the demand to offer .NET applications with its powerful functionalities to access to UNICORE services, such as the task repository and so to use the pull-based approach with .NET actors, we decided to develop a solution for the .NET framework. With this solution it was intended to offer developers the opportunity to use the rich set of .NET capabilities such as DataGridViews, integrated reporting tools, and the Language Integrated Query (LINQ). This paper describes the architectural decisions and concepts realised within this project. Based on this .NET API, it should be possible to develop .NET client applications to access Grid resources hosted by the UNICORE middleware, e.g. by using graphical or command line user interfaces. The UNICORE services are deployed in a hosting environment that supports the standardised Web Service Resource Framework (WSRF). Thereby, the well-known factory pattern is used to create service instances that

are managed in a resource home and referenced by unique endpoint references (EPR). According to WSRF, Grid resources are represented by stateful web services that can be invoked by using particular WS-Addressing SOAP headers. Consequently, clients have to be compliant to the WSRF standard. A corresponding SOAP engine for .NET has been developed at the University of Virginia called WSRF.NET. Because this framework provides a .NET based implementation of the WSRF standard, it was an appropriate starting point to develop our .NET client API for UNICORE. WSRF.NET integrates other Microsoft specific technologies like the Microsoft Web Service Enhancements (WSE) that offers interesting development perspectives such as digital signatures and message authentication. The developed .NET API, called NHiLA, is currently interfacing with the UAS and is based on WSRF.NET 3.0.1, WSE 3.0, and the .NET Framework 2.0. Unfortunately, higher versioned .NET Frameworks are not compatible with WSRF.NET, which means that LINQ applications have to be integrated over a LinqBridge. To evaluate our solution we developed a generic client that can be used as a skeleton for future more advanced .NET clients to the UAS, as you can see in figure 2. While the generic client only implements the core communication functionality including the transport level security concepts of UNICORE, we extended this skeleton by further clients such as a TargetSystemServiceClient and a RegistryClient.

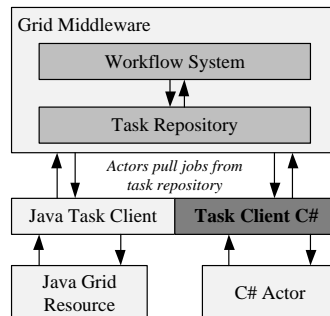


Figure 2. Integration of .NET applications as actors by a corresponding C#-client.

The paper will give an overview of our set of implemented clients and will also explore the experiences we made with respect to interoperability, functionality, and extensibility of WSRF.NET and UNICORE. It concludes with a summary and will give an outlook on the perspectives of .NET applications that interact with UNICORE services.

The paper is organised as follows. In section 2 the UNICORE client-layer is described by indicating potential entry points for integrating interfaces for C# applications. Furthermore, the currently existing UNICORE clients are explained in section 2 as well. Afterwards, in section 3 the current state of the .NET High Level API is introduced before giving a short summary and an outlook for the project in section 4.

2 UNICORE Client Layer

This section describes the existing UNICORE Java clients as well as the integration of the .NET client API.

2.1 UNICORE Commandline Client (UCC)

The UNICORE Commandline Client (UCC) is very powerful. The capability to access all features of the UNICORE service-layer in a shell or scripting environment is given with the UCC. It is possible to run jobs and to monitor their status with this tool. In addition to this, workflows can be submitted and controlled with the UCC.

Storage-Management is also provided by the UCC. File Transfer is possible from local drive to remote and additionally from server to server. One of the main advantages is, that the UCC is easily extensible, because it is simple to add new commands.

2.2 UNICORE Rich Client (URC)

The UNICORE Rich Client (URC) is an Eclipse-based graphical user interface that offers the full set of functionalities allowing users to interact with the Grid. With its graphical view of the Grid, there is the possibility to find specific resources, services or files through several filtering mechanisms. Hence, detailed resource requirements for jobs that need to be executed can be specified by the user.

Furthermore, there are modelling tools to design complex scientific workflows, like an editor that provides graphical programming. Because of this it is possible to arrange building blocks, loops or if-statements with a few mouse clicks. Security options can be configured for several panels. Therefore users are able to specify whom they want to trust on the Grid. Supplementary, they can decide how to identify themselves.

The URC is using the concept of GridBeans, which are small software packages. With these packages scientific application, which are available on the grid, can be accessed through the provided tailored graphical user interfaces. Output data of scientific simulations in the form of output files, which are downloaded to the client machine after the jobs have been executed, can be visualised by the specified software packages of the GridBeans concept. As a result the integration of various types of applications should be smooth. As previously mentioned there is an easy way to extend the GridBean, by developing a new one using the respective API and documentation.

Because of the widely spread Eclipse Environment, many users are familiar with the use of this graphical interface. So the entry barrier is lower for new users. But Eclipse-based applications are not platform independent, although written in Java. The URC tries a smooth integration into different platforms.

Useful is the sophisticated plugin mechanism of the Eclipse platform. The software components in an Eclipse-based client are plugins. They got a well-defined range of functions. After all, the UNICORE Rich Client is very extensible.

2.3 High Level API (HiLA)

There is also a third way to use the UNICORE middleware. The High Level API is specially made for Grid Applications. With this API you are capable to develop specific

UNICORE clients with a few lines of code. This will not exclude the complex functionality of the client. UNICORE 6 can also be integrated in user application without using the UCC or URC. There is a more direct way with HiLA for submitting jobs to Grid resources. So HiLA is something like a coding toolkit for building collective tier Grid services and client interfaces. The UNICORE principle of seamlessness is implemented by modelling the Grid through an object-oriented facade. That means, the underlying resources are represented in an abstract way. Furthermore, this includes the encapsulation of security configuration behind well defined interfaces. By now HiLA is evolving with the aim to operate concurrently over multiple resources. Hence, powerful collective data management and monitoring can be performed.

3 .NET High Level API (NHILA)

The introduction discussed the concept of using pull patterns to bind resources to tasks has been ported to existing Grid technologies in the scope of the ongoing project HiX4AGWS². In particular, the UNICORE Grid middleware has been extended by establishing a corresponding task repository to allow an actor-driven execution of tasks. To allow the integration of .NET applications as potential actors to the task repository, a corresponding C# client has been implemented, which leads to the resulting architecture in figure 3.

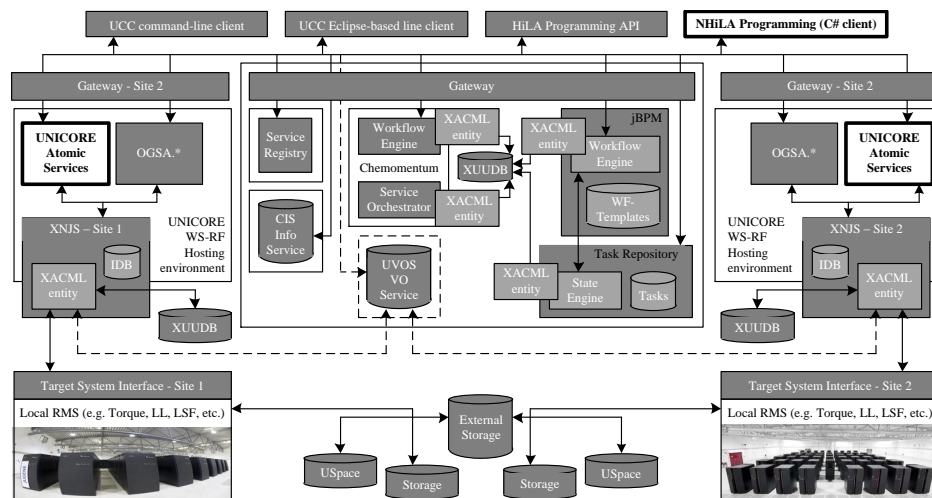


Figure 3. UNICORE Architecture with NHILA client to access the UNICORE middleware through the Atomic Services Interface.

The client layer of the UNICORE middleware has been extended by the NHILA component shown in the upper right of the figure. Currently, this API provides access to the UNICORE middleware through the proprietary UNICORE Atomic Services (UAS) only. Therefore, a base client has been implemented that is extended by each special client providing access to one particular atomic service like for example the TargetSystemService.

This section explains the architecture of the client-layer as well as WSRF.NET, the used core framework. At the end of the section, the experiences are described that are gained during the development of NHiLA using WSRF.NET tools and Visual Studio 2005 as IDE.

3.1 WSRF.NET

To realise a .NET client API for UNICORE a .NET specific implementation of the well known Web Service Resource Framework (WSRF) is needed. Here, the Grid Computing group of the university of Virginia developed a solution called WSRF.NET⁷. This framework has been developed until 2006, whereby the latest version 3.0.1 of WSRF.NET contains a collection of libraries, tools and applications according to the WSRF standard of Oasis. By using WSRF.NET it is very straightforward to build stateful Web Services that are compliant to WSRF. Beyond, the framework offers a platform to realise Grid Computing technologies with the .NET framework. Further technologies that have been developed by Microsoft are also included in WSRF.NET. This involves for example the Microsoft Web Service Enhancements (WSE) that allow the configuration of extended setups for security issues like digital signatures and certificates as well as other SOAP protocol extensions.

3.2 NHiLA Clients

This subsection describes the overall architecture of NHiLA. There is a base client that contains the similarities of all specific clients. The base client implements the basic structure of any further client which implies for example the core communication as well as the security related issues. Furthermore, the base client provides interfaces to create proxies to the corresponding UNICORE services. All specific clients that extend the base client are used to access the UNICORE Atomic Services (UAS) which are deployed as web services. The UAS offer a proprietary interface to the UNICORE middleware for clients and users. The base client is called *BaseWSRFClient*. The described structure is illustrated in the following figure.

In the following it is explained how to create a client for an arbitrary UNICORE service by using the *BaseWSRFClient*. Additionally, all existing .NET clients to the UAS are introduced.

1. **BaseWSRFClient:** The UNICORE server provides an interface description for each atomic service by using the Web Service Description Language (WSDL). Visual Studio 2005 is used to automatically generate a corresponding stub by parsing each WSDL file with a so called Add Web Reference function containing the tool *wsdl.exe*. In this process Microsoft WSE is also involved. If a proxy to access a particular UAS is requested, an instance of the corresponding stub is created. To link the stub instance with a particular Endpoint Reference (EPR) the tool *WSUtilities* is used which is part of WSRF.NET. The UNICORE service can be invoked by using the HTTP protocol and the secure socket layer (SSL). Because of this, the service consumer has to authenticate himself by a corresponding certificate that has to be introduced to the stub instance. Microsoft WSE enables the stub to provide the required security information. Thus, the authentication related information can be easily integrated into the stub by submitting the corresponding X509 certificate to the generated stub method.

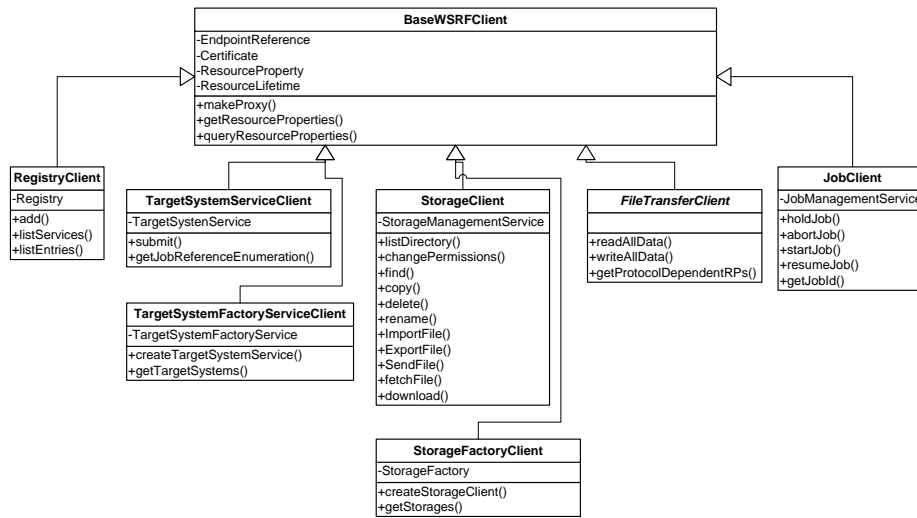


Figure 4. NHiLA Client Architecture.

Thereby, an entire proxy can be created to use full functionalities of the UAS (Figure 3).

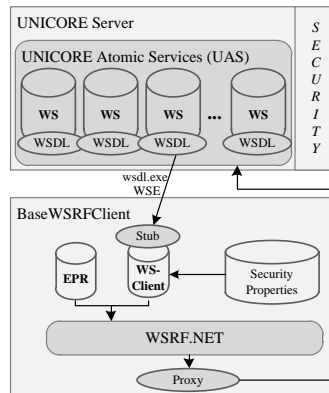


Figure 5. Stub Generation in NHiLA.

2. **RegistryClient:** The RegistryClient provides access to the Registry and the Service-Group Service. So, a user is able to add new registry entries or to look up existing service entries.
3. **TargetSystemFactoryServiceClient:** The TargetSystemFactoryServiceClient serves as a container for TargetSystemServiceClients. So it is possible to create TargetSystemServiceClient instances or to list all available target system EPRs that are accessi-

ble for the specific client.

4. **TargetSystemServiceClient:** The TargetSystemServiceClient can be used to submit jobs to concrete Grid resources or to request for jobs that are currently running on the particular target system.
5. **StorageFactoryClient:** Similar to the TargetSystemFactoryServiceClient, the StorageFactoryClient is a factory to create StorageClients. Also the StorageFactoryClient is able to list all available and accessible storage instances.
6. **StorageClient:** By using a StorageClient, it is possible to access the StorageManagement Service. Consequently, a user is able to write data to a remote file or to read data from a remote file on a particular storage that is accessible by an EPR. Furthermore, new directories can be created, files can be modified, copied or searched.
7. **FileTransferClient:** The core functionalities to manage a file transfer are provided by the FileTransferClient. There are two opportunities to send files. On the one hand you can use the Random-Byte IO and on the other hand you can send files by a Streamable-Byte IO. For both cases a particular client has been developed (RByteIOClient und SByteIOClient) that implement the corresponding technologies to transfer files. Both specific clients inherit from the FileTransferClient.
8. **JobClient:** The JobClient has been designed to realise the job management. So, the JobClient provides an interface to the job working directory. There exist methods to start, cancel, and continue or to hold a job. Furthermore, it is possible to wait until the execution of a job is finished.

3.3 Experiences

The project started with the installation and configuration of a UNICORE server. Several unexpected difficulties arose when the WSRF.NET SOAP Engine was installed. According to the University of Virginia it was known that the development of the framework stopped in 2006 and the last version is compatible with .NET 2.0. We assume that WSRF.NET is not compatible to other versions of .NET because the installation failed on a Windows 7 system with .NET 4.0. Therefore, the environment has been installed on Windows XP with a .NET framework in version 1.1. Further failures occurred during the installation of WSRF.NET because of the default usage of the free database Xindice. We switched to the SQL database of Visual Studio 2005 before WSRF.NET has been successfully installed and all supplied NUnit tests run errorless.

Further barriers occur-ed when the stubs from the WSDL files of the UNICORE Server were generated. The tool wsdl.exe is not able to work with files that are only accessible via the HTTPS protocol. Because of this, the WSDL files had to be stored local, before generating the stubs by using the function Add Web-References of Visual Studio. However, there remained some exceptions thrown during the generation procedure. The reasons were for example elements that occurred multiple times or references to XML schemas that were not reachable. These errors had been solved by deleting redundant elements from the WSDL files and storing the necessary XML schemas locally. Where needed the corresponding schemaLocation attributes in the WSDL files were adapted to the new

locations. Additionally, some any-Tags had either to be removed from the WSDLs or to be changed to specific value-types.

The existing errors had been analysed by a specific tracing tool, to monitor the SOAP traffic. Therefore, the corresponding ports of the UNICORE installation had to be adapted and the tracing tool had been interconnected. This tool helped to recognise that the content-Type entry of the HTTP header had to be changed from application/xml to text/xml, because otherwise the responses of the UNICORE servers could not be handled by the client.

4 Summary and Outlook

This section summarises the project, whose main goal has been the development of a .NET Client API for UNICORE, and gives a short outlook. The need for a C# interface occurs in the scope of the HiX4AGWS project. So far, it was not possible to connect .NET applications as actors to the task repository running in the UNICORE hosting environment. This is a fundamental issue of the pull-based approach because increasing the amount of actors is very important for the workflows' performance. Now, after the development of the .NET High Level API, corresponding .NET actors can be realised accessing the task repository.

One big shortcoming of the client API is the incompatibility with higher versions of the .NET framework, caused by WSRF.NET. So, a next step might be either to upgrade WSRF.NET or to dissolve the dependencies to that framework. As a result it would be possible to use .NET functionalities that are developed after .NET 2.0 like for example LINQ. Furthermore, the NHiLA should be more and more adapted to the existing Java HiLA provided by the UNICORE team. So, full functionalities of HiLA should be supported in the .NET-based NHiLA as well. Afterwards, it would be an option to develop a graphical user interface that also supports newer capabilities of .NET.

Acknowledgements

The .NET Client API (NHiLA) for UNICORE has been developed within the scope of the History-tracing XML for an Actor-driven Grid-enabled Workflow System project, supported by the Federal Ministry of Education and Research and the Federation of Industrial Research Associations in Germany under the project number 17N3409.

References

1. J. Yu, and R. Buyya, *A Taxonomy of Workflow Management Systems for Grid Computing*, Journal of Grid Computing (Page 1-2), Vol. 3, Oxford, Clarendon, 2005.
2. History-tracing XML for an Actor-driven Grid-enabled Workflow System, see: <http://www.fh-aachen.de/hixforagws.html>.
3. F. Berretz, S. Skorupa, V. Sander, A. Belloum, and M. Bubak, *Actor-Driven Workflow Execution in Distributed Environments*, Lecture Notes in Computer Science, Euro-Par 2010, Ischia, Italy, 2010.
4. IKVM.NET, see: <http://www.ikvm.net>.

5. C Intermediate Language at sourceforge, see:
<http://cil.sourceforge.net/>.
6. IKVM.NET User's Guide, see:
http://sourceforge.net/apps/mediawiki/ikvm/index.php?title=User's_Guide.
7. M. Humphrey and G. Wasson, *Architectural Foundations of WSRF.NET*, International Journal of Web Services Research, pp. 83-97, April-June 2005.
8. Bastian Demuth, Bernd Schuller, Sonja Holl, Jason Daivandy, Andre Giesler, Valentina Huber, Sulev Slid, *The UNICORE Rich Client: Facilitating the Automated Execution of Scientific Workflows*, Proceedings of the 6th IEEE International Conference on e-Science (e-Science 2010), pp. 238 - 245, IEEE Computer Society Press, ISBN 978-1-4244-8957-2, December 2010.
9. UNICORE Command-line Client (UCC), see:
<http://unicore.eu/documentation/manuals/unicore6/ucc>
10. Björn Hagemeyer, Roger Menday, *HiLA 2.0 - Evolutionary Improvements*, Proceedings of the UNICORE Summit 2010, IAS Series, Volume 5, ISBN 978-3-89336-661-3, pp. 45-50.

Running Local Jobs in the UNICORE Workflows

Marcelina Borcz^{1,2}, Rafal Kluszczynski¹, and Piotr Bała^{1,2}

¹ Interdisciplinary Centre for Mathematical and Computational Modelling
University of Warsaw,

Pawińskiego 5a, 02-106 Warsaw, Poland

² Faculty of Mathematics and Computer Science,

Nicolaus Copernicus University,

Chopina 12/18, 87-100 Toruń, Poland

E-mail: {mborcz, rkluszczynski, bala}@icm.edu.pl

The article presents the GridBean for running local jobs using the UNICORE Rich Client. This solution addresses the need to execute applications which are not available on the Grid such as interactive graphical tools. In the traditional approach, to run such a program in a multi-stage experiment the user has to design two separate workflows. Running local jobs, downloading the data to the local computer and uploading results of the local program had to be done manually. The presented GridBean makes it easier to run local jobs simply as a part of the workflow.

1 Introduction

The possibility to execute complicated workflows consisting of various tasks is a very important feature for the user. It is a key functionality provided by many Grid systems. The majority of the workflows is designed to run automatically without intervention from the users. Usually there are limited options to steer workflow execution and the only action which can be taken is to suspend or terminate the workflow. However, for complicated workflows there is a need to provide more advanced verification of the execution. Users would like to check files generated by the particular steps and steer the further execution depending on them. In many cases a decision cannot be performed automatically and requires manual examination of the files. This process can involve visualisation which has to be performed locally.

User interaction with the workflow has been already addressed and some solutions have been developed. Such attempts are known for Kepler¹⁰ or GWES¹² but they are based on the synchronous execution of the workflow. These solutions cannot be used in UNICORE because interactive manipulation of the workflow has to take place at the time suitable for the user which can be different from the time when parts of the workflow are executed. Additional difficulty is caused by the need to transfer data from the Grid node to the local computer used for visualisation and manipulation of the data. Therefore the execution of the interactive parts of the workflow has to be aligned with the asynchronous execution of the Grid jobs in order to release Grid resources when the system is waiting for user action. The problem of interactive execution of the workflow on the UNICORE grid is addressed here.

This paper is organised as follows: elements of Grid workflow management system are described in the next section of the paper. The third section focuses on the GridBean for local jobs. It includes the description of the plugin. Some drawbacks of the solution and ideas for an improvement are also mentioned. The example of the workflow with the local

job is described in the fourth section. It consists of BLAST¹, Clustal⁷ and R¹¹ applications. In the section there are presented some experiments in which similar programs were executed and a workflow with Local Jobs could be used. The paper is concluded with the summary and directions for further work.

2 Workflows

The definition of the workflow in Grid systems was presented by the Global Grid Forum in 2004⁵. Workflows can be seen as analysis pipelines: the output data of one subtask may be the input for other jobs. Such scenarios allow for easy usage of the distributed resources. It simplifies collaboration between scientists involved in multistage experiments. Because of that, workflows are used for multi-stage research in many scientific disciplines like biology, chemistry or physics.

There exist many Grid workflow systems such as Taverna⁸, Kepler¹⁰ or Triana¹³. In paper a by Yu and Buyya¹⁵ detailed descriptions of the systems are presented pointing out differences between them. The authors determine five elements of a Grid workflow management system: Workflow Design, Information Retrieval, Workflow Scheduling, Intermediate Data Movement and Fault Tolerance. They proposed a taxonomy of each of them. Workflow design applies to the structure of the workflow, the used model and composition system. Information Retrieval includes different strategies of getting information about resources. Scheduling is responsible for managing execution of jobs while Intermediate Data Movement focuses on staging data between resources. The last element is responsible for identifying and handling failures in workflows.

Most of the workflow systems have user-directed (language or graph based) composition systems. This approach is also present in the UNICORE middleware. The most important feature of UNICORE is the ability to build and run workflows in an easy way using UNICORE Rich Client. Sophisticated workflows including loops and if-else statements can be designed in a graphical way. Such an abstract workflow graph is translated to a concrete specification. The UNICORE Workflow Engine translates a workflow description into the set of single jobs which are run in the specific order. For finding the most suitable sites for particular tasks the UNICORE Service Orchestrator is used. It also monitors states of the execution.

Some systems like Kepler allow scientists to use programs available as web services as well as local tools. An example of a workflow consisting of such task is presented in the paper by Waddell *et al.*¹⁴ Moreover, the system can use e-mail and pause actors to allow users interaction¹⁰.

An example of an interactive workflow management system is also described by Simo *et al.*¹² The main component of the system is the Grid Workflow Execution Service (GWES) which manages and executes workflows described in a Grid Workflow Description Language based on Petri nets.

3 A GridBean for Local Jobs

3.1 Motivation for Local Execution in UNICORE

Subtasks of workflows designed and run by means of the UNICORE middleware can be executed on different target systems. In that way users have access to various programs

usually maintained by different computer centres and administration domains. Despite of this advantage of the Grid, some users need applications that may be very specific and are not available at any target system. Researchers also need to use interactive graphical tools. By running them, they want to visualise or process results of calculations made on the Grid before they will be processed by other programs. In that situation a user can execute a program on his own computer by downloading the necessary data, running the application locally and then upload the results. Such a scenario results in the necessity to run two separate workflows: one before and one after executing the application on the local workstation. While workflows can be executed from the UNICORE Rich Client, the local job has to be started separately. This is inconvenient for the users and there are requests to organise the whole process within the UNICORE Rich Client. In result all user actions can be performed with the UNICORE Rich Client making it a full featured problem solving environment.

3.2 Local Execution GridBean

To solve the problems described above we have developed a GridBean for local jobs. It can be loaded to the UNICORE Rich Client and used it in the workflows. The GridBean makes it easier to run local tasks which require user interaction. The GridBean provides user interface presented in the Figure 1 which has three text fields. The first one allows to define the name of the job, the next one allows to enter the command for execution on the local computer. In the third field there should be a directory defined where the intermediate workflow data will be downloaded. This directory is also used as the working directory for the local application.

After the subtask related to the local application is started and has the running status, the user should presses the run button. Files indicated in the workflow are then copied to the local job directory and the local application is launched on the user's computer. The researcher can now process the data using that program. When it is successfully completed, the output results which will be used in the subsequent tasks should be put in the appropriate directory and the stop button has to be pressed. The files are uploaded to the remote job directory and the following subtasks are started by the Workflow Engine.

3.3 Target System Configuration

To execute interactive workflows with the Local Execution GridBean in the way presented above, UNICORE/X and LRMS configurations have to be slightly adopted. To support local execution of tasks, a special application (named *LOCAL_JOB*) has to be registered in an incarnation database. It corresponds to a program performing the loop and preventing execution of the next steps of a workflow until a file with local job status is created. When the user finishes local execution and presses the stop button the file is uploaded to the directory of the task.

This functionality requires from the site administrator to configure the queuing system in an appropriate manner. A special queue in the LRMS batch system which submits jobs only to a dedicated machine (it may be a virtual machine) has to be configured. This queue is configured only to accept *LOCAL_JOB* tasks. Such jobs do not require many resources, so queue configuration may allow to run many of them simultaneously on a single virtual

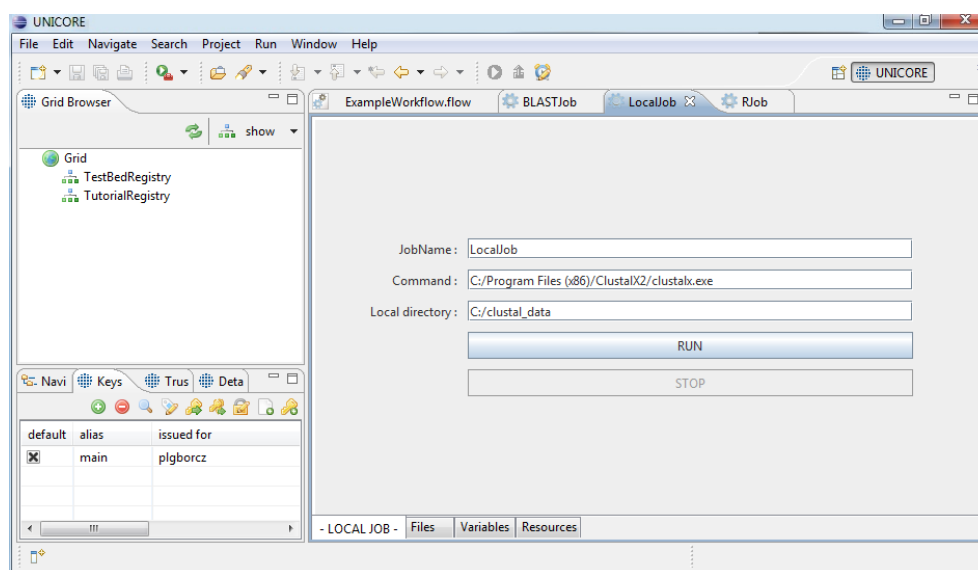


Figure 1. GridBean panel for local jobs.

machine (exceeding even the number of physical CPUs). Unfortunately, at the level of a batch queuing system there is a limit for the number of running jobs and also a limit of single job's execution time, which restrict users to run their local jobs as a part of a workflow to some period of time, for example, a week. Such modifications of the queuing system can be difficult due to administrative and practical reasons.

The same effect can be achieved by an extension of the UNICORE Workflow Engine. Namely, the pause and resume possibility should be built into the Workflow Engine. This would allow to pause the workflow before running the indicated subtask. The Local Execution GridBean would receive signal that workflow is paused and would enable the execution of local job. After the local task would have been finished, the resume signal would be generated and the workflow could be resumed. The local job would not be send to the Service Orchestrator and to the Target System, so it would not use Grid resources for local execution as it is done now.

4 Example Workflow Scenario

As it has been stated in the introduction, the execution of the workflows with the local job execution is requested by the users. Below we present an example scenario. In the Figure 2 a workflow is presented which uses BLAST, Clustal and R applications built with the appropriate GridBeans^{4,3}. At the first stage of the workflow a protein alignment is done. The BLAST program finds in the given database sequences similar to the query string and points out the places of mutations. Later the multiple sequences alignment is done by the Clustal application. If the researcher would like to do some manual verification of the BLAST results by looking for example at the phylogenetic tree, Clustal has to be run as a

local job. In this case, the graphical version of the program ClustalX⁹ is executed. In the last part of the experiment the results are statistically processed by the R program.

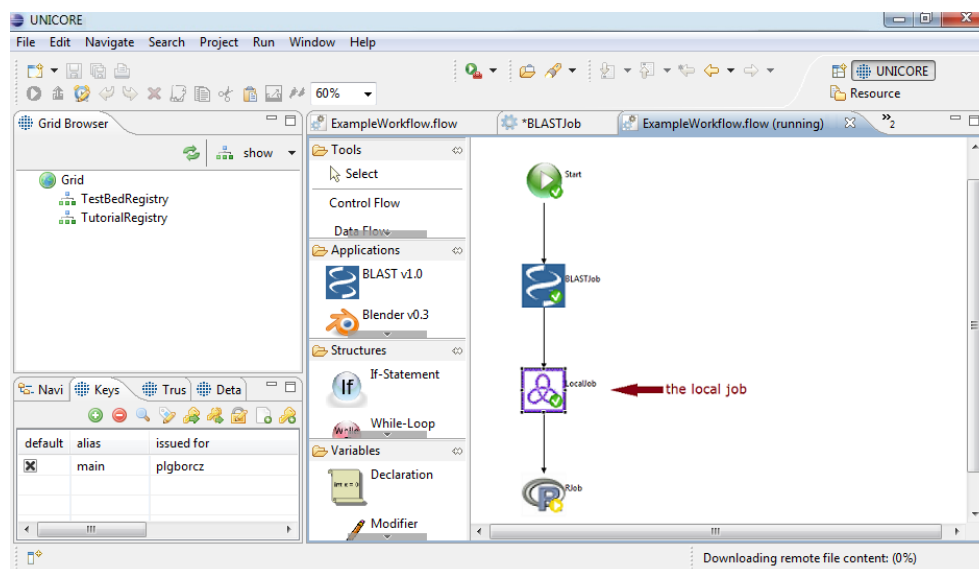


Figure 2. Example of the workflow with local job.

All files needed for the workflow should be determined during the design of the multi-stage procedure. The Clustal application aligns sequences in the FASTA format which can be extracted from the output of the BLAST program. A script performed in the R program reads the result of the multiple alignment which is written in a file with an *aln* extension. The files needed and returned by the local application are presented in the Figure 3.

An experiment which uses a similar workflow scenario is presented in an article by Bui *et al*². The authors focus on searching for sequence regions conserved across the majority of Dengue Virus Proteins. They use BLAST and ClustalX programs. The results are statistically processed by counting the Shannon Entropy. A pipeline built of BLAST and ClustalX programs followed by statistical analysis is also used in an experiment described by Fornasari *et. al*⁶. The goal of their work is finding the sequence determinants of the different quaternary assemblies of Riboflavin. Authors of perform the phylogenetic analysis which focuses on seeking for evolutionary relation among organisms. Results are usually visualised in form of phylogenetic trees. There are number of programs used for visualisation such as PAUP, PHYLIP, SplitsTree, ClustalW and ClustalX. It is usually more comfortable for the scientists to use phylogenetic tools in the graphical way. Some of them do not even have the batch option and have to be executed in an interactive manner. The Local Execution GridBean allows for easy integration of such tools with the analysis workflows run on the Grid.

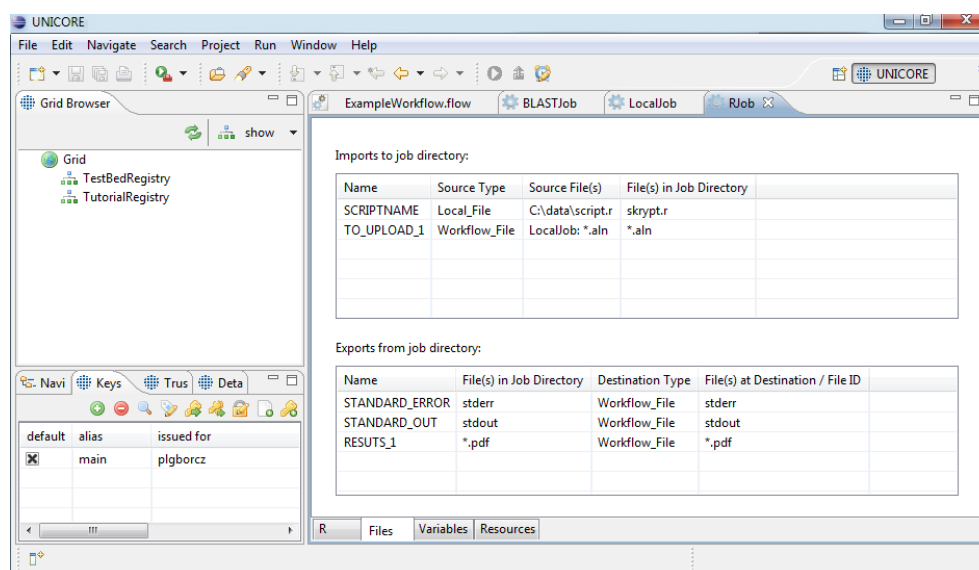


Figure 3. GridBean panel for local jobs.

5 Summary

The possibility to run the local jobs as part of the workflow is a very important feature for the researchers who use specific programs or graphical applications. In the scientific areas like bioinformatics or physics there is a strong necessity to manipulate data in an interactive way. In order to make it easier the GridBean for the local jobs has been created. However, to fully support such task there are some changes in the UNICORE workflow system needed.

References

1. S.F. Altschul, W. Gish, W. Miller, E.W. Myers, D.J. Lipman, *Basic local alignment search tool*, *Journal of Molecular Biology*, 215 (3), pp. 403-410, 1990.
2. H.H. Bui, J. Botten, N. Fusseder, V. Pasquetto, B. Mothe, M.J. Buchmeier, A. Sette, *Protein sequence database for pathogenic arenaviruses*, *Immunome Research*, 3, 2007.
3. M. Borcz, R. Kluszczynski, P. Bała, *Statistical Analysis of Biomolecular Data Using UNICORE Workflows*, *BIOINFORMATICS*, pp.n 217–220, 2010.
4. M. Borcz, R. Kluszczynski, P. Bała, *BLAST Application on the GPE/UnicoreGS Grid*, In *Proceedings of Euro-Par 2006 Workshops*, LNCS 4375: pp. 244-252, 2007.
5. G.C. Fox, D. Gannon, *Special issue: Workflow in grid systems*, *Concurrency and Computation: Practice and Experience*, 18(10), 2006.
6. M.S. Fornasari, D. Laplagne, N. Frankel, A. Cauerhff, F.A. Goldbaum, J. Echave, *Quaternary structure sequence determinants in Lumazine Synthase*, *Molecular Biology and Evolution*, 21, pp. 59–69, 2004.

7. D.G. Higgins, P.M. Sharp, *CLUSTAL: a package for performing multiple sequence alignment on a microcomputer*, *Gene*, 73, pp. 237–244, 1988.
8. D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M. Pocock, P. Li, T. Oinn, *Taverna: a tool for building and running workflows of services*, *Nucleic Acids Research*, vol. 34, iss. Web Server issue, pp. 729–732, 2006.
9. F. Jeanmougin, J.D. Thompson, M. Gouy, D.G. Higgins, T.J. Gibson, *Multiple sequence alignment with Clustal X*, *Trends in Biochemical Sciences*, 23, pp. 403–405, 1998.
10. B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger-Frank, M. Jones, E. Lee, J. Tao, Y. Zhao, *Scientific Workflow Management and the Kepler System. Special Issue: Workflow in Grid Systems*, *Concurrency and Computation: Practice and Experience*, 18(10), pp. 1039–1065, 2006.
11. R Development Core Team, *R: A language and environment for statistical computing*, R Foundation for Statistical Computing, Vienna, Austria, ISBN 3-900051-07-0, 2008.
12. B. Simo, O. Habala, E. Gatjal, L. Hluchý, *Interactive In-Job Workflows*, Proceedings of the 8th international conference on Computational Science, 1, pp. 116–125, 2008.
13. I. Taylor, M. Shields, I. Wang, A. Harrison, *The Triana Workflow Environment: Architecture and Applications*, *Workflows for e-Science*, pp. 320–339, 2007.
14. I. Waddell, N. Jones, C. Steed, X. Yuan, Yaohang Li, *Using the Workflow Technology in Secure Software Engineering Education*, Colloquium for Information Systems Security Education, 2010.
15. J. Yu, R. Buyya, *A Taxonomy of Workflow Management Systems for Grid Computing*, *Journal of Grid Computing*, 3, pp. 171–200, 2006.

UNICORE Testbed of Physics Faculty of Saint-Petersburg State University

Dmitry Kasterin and Margarita Stepanova

Department of Computational Physics, Physics Faculty, Saint-Petersburg State University
Ulyanovskaya 1, Peterhof, 198504 Saint-Petersburg, Russia
E-mail: {dmk, mstep}@mms.nw.ru

This paper presents UNICORE-related activities at the department of computational physics of the physics faculty of Saint-Petersburg State University. A brief overview of the activities that have been carried out at our department is given. The UNICORE site and its structure is described. Application of the site in education and research is discussed.

Introduction

The activities in the area of distributed computing that have been carried out at our department since 2005 are aimed at the application of grid technologies in education and science, as well as at study and experimental verification of new technological solutions¹. One of the directions of our work is development and support of a multi-functional computational system based on cluster resources of the physics faculty. The system is used mainly in educational activities on parallel and distributed computation technologies, but the resources are also granted to students that perform computations for their scientific work.

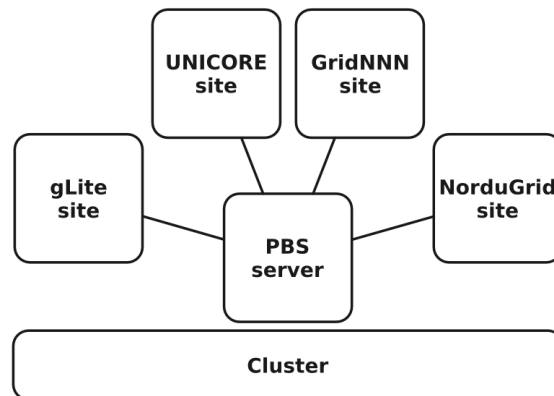


Figure 1. Scheme of the integrated grid system.

We have created an integrated grid system that provides simultaneous access to a single computational cluster from various grids (GridNNN, NorduGrid ARC, gLite and UNICORE based). Figure 1 gives a schematic overview of the system. In such configuration we can avoid division of available hardware resources into several clusters with their

own settings. The proposed solution provides high mobility and utilisation efficiency of computational resources and enables students to gain wide practical experience with extensive set of modern grid software. Access to user jobs and scientific software packages can be granted via both a batch system and grid sites.

UNICORE Site

Our interest in the middleware has arisen for several reasons.

- First of all, UNICORE is an original project and does not rely on side components.
- Secondly, we believe UNICORE not to be widespread and well-studied in Russia as compared to the other middleware, e.g. Globus Toolkit.

We have accomplished the installation of a basic UNICORE 6.3.1 bundle and configuration of a site. The following services have been installed: Gateway, UNICORE/X, XUADB and TSI. All services use valid certificates from Russian Data Intensive Grid (RDIG) CA. User certificates solely from RDIG CA are supported.

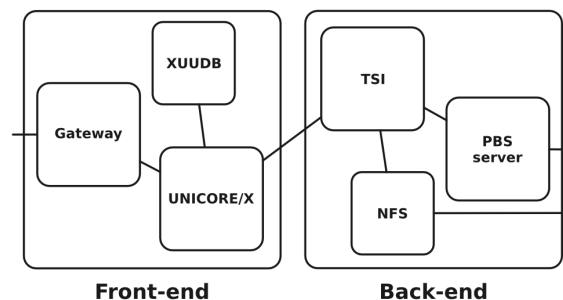


Figure 2. Structure of the site.

The site comprises (figure 2):

- a front-end node with Gateway, UNICORE/X and XUADB services
- and a back-end node with TSI service and an instance of PBS server connected to a cluster

Such placement of the services meets our needs in the best way.

A cluster comprises 12 2xIntel Xeon 3GHz Dual Core HT / 4GB RAM / Scientific Linux 5.4 nodes and is operated by PBS Torque 3.0.0. Special measures have been taken in order to improve the control of resources:

- PBS has been built with `cpuset` Linux kernel feature patches enabled²
- OSC `Mpiexec` utility, which supersedes `mpirun` and `mpiexec`, is used³

A dedicated queue with 32 CPUs for UNICORE jobs is available.

Educational Activities

On the basis of the integrated grid system, seminars are held. These seminars are the part of a special course on grid technologies for students of the physics faculty and the mathematics faculty. The course enables students to learn modern computational systems and tools, to put into practice their theoretical knowledge and to acquaint themselves with the ways grid technologies can be applied in their research work. Special attention is paid to launching of scientific applications. During the course students have to accomplish their self study that includes development of their own applications with the use of MPI and OpenMP technologies and compilation with the necessity to run those applications on grid sites. This spring, the UNICORE site was used in the course for the first time. Students have learnt about the command-line client, job description language and also rich client.

Scientific Packages

The underlying cluster is used not only in education, but also for scientific computations. The main part of load is molecular dynamics and quantum chemistry simulations that are performed by various professional scientific packages.

Using the IDB mechanism we have made the LAMMPS⁴ and Firefly (PC GAMESS)⁵ packages available on the UNICORE site. The packages can be launched in both serial and parallel modes. Parallel mode is available by means of an MPI execution environment.

Here is an example of a LAMMPS job description:

```
{
  "ApplicationName": "LAMMPS",
  "ApplicationVersion": "10Sep10",
  "Arguments": [
    "-echo log"
  ],
  "Environment": [
    "INPUT=in.pour",
    "LOG=log.pour"
  ],
  "Imports": [
    { From: "in.pour", To: "in.pour" }
  ],
  "Exports": [
    { From: "log.pour", To: "log.pour" },
    { From: "dump.pour", To: "dump.pour" }
  ],
  "Resources": {
    "Nodes": 2,
    "CPUsPerNode": 2
  },
  "Execution environment": {
    "Name": "MPI",
    "Options": [ Verbose ]
  }
}
```

Conclusion

UNICORE site plays an important role in educational and scientific activities of our department. Our future plans include maintenance and further development of this site. We believe that this will enable us to provide students with the most recent versions of grid middleware packages and to continue our research on grid technologies.

References

1. M.M. Stepanova, A.N. Makarov, S.L. Yakovlev. Grid-systems in process of research and teaching at faculty of physics of SPbSU. Distributed Computing and Grid-Technologies in Science and Education: 3rd Intern.Conf. (Dubna, June 30 – July 4, 2008). — Dubna: JINR, 2008, pp.113 – 114.
2. Linux cpuset support in Torque, last visited: June, 2011,
<http://www.adaptivecomputing.com/resources/docs/torque/\\3.5linuxcpusets.php>.
3. Mpiexec home page, last visited: June, 2011,
<http://www.osc.edu/~djohnson/mpiexec/>.
4. LAMMPS Molecular Dynamics Simulator home page, last visited: June, 2011,
<http://lammps.sandia.gov/>.
5. Firefly home page, last visited: June, 2011,
<http://classic.chem.msu.su/gran/gamess/>.

EMI UNICORE Execution Services for Interoperability Across Middleware

**Michele Carpené¹, Ahmed Shiraz Memon²,
Mohammed Shahbaz Memon², and Bernd Schuller²**

¹ CINECA

40033 Casalecchio di Reno, Bologna, Italy

E-mail: m.carpen@cineca.it

² Jülich Supercomputing Centre,

Research Centre Jülich, 52425 Jülich, Germany

E-mail: {a.memon, m.memon, b.schuller}@fz-juelich.de

Increasing demand of computing and data infrastructures has provided new capabilities for large scale scientific initiatives supporting European research. A number of major projects have been established within Europe to help communities to share resources, access them, execute jobs, collect results and share information. What is needed now is the consolidation of the computing infrastructure and a simplification and standardisation of middleware. The European Middleware Initiative (EMI) is a collaboration of the four major software tools: ARC, gLite, UNICORE and dCache, the final goal is to integrate middleware components, deploying them in EGI, PRACE and other DCIs, providing at the end a single release. EMI components currently implement job description and job management in different ways, generally incompatible, even if the provided core functionality is approximately the same. Many standard mechanisms for job description and standard interfaces for job submission have been created and used in production, as for example JSDL and OGSA-BES, but now a more general specification is required to define a common pattern for the creation and management of single activities on a computing element. This issue is being addressed within the Middleware Development working group and a specification for an EMI Execution Service has been defined. This results in an agreement about which interfaces to use in order to perform the required activities. Following this specification, the job submission, data staging and other base functionalities are going to be implemented within the EMI UNICORE ES (Execution Service). This paper presents the UNICORE ES architecture and implementation, motivating and discussing proposed solutions. First the interface specification is shown, then the general software architecture is presented. Finally it is explained how EMI ES will be based on standard UNICORE libraries.

1 Introduction

In the last decade new infrastructures have been established to offer middleware services to scientific community users. Many scientific applications and projects grew up and needed the support of different resources and Grid services. In Europe the Distributed European Infrastructure for Supercomputing Applications (DEISA⁵) provides services and interfaces to services, adopting tools as the UNICORE 6 Middleware³ for HPC applications, meanwhile gLite Grid middleware continues to provide computational capabilities to physicists and scientists. Recently the project EGI-InSPIRE⁷ has been created to coordinate and maintain a sustainable European infrastructure to support European research communities and their collaborators.

Currently ARC, dCache, gLite and UNICORE are the major European middleware providers, and it is desirable to consolidate these components for the deployment in EGI, as

part of the Unified Middleware Distribution (UMD). The EMI project (European Middleware Initiative⁴) is in charge of that activity and will work with EGI-InSPIRE to enable the vision of providing European scientists and international organisations with well-designed services, also supporting users on incidents. EMI also plans to offer its middleware to PRACE⁶ and other distributed computing infrastructures. In that context a strong analysis of the middleware capabilities is needed to plan the integration of heterogeneous services and the consolidation of an uniform software layer. The new middleware product will be in charge of performing all the activities related to user accounting, job management, storage management, security and everything about super computing applications support. In particular EMI ES have been conceived as a set of services for job submission and execution, designed to reach interoperability and to offer multiple capabilities for different users in the scientific community. EMI ES should be seen as a common access point to the Computing Elements, to define how this can be reached a document has been produced, including the specification for the services functionality. The EMI ES implementation will integrate existing functionalities, following the specification guide and will expose these interfaces to the external world.

2 EMI Execution Services Specification

The EMI Execution Services Specification¹ provides the interface description, data and state models, activity, and resource specification for each single execution service. This interface represents an agreement between ARC, gLite and UNICORE. The detailed specification of the EMI ES is a continuation of the work previously done in PGI² (Production Grid Infrastructure) and covers many different aspects:

- Common Interfaces to create and manage activities
- EMI Activity Description Language (an XML dialect for describing the activity including data staging and resource requirements)
- Data staging capabilities
- Activity related information
- Resource related information
- Delegation

The specification covers the same aspects as the existing OGSA-BES⁸ and JSDL⁹ specifications. However, several features only available in JSDL as extensions have been integrated into the EMI ES specification in order to create a unified XML schema. The Activity Description XML Schema covers data staging, execution, runtime environments, parallel applications and resource description, it can be seen as a merger of the JSDL 1.0 schema and some critical and often used extensions. The EMI Execution Services introduce capabilities, as for example array lists to pass identifiers as a parameter set. This is useful in order to submit multiple jobs, or to retrieve information about multiple activities in a single call. Parallel job submission is supported and a flexible model synthesises the capabilities of the involved middlewares for what concerns data staging.

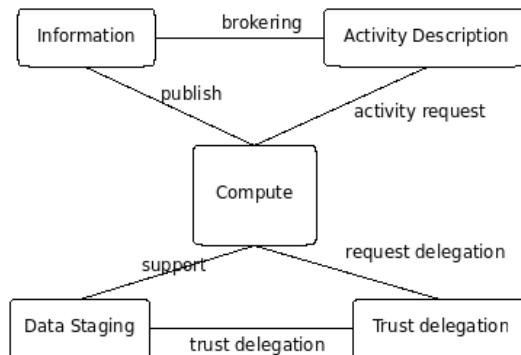


Figure 1. Interaction between elements.

Figure 1 shows how multiple elements are interconnected around the Compute Element: the Information element publishes information about the Compute part, the Compute Element supports Data Staging and the Trust Delegation mechanism is in charge of providing Proxy support for authentication.

The GLUE2 schema is used consistently as the information model for resource and activity descriptions. The information about resources exposed by the service can be published following the GLUE2 model. The resource description of the EMI-ES is based on the ComputingService and the generic service elements of the GLUE2 XML rendering. For what concerns Data Staging there are two main modalities to perform both *stage-in* and *stage-out* operations. The first is the “Server data pull” modality: the ES pulls needed data from the specified sources and makes them available in the *session* directory, or pulls data from the ES *stage-out* directory. The second one is the “Client data push” modality: the Client uploads data into the *stage-in* directory and alerts the Server by mean of a flag, or pushes relevant data into the specified target. The only data transfer protocol allowed at the moment by the ES specification is GridFTP, it is used for everything regarding file transfer or data management before or after the execution.

Finally the Delegation part consists of a compromise between the different Middleware needs since a sophisticated Delegation mechanism is not always provided, a base Proxy support has been added while later this will be replaced by a more powerful one.

The EMI ES are composed of two main modules: The first one is the *Activity-Factory*, responsible to create activities and manage resource information. It implements the following web service interfaces (WSDL port-types) and operations:

- Create port-type: CreateActivity
The Create port-type gets the Activity Description in XML format (ADL) and performs the activity creation.
- ResourceInfo port-type: GetResourceInfo, QueryResourceInfo
The ResourceInfo port-type contains methods to retrieve information related to the Computing Element and doesn't include information about activities created through the Activity-Factory.

- Delegation port-type: InitDelegation, PutDelegation, GetDelegationInfo
Delegation provides Proxy support.

The second module is the *Activity-Manager*, which is responsible to manage activities and activity related information. It implements the following port-types and operations:

- ActivityManagement port-type: GetActivityStatus, GetActivityInfo, NotifyService, PauseActivity, ResumeActivity, CancelActivity, WipeActivity, RestartActivity
The ActivityManagement contains methods to perform requested operations on activity lists, when one of the methods is called the list of the activity IDs is passed as an argument and for each activity ID the proper function is invoked and the job status changes accordingly.
- ActivityInfo port-type: QueryActivityInfo, GetActivityStatus, GetActivityInfo
The ActivityInfo part offers methods to get GLUE2 schema information about activities, when an identifiers list is passed to the GetActivityInfo method an array containing info objects is returned.

The Execution Service agreement represents a significant milestone for the EMI project and will enable transparent job submission and management to the different EMI middlewares; future modifications of the specification due to external requirements are expected.

3 EMI Execution Service Implementation in UNICORE

EMI ES are currently being implemented in UNICORE following the specification. From a general point of view the ES code is composed by four main standard SOAP web services, associated to the corresponding interface Port Types: the *DelegationService*, the *Create-Activity*, the *ResourceInfo* and the *ActivityManagement*. Even though services signatures do not perfectly match module names reported above, they perform all the functionalities described in order to implement the whole execution mechanism. Since EMI is actually in progress, and we're at the end of the first project year, many services, which are going to be shown in the next phase, have not yet been realised.

In the future other components, as for example the EMI Client and the EMI Registry, will be introduced and they will provide functionalities similar to the standard one from UNICORE. What we're describing here is the way the UNICORE EMI ES interact with standard UNICORE Services and how they rely on UNICORE base classes and packages.

3.1 The DelegationService

The first component which has been implemented is the *DelegationService* (DS), which interacts directly with the Client before a job submission request. Its purpose is to provide Proxy certificates to be used for data staging. This is required, since currently proxy certificates are the only available trust delegation mechanism that can be used within all the three middlewares.

The DS can be contacted by the Client, which receives from the server a CSR (Certificate Signing Request), the Client then creates and signs a Proxy certificate with its Private Key and sends it to the Server. The server verifies and stores the Proxy with its proper delegation ID. Once a Proxy has been created the service can use the delegation ID to perform

all the requested operations. More than one Proxy can be associated with a single user, this can be useful if we want to use credentials belonging to different VOs to query the same service. In this way we have to remember the list of delegation IDs associated with the related Proxies. At the moment the EMI UNICORE Delegation Service is used and tested for Job Submission and Data Transfer activities by GridFTP.

3.2 The Creation Port-type

The *CreateActivity* interface method is invoked to submit jobs to the system, the method gets the Activity Description and creates a corresponding Java object which can be used to submit a single job.

The XML Activity Description is first validated, then the activity can be created and the Client gets back the information about the path where to import the script input files. Even if three directories are expected following the specification: *stage-in dir*, *stage-out* and *session*, these three directories are the same in UNICORE, and only the session directory is used to move input data; after negotiating the Activity creation with the server now the Client is able to copy any kind of input file or executable using the URL received.

In Figure 2 the whole sequence is shown: The Client first invokes the *InitDelegation* and gets the Certificate Signing Request (CSR), then sends the signed Proxy to the Server with the *PutDelegation* method and finally asks to create the Activity. When the *StageIn* directory URL is returned the Client stores input files and notifies the service to start the job, the Proxy is retrieved by means of the delegation ID previously negotiated.

Data staging operations are executed following the modalities explained previously in the specification paragraph. UNICORE supports multiple protocols such as ByteIO for file transfer and data staging, here GridFTP will be the only one used. Thus, a local installation of the GridFTP Server has to be used to perform GridFTP transfers employing the DS Proxy. The client code invokes the *globus-url-copy* tool to move files. If an input file will be used during execution the Client pushes it into the *session* directory and after the first negotiation phase the Client contacts the Server using the *notify* method, the Server knows that data are available in the *session* directory and starts up the job which goes into the PROCESSING-RUNNING state, as the specification requires, finally the job unique id is generated and stored to be retrieved later. Each time the Server has to act on the behalf of the activity owner the delegation support is required.

3.3 The ActivityManagement Port-type

Once the job has been created, the *ActivityManagement* interface offers methods to perform operations handled by the *Activity-Manager*, as for example: *getStatus*, *removeJob*, *pauseJob*, *resumeJob*, *notify*, *getActivityInfo*. Each method is called passing an activity identifier list as it is expected from the specification.

If the *removeJob* is called the job is removed and the corresponding session folder is deleted. If instead the *pauseJob* is called the job is paused and resumed only if the *resumeJob* is invoked. The Client invokes the *notify* just to alert the service that input files have been stored. In case the *getStatus* method is invoked the *ActivityManagement* asks the UNICORE execution manager (XNJS) for the status of a number of jobs and maps the UNICORE status values to the corresponding EMI status values. Many different State definitions have been specified in the interface document, possible states are assigned mapping

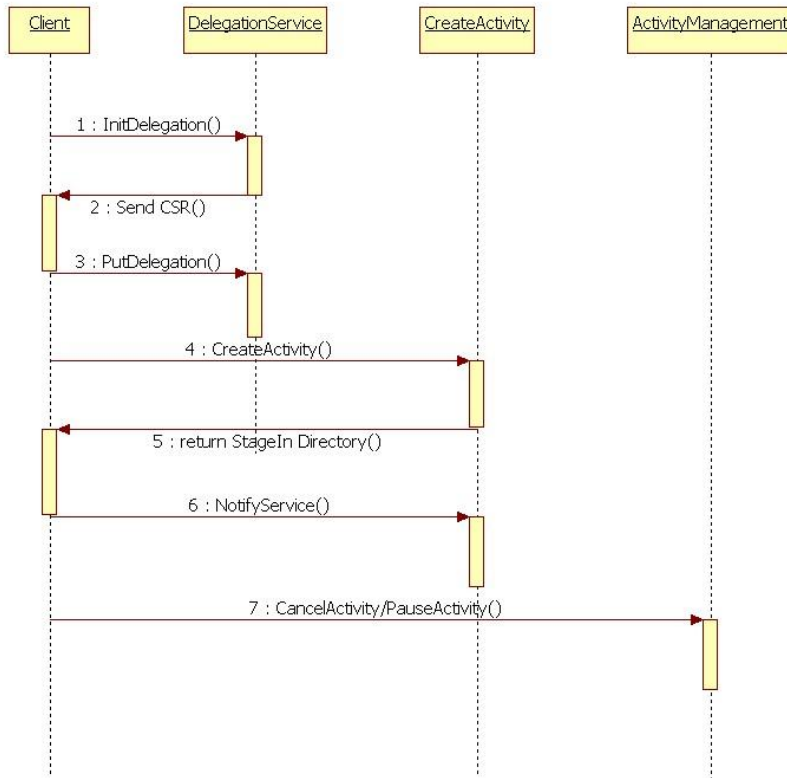


Figure 2. DelegationService invocation, Activity creation and Client data push.

the job status number to the corresponding *EMI ActivityStatus* object which is filled by setting the status property. For example if the status number is given by *ActionStatus.DONE*, the corresponding EMI status is *TERMINAL*:

```
status.setStatus(PrimaryActivityStatus.TERMINAL);
```

An attribute is also added to specify the cause of a certain status or to provide additional information about the action performed by the service:

```
status.addNewAttribute().setStringValue("APP-FAILURE");
```

Finally the status object is returned to the caller.

The *getActivityInfo* method instead returns the list of activities and their attributes, the activity list is passed as an argument to the *getActivityInfo* which iterates through activity ids and for each one of them, it stores the GLUE2 activity model properties. Then an array of *ActivityInfo* elements is returned.

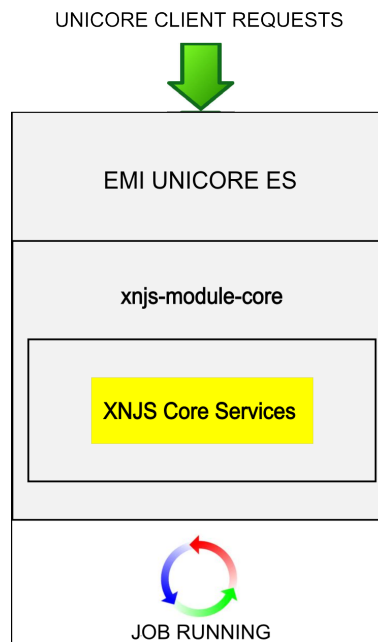


Figure 3. Core Services.

3.4 The ResourceInfo Port-type

The resource information about the Computing Element is provided from the *GetResource-Info* method according to the GLUE2 computing service model. All the information about CPUs, memory available and system features is returned directly from the UNICORE XNJS.

4 UNICORE Libraries

The main goal of this work was to use existing UNICORE base components for Job Submission and implement a Java interface to integrate them into EMI services. This has been achieved importing UNICORE XNJS libraries as dependencies and starting to create a new code layer above, which implements features and methods. In detail the *xnjs-module-core* has been included into the project configuration file in order to import base XNJS functionalities. The classes contained in the XNJS dependency module provide methods to start services and to support submission. When the EMI services are invoked to submit a job a new activity is created starting from the XML ADL and the activity description is processed and validated, then in case the XML is compliant with the XML-ADL schema the job can be executed using standard job submission mechanisms. When *notify* is called the proper XNJS run method is invoked passing on the identifier of the created job, since the job ID stored by the XNJS server is the same as stored by the Execution Services. Once an activity has been created the XNJS Server takes care to track job status and log

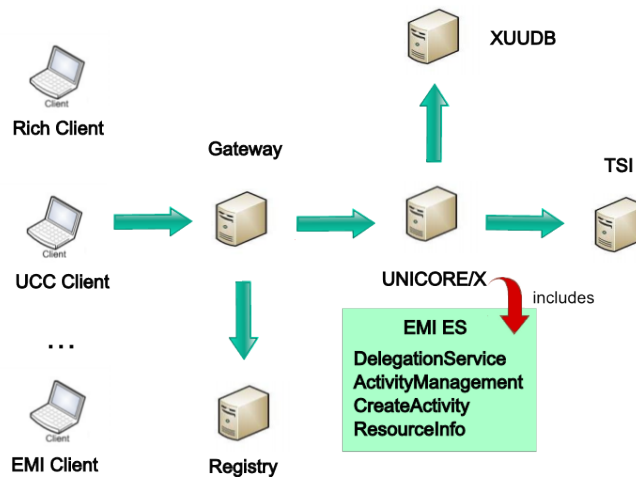


Figure 4. Global system view.

events, each time an interface method is called the corresponding standard UNICORE method is invoked and the job status changes accordingly, Figure 3 shows how the EMI services interact with standard UNICORE Server libraries. The UNICORE ES will be integrated into the standard UNICORE/X server and runs in a UNICORE container interacting with the other standard UNICORE elements as shown in Figure 4. Configuration is done using the usual configuration files and the services have to be declared in the .xml file which lists the services to execute, in order to be started up when the Server is up and running. It is expected that the UNICORE ES will publish its address to the EMI Registry service which is another EMI development. Then, the EMI-ES client will contact the EMI Registry to get the list of the available services.

5 Conclusions

The EMI Execution Service is an important common specification between ARC, gLite and UNICORE, which aims for a common mechanism for job submission and management, covering important scenarios from both high-performance and high-throughput computing. The EMI-ES specification and its ongoing implementation in UNICORE have been described, starting from its specification, explaining how the interface implementation has been achieved and how these services are going to be integrated with the UNICORE core servers and how a future EMI Grid environment can use these services.

The standards-based interoperability between ARC, gLite and UNICORE with respect to job submission and job management remains the fundamental goal to reach, and future work will focus on Common Authentication Libraries, Explicit Trust Delegation and Storage Data Management.

Integration Tests will be performed between different EMI middleware components and it is expected that these services will become a stable part of the EMI release starting from the beginning of the second project's year.

References

1. B. Schuller, A. Konstantinov, M. Sgaravatto and other contributors: EMI Execution Service Specification https://twiki.cern.ch/twiki/pub/EMI/EmiExecutionService/EMI-ES-Specification_v1.03.odt.
2. B. Konya, M. Marzolla, M. Riedel and other contributors: PGI Execution Service Specification http://ogf.org/gf/group_info/view.php?group=pgi-wg.
3. UNICORE <http://www.unicore.eu/>.
4. EMI <http://www.eu-emi.eu/>.
5. DEISA <http://www.deisa.eu/>.
6. PRACE <http://www.prace-ri.eu/>.
7. EGI <http://www.egi.eu/>.
8. I. Foster and other contributors: OGSA-BES Specification <http://www.ogf.org/documents/GFD.108.pdf>.
9. A. Anjomshoaa, M. Drescher and other contributors: JSDL Specification <http://www.ogf.org/documents/GFD.136.pdf>.

Compiling and Running Custom Code in the UNICORE Middleware

Piotr Piernik², Piotr Bała^{1,2}, and Krzysztof Benedyczak^{1,2}

¹ Interdisciplinary Center for Mathematical and Computational Modelling,
University of Warsaw, Warsaw, Poland

² Nicolaus Copernicus University
Toruń, Poland

E-mail: {piernik, bala, golbi}@mat.unk.pl

In this paper we present two solutions which help UNICORE users to remotely build and subsequently execute custom applications. The first one possess an interface with all intended features. It is easy to use and should be a valuable addition to the UNICORE client-side portfolio from the user perspective. The second one utilises Execution Environment framework and extends it to support automating build, compilation and execution of the users codes. Both solutions are intended to fill the gap between the offering of the generic UNICORE software and highly sophisticated D2E environment targeted at experienced developers.

1 Introduction

The UNICORE middleware¹ provides a highly sophisticated interface for executing applications preinstalled on target sites. In addition to the standard execution, it is possible to discover such applications, get information about their required arguments and use them in brokering process. The user can be aided with an application-specific, pluggable component called GridBean². GridBeans, loaded into the UNICORE Rich Client can provide a complex GUI for an application's input preparation, local post-processing and visualising the application's outcome. The main drawback of the GridBean mechanism is the necessity to provide components dedicated to the particular application. This process requires knowledge of Java programming and is not accessible to most of the users. The Generic GridBean, one of the recent advances in the UNICORE client-side eliminates this problem. It can dynamically build an interface based on an enriched specification of application arguments defined on the server side.

Invocation of programs which are installed by site administrators is often insufficient. The users may want to execute custom applications: both third party and developed by themselves. Execution of an application which is available in a binary form (or is programmed in a directly interpreted language like Python or Ruby) does not present a problem in UNICORE: the Generic GridBean can be used to upload a binary and execute it. The upload of the application and its execution by the Script GridBean is also possible and simple. Unfortunately, custom applications are often provided in a source code, so they need to be compiled before execution. Such functionality is not provided by the GridBeans mentioned above.

Compiling and running own applications presents an issue for the UNICORE users. Assuming that compilers are defined as applications on the server side it is hard to use the Generic GridBean for building an application. Sources must be uploaded in an unpacked form (using the "directory" upload option) what is slower then uploading an archive and

extracting it on the server side. Moreover, builds involving more than one step are impossible. Unfortunately this is the case of GNU autotools which are used in a vast majority of C and C++ programs. Users are therefore forced to use Script GridBean and program the whole compilation logic. This approach is also imperfect as it requires users to write a compilation script and to manually discover locations of specialised compilers which are installed at site specific paths. This is especially hard when multiple versions of a compiler are available and different versions require different command line arguments.

The situation gets more complicated when the application is not a simple serial program, but a parallel MPI program. In this case the user is unable to take an advantage of UNICORE support provided as MPI Execution Environments unless the building and invocation process is split into two steps. In this case and also in the situation where the application is going to be used several times, the user has to manually upload the executable file to a permanent storage and write another script to download and invoke it.

In this paper we describe a solution which helps users in the case of building and executing custom codes via the UNICORE middleware. The solution is developed in two similar variants: one for standard execution and another one for MPI codes.

2 Existing Solutions

2.1 DEISA D2E

DEISA⁵ is a large European project maintaining a Grid of HPC resources. The Grid infrastructure is exposed to the users using the UNICORE middleware with some additional project specific extensions and features.

DEISA D2E³ is an advanced Grid development IDE, based on the Eclipse Parallel Tools Platform (PTP) technology⁴. Eclipse PTP, which is an interesting solution on its own, provides a set of features for the development of parallel applications. Eclipse PTP allows for compiling, debugging, launching and profiling different kinds of parallel programs, like MPI or OpenMP based. The development machine is usually not an execution machine so Eclipse PTP provides an option to perform all the above functions remotely. Eclipse PTP integrates with a variety of resource schedulers (like SLURM or PBS). To access remote machines Eclipse PTP requires the ability to open SSH tunnels. One should note that on the remote side a special server component (or components) needs to be installed.

DEISA D2E is a tool which extends the fundamental Eclipse PTP features by integrating it with the UNICORE middleware. It is possible to use an additional authentication mechanism (GSI-SSH, which is enabled at DEISA sites) and more easily port applications to various resource managers (what is a core UNICORE feature). Additionally, D2E allows users to use the PARAVR⁶ profiling tool.

2.2 MPI-Start

MPI-Start⁷ is another tool which simplifies development and execution of distributed MPI programs on the Grid. MPI-Start solves the following MPI-related problems which arise in a heterogeneous Grid environments:

- Discovery of selected MPI flavour location on a target system.

- Distribution of compiled binaries of user's application (and potentially its other input files) to nodes assigned by a Local Resource Management System (LRMS).
- Collection of outcomes.

It must be noted that the above steps are usually realised in a very similar way but there are differences related to the particular versions of LRMS and MPI implementations. MPI-Start tries to address them.

MPI-Start is implemented as a set of scripts, which needs to be installed on a Grid target system. MPI-Start detects (or can be configured) to support the available LRMS and one or more MPI implementations. MPI-Start consists of a core module and its plug-ins, which can be implemented using three types of frameworks:

- Scheduler framework — provides LRMS integration, SGE, PBS and LSF are currently supported.
- MPI framework — provides MPI implementation integration, OpenMPI, MPICH, MPICH2, LAM-MPI and PACX-MP are supported.
- File distribution framework — provides a file distribution capability. The fundamental ones are based on a shared file system and on SCP tool.

The user who wants to use MPI-Start must prepare special scripts which define commands and environmental variables. The variables are the main MPI-Start interface, allowing for its customisation. Additionally pre- and post- invocation hooks can be defined with commands to be run in specific application build phases.

2.3 Execution Environments

Since its beginnings UNICORE provides a possibility to invoke remote applications, described in a platform independent way. From the version 6.3.0 on it is possible to use so called execution environments which allow for executing an application in a dedicated environment. One of the classic examples is the invocation with MPI.

UNICORE advertises available execution environments and their parameters. With support built into the UNICORE Rich Client a user can edit the chosen environment's options in a convenient way, using an auto-generated graphical interface. UNICORE server side components translate execution environment parameters into a site-specific form.

Execution environments can support invocation of MPI applications, they can also expose various MPI implementations (for each a separate environment should be defined). However it is hard to perform compilation using this facility. Additionally, the administrator has to prepare a complicated description of each MPI flavour on their own. In UNICORE it is assumed that a shared cluster file system is available, therefore coping of input files and executables is not a strictly required feature.

3 Target Audience

D2E provides a highly sophisticated and advanced development environment for UNICORE users. However, D2E does not cover the needs of all kinds of users. First

of all it requires a specialised installation on the server side. GSI-SSH is not a typical access method for the UNICORE systems, the installation of Eclipse server side tools must also be performed. Quite often users want to invoke an application which is not installed on a target site. Users who are not advanced programmers and have developed a simple application may only want to perform some trivial testing to fine tune the source. In their case usage of a complicated system as Eclipse PTP or D2E can be considered as an unacceptable overhead.

In this paper we present two solutions which try to address the needs of the audience described above. The aims of the solution are:

- Support for building, reusing previously built binaries and the execution of compiled programs.
- Tight integration with UNICORE client-side. This point ensures that existing UNICORE users will be able to quickly learn the new tool.
- Maximum utilisation of the UNICORE server-side capabilities, what should help UNICORE administrators to deploy the solution easily.
- Providing support not only for parallel applications, like MPI, but also for serial programs which are also run on the Grid.

4 The Proposed Solution

4.1 The Initial Approach

Our primary solution is composed of three parts: an advanced GridBean component called CaR (Compile and Run) GridBean, a helper Bash script (further on called the CaR script) which should be installed on the server side and a special configuration that must be placed in the UNICORE IDB.

The CaR GridBean is responsible for input preparation and job description which invokes the CaR script. The script can perform one or more of the four available phases:

- Sources preparation phase — prepares sources for compilation, what usually means that the uploaded archive must be unpacked.
- Compilation phase — compiles the sources using a chosen compiler.
- Execution phase — invokes the binary application.
- Outcome export phase — prepares the compiled application for export, usually by packing it into a single archive.

Invocation of the MPI applications is done using the MPI-Start framework. The CaR script prepares the necessary MPI-Start configuration so the user does not need to be concerned with the MPI-Start specifics.

The most interesting part is the compilation phase. A sequence of compilation commands is prepared on the client side by the CaR GridBean (see the fig. 1). This is possible because description of available compilers and their arguments is provided in the IDB. The

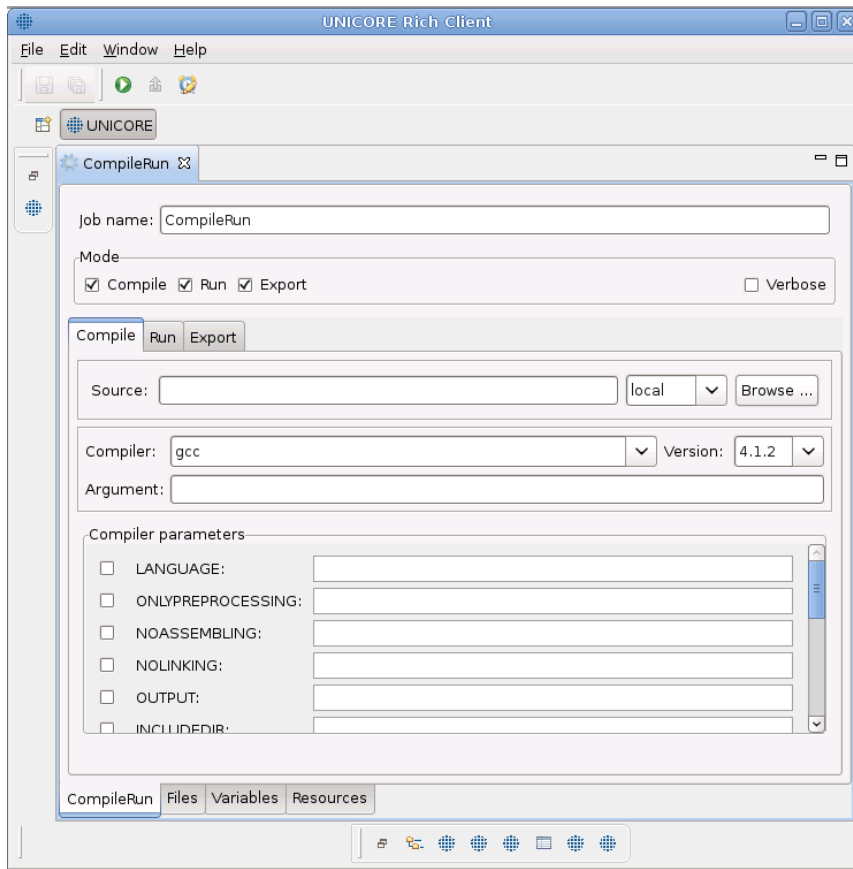


Figure 1. The compilation tab of the CaR GridBean.

CaR GridBean can therefore show a list of available compilers and generate an interface with the possibility to set all options. The user has to enter the execution command. Of course he can select which operations (phases) are to be performed and for example only compile the sources without executing them.

It can be noted that IDB configuration plays an important role in the solution. It is required to add an entry for the CaR script and all compilers. As we already wrote, the CaR GridBean prepares the final build commands. To be able to create them the IDB must publish each compiler location and actual names of parameters (normally only the symbolic names are published). This is done by additional entries in the Info section of the IDB for each compiler. An entry in this section also informs the CaR GridBean which of the available applications are compilers.

It is also possible to put two additional kinds of entries in IDB. One is for the MPI-Start framework. Its presence informs the CaR GridBean that MPI mode can be enabled. Finally, with an appropriate IDB entry it is possible to enable support for the popular GNU autotools suite.

4.2 Execution Environment-based Solution

The first solution which we developed provides a user friendly interface. Despite of this fact it also has two disadvantages: the IDB descriptions are extremely complex (some of the data about compilers needs to be entered twice) and a job prepared by a CaR GridBean is not abstract — it can be only invoked on a pre-selected target system as compilation commands are written in input files.

To circumvent the above issues we have designed an alternative approach. The idea is to expose compilers as normal applications in the IDB and the main CaR script as a special Execution Environment. All additional arguments can be passed as Execution Environment parameters. With such an approach the compilation command is built automatically by the UNICORE XNJS. By using naming convention compilers can be distinguished from ordinary applications. The job descriptions are fully abstract and can be brokered to any target system supporting the required Execution Environment and the selected application.

The skeleton of the IDB entry defining the CaR Execution Environment is presented below.

```
<jSDL-u:ExecutionEnvironment>
  <jSDL-u:Name>CaR</jSDL-u:Name>
  <jSDL-u:Version>2.0</jSDL-u:Version>
  <jSDL-u:Description></jSDL-u:Description>
  <jSDL-u:ExecutableName>
    /opt/apps/wrappers/compile-run/CaR_MPI_EXEC_ENV.sh
  </jSDL-u:ExecutableName>
  <jSDL-u:CommandLineTemplate>
    #EXECUTABLE #ARGS -c "#USERCOMMAND #USERARGS"
  </jSDL-u:CommandLineTemplate>
  <jSDL-u:Argument>
    <jSDL-u:Name>MODE</jSDL-u:Name>
    <jSDL-u:IncarnatedValue>-m</jSDL-u:IncarnatedValue>
    <jSDL-u:ArgumentMetadata>
      <jSDL-u:Type>choice</jSDL-u:Type>
      <jSDL-u:Description>Mode</jSDL-u:Description>
      <jSDL-u:ValidValue>compile</jSDL-u:ValidValue>
      <jSDL-u:ValidValue>run</jSDL-u:ValidValue>
      <jSDL-u:ValidValue>compilerun</jSDL-u:ValidValue>
      <jSDL-u:IsMandatory>>true</jSDL-u:IsMandatory>
    </jSDL-u:ArgumentMetadata>
  </jSDL-u:Argument>
  <jSDL-u:Argument>
    <jSDL-u:Name>SRC</jSDL-u:Name>
    ..removed...
  </jSDL-u:Argument>
  <jSDL-u:Argument>
    <jSDL-u:Name>RUN</jSDL-u:Name>
    ..removed...
  </jSDL-u:ArgumentMetadata>
  </jSDL-u:Argument>
  <jSDL-u:Argument>
    <jSDL-u:Name>MPI_IMPL</jSDL-u:Name>
    ..removed...
  </jSDL-u:Argument>
  <jSDL-u:Argument>
    <jSDL-u:Name>MPISTART_ARG</jSDL-u:Name>
    ..removed...
  </jSDL-u:Argument>
  ..OTHER SETTINGS FOLLOW: file export, verbose mode...
</jSDL-u:ExecutionEnvironment>
```

With such approach the client side (i.e. the CaR GridBean) can be replaced by the Generic GridBean. In its main panel the user has to choose and configure a compiler. In the Execution Environment tab the remaining options are configured. Generic GridBean

interface is much less user friendly but it is impossible to overcome this currently as the GridBean API does not expose any feature to manipulate Execution Environments from the GridBean code.

5 Summary

We have presented two solutions which help UNICORE users to remotely build and subsequently execute custom applications. The first of them possess an interface with all intended features. It is easy to use and should be a valuable addition to the UNICORE client-side portfolio from the user perspective.

Unfortunately the server side of this solution has two drawbacks: description of compilers is doubled in the IDB and parts of the compilation scripts are prepared on the client-side, what bounds a job description to a particular target system. We have managed to overcome those issues by the usage of UNICORE Execution Environment feature. This approach can be seen as a perfect choice. However, as the Execution Environment configuration can not be manipulated by a GridBean code, it is not possible to present an equally convenient interface as it is provided by the first solution.

Both solutions are intended to fill the gap between the offering of the generic UNICORE software and highly sophisticated D2E environment targeted at experienced developers. In future we would like to either extend the Execution Environment based solution with a convenient interface (if possible) or to slightly refactor the first solution to minimise its drawbacks, as at least it should be possible to eliminate the binding of job's description to a specific target system.

References

1. UNICORE Middleware website, URL: <http://www.unicore.eu> (16.07.2011).
2. GridBean Developer's Guide, URL: <http://www.unicore.eu/documentation/manuals/unicore6/files/GridbeanDevelopersGuide.pdf> (04.06.2011).
3. G. Fiameni, S. Memon, J. Jimenez, S. H. Leong: DEISA Development Environment — D2E. Proceedings of the UNICORE Summit 2010, IAS Series, Volume 5, ISBN 978-3-89336-661-3.
4. The Eclipse PTP project, URL: <http://www.eclipse.org/ptp/> (16.07.2011).
5. The DEISA project, URL: <http://www.deisa.eu/> (16.07.2011).
6. The Paraver tool website, URL: http://www.bsc.es/plantillaA.php?cat_id=488 (16.07.2011).
7. The MPI-Start project, URL: <http://grid.ifca.es/wiki/Middleware/MpiStart/> (16.07.2011).

Supporting Management of XACML Authorization Policies

Tomasz Królikowski¹, Piotr Bała^{1,2}, and Krzysztof Benedyczak^{1,2}

¹ Nicolaus Copernicus University
Toruń, Poland

² Interdisciplinary Center for Mathematical and Computational Modelling,
University of Warsaw, Warsaw, Poland
E-mail: {tomek242, bala, golbi}@mat.umk.pl

XACML is a powerful and very flexible language for expressing authorization policies. It is used in many industrial solutions and gains an increasing popularity in grid area. It is one of the foundations of the UNICORE middleware authorization stack, XACML engine is also used in Argus authorization service. With great expression power there are also problems. The language is very complex and it is not human friendly being complicated and verbose. As mistakes in policy can render the whole site vulnerable, a special care must be taken, to evaluate and test each update of a site's policy. In this paper we provide an overview of a solution allowing for syntax checking, mock evaluation and debugging of XACML policies. As the tools were inspired by the UNICORE software a special UNICORE profile is provided allowing for checking the most popular constraints which should be most often present in any UNICORE policy.

1 Introduction

Configuring users authorization, especially in large, complicated systems, is a great challenge. Several theoretical concepts were proposed and their practical realisations are used in production. The variety of solutions starts from relatively simple UNIX file system permissions, which offers group and owner based access control and advances through Access Control Lists (ACLs)¹ which provide much more flexible model. Unfortunately ACLs are hard to manage if number of users is high.

The solution which is on one hand extremely flexible and on the another relatively easy to configure in case of high number of users is the policy based authorization based on users' attributes. Users are assigned attributes describing their properties. Attributes can define group membership, roles, institutional affiliation and much more. Attributes can be managed with easy to use tools. Subsequently authorization can be defined in an abstract and universal way as a set of rules generally specifying which attributes a user must possess to perform a given action.

In this paper we discuss popularly used policy language: eXtensible Access Control Policy Language (XACML)². It is a solution of choice for many industrial and academic systems. This work was specifically inspired by the UNICORE grid middleware³ which uses XACML as a basic building block of its authorization stack. While the standard UNICORE authorization policy need not to be frequently modified, any attempt to do so (including changing of the default policy by UNICORE developers themselves) is a very challenging, error-prone and risky operation. It is hard to validate the syntax of the policy. What is much more problematic, the testing of the modified policy can be only performed by invoking grid operations. This is very cumbersome: should be done with different

credentials, it is not obvious which operations should be executed and finally test results may be correct but only because of an incorrect policy. This situation can be detected by a complete set of tests only.

The next section introduces in more details the XACML language. Next, tools designed to help evaluating, testing and debugging XACML policies are presented. These tools are fully generic and applicable for all policies, but we also prepared UNICORE-specific test suite which allows for quick testing of the standard principles of the UNICORE policy. Finally the summary and future work is outlined.

2 The eXtensible Access Control Policy Language

XACML is an extremely flexible policy language. Probably it is the most popular language among the advanced ones. It is enough to mention that both big companies like IBM (in its Tivoli Security Manager product) or Oracle/BEA (in AquaLogic Enterprise Security) and Open Source projects like Fedora Repository are all using XACML. UNICORE is no exception: complex requirements of the grid system are very well fulfilled by the XACML language. Moreover Argus system is providing a more general, remotely accessible authorization service, which is being integrated with different middlewares: UNICORE, ARC and gLite.

XACML is XML based. The specification consists of three parts specifying the syntax of an authorization policy, the syntax of a request which is used to express a query about an authorization decision and finally the request evaluation rules with respect to a defined policy. The request is quite simple: it specifies (using attributes) the resource which is being accessed, who is trying to get access (the subject), what action is going to be performed by the subject and possibly some additional information about an environment where the action is going to take place.

For administrators the most relevant part is of course the syntax (and semantics) of the policy, as the rest is handled automatically by the software.

The thing which we up to now informally described as an XACML policy can be composed out of many „subpolicies” (e.g. authored by different administrators). Therefore the root element of the policy language is called a Policy Set. Each Policy Set can contain other policy sets or regular policies. The evaluation is performed as follows: each policy set element is evaluated and evaluation results are merged using a combining algorithm chosen for the policy set. The list of algorithms is defined in the specification and include algorithms like *deny-overrides* or *permit-overrides*.

The main element of the policy is a list of rules. The policy evaluation is analogous to the Policy Set evaluation: rules are evaluated and a final result is produced using a rule combining algorithm defined for the policy.

Each rule defines its effect (Permit or Deny) and additionally can contain a boolean expression (a so called condition). If the evaluation of the condition succeeds (or if there is no condition at all) then the result of the rule evaluation is set to its effect (however see the next paragraph). Otherwise it is NOT_APPLICABLE.

Last but not least each policy set, policy and rule should possess a target (for policies it is even mandatory). The target specifies for which requests the element applies, by providing required subject, action, resource or environment attributes. If the request is not meeting the target requirements then the element is skipped, or more formally speaking its

validation finishes automatically with a NOT_APPLICABLE result.

To sum up the typical work to create an XACML authorization policy includes:

1. Decision what structure is the most convenient, i.e. whether generally the access is granted or denied. This determines which combining algorithm shall be used.
2. Optionally if the policy is going to be very big and complicated it may be split into several ones and a root policy set must be created.
3. For each policy the target must be defined (often it is just a catch-all target).
4. Finally for each policy its rules, which provide an actual logic of the authorization, must be defined.

The evaluation result can be Permit, Deny, NOT_APPLICABLE (if none of the targets and rules' conditions used in the policy applied) or INDETERMINATE if any error occurred.

2.1 XACML in UNICORE

In order to perform authorization, the UNICORE middleware first authenticates the user and then collects his or her attributes. The gathered attributes and the user's grid name form a list of XACML subject's attributes. A web service which is invoked by the user is a principal resource attribute. Additionally if the accessed web service provides an access to a stateful WSRF⁴ resource, then its identifier is complementing the XACML resource attributes. An invoked web service operation is used as an XACML action attribute.

The UNICORE default policy uses several rules, but the most fundamental one is granting access basing on the *role* attribute. All users possessing this attribute with the value *admin* can perform all operations on any resource. Users who have the *user* role, get access to a rich set of selected operations, allowing for typical grid operations like job submission.

The above description illustrates the XACML usage basics in UNICORE. However there are more challenging issues, which can be solved by means of XACML. As an example let's consider the access granting for the WSRF resource representing a previously submitted job. The possession of the *user* role is not enough as then all *users* would be able to control other users jobs. Therefore UNICORE introduces an additional *owner* attribute, in the XACML resource class. The value of this attribute is determined at runtime and is set to the identity of the WSRF resource creator. Finally the default XACML policy contains a rule that is always allowing the users to access owned resources.

We can consider an even more complicated scenario. Let's assume that a user that had been previously allowed to access a site was banned, i.e. his *role* attribute value was changed to something different than *user* (and *admin*). If the user had created a WSRF resource before he was banned, then he will be able to access the existing resource with the default UNICORE authorization policy. However it is not a big problem to change this behaviour. For instance we can add an another distinguished value of the *role* attribute, e.g. *banned* and a new rule in the policy that explicitly denies access to all clients with this value. There are also few other possibilities.

3 Supporting XACML Policy Authoring

Experience gained with authoring UNICORE authorization policies shows that the most typical problems are:

1. XML syntax errors,
2. semi-syntax errors where formally the policy is correct, but attributes or function names which are used are incorrect,
3. semantic errors, when an update of a policy has side-effects which block standard features of the affected UNICORE server (e.g. owners can't access their resources or clients with the *user* role are not permitted).

The first problem listed above can be solved using any standard XML validation tool, which supports XSD validation. However this requires downloading and maintaining XSD files for XACML language. The two remaining problems are much harder to overcome as it was explained in the introduction.

To assist policy writers we have prepared a generic library of functions. This library is a base of two application: a console one and a web one. Internally the library code is extensively using the XACML processing engine from the HERAS AF project⁵.

Both applications share the following functionality:

- *Validation* feature is based on the standard XML Schema validation of a developed policy. The validation results are formatted to clearly show all syntax error positions.
- *Evaluation* is used to check a result of a policy based on a given request.
- *Debugger* is the most advanced solution, which greatly extends the evaluation feature. Debugger evaluates a provided request and shows not only the final decision but also all intermediate results of each rule. It is also possible to see whether a rule was invoked at all and if it contributed to the final decision (what is not always the case, as it depends on the combining algorithm).

Additionally a UNICORE test suite was created. It contains nine test cases. Each test case is consisting of a authorization request and an expected result of its validation. As it is possible to run a test suite automatically with a single command, UNICORE administrator can easily verify whether all fundamental rules of the UNICORE policy are preserved by a developed policy.

3.1 Console Tool

The console tool provides a simple interface which can be used from a text terminal. The user can invoke the operations described above using simple command line switches. Example of the policy debugging is presented below.

```
> java -jar etXacml.jar -debugger security_policy.xml Request_unicore02.xml
-----
                          V A L I D A T I O N
-----
[INFO] Prepare validation...
```

```
[OK] C:\xacml\examples\policies\security_policy.xml
[OK] C:\xacml\examples\requests\Request_unicore02.xml
```

```
-----
                        E V A L U A T I O N
-----
```

```
[INFO] Prepare evaluation...
[INFO] Decision: PERMIT
```

```
-----
                        D E B U G G I N G
-----
```

```
[INFO] Prepare debugger...
```

```
-----
```

ELEMENT	ID	DECISION	USED
- Policy	- DefaultPolicy	- Permit	- *
- Rule	- UserRule	- NotApplicable	-
- Rule	- Permit:AnyResource_for_its_owner	- NotApplicable	-
- Rule	- Permit:AnyResource_if_owner_is_consignor	- NotApplicable	-
- Rule	- Permit:TargetSystemFactoryService_read-access	- Permit	- **
- Rule	- Permit:TargetSystemFactoryService_for_user	- Permit	-
- Rule	- Permit:Default_SMS_for_user	- NotApplicable	-
- Rule	- Permit:BESManagement_for_user	- NotApplicable	-
- Rule	- Permit:BESFactoryService_for_user	- NotApplicable	-
- Rule	- FinalRule	- Deny	-

```
-----
```

As it can be seen the debugging is preceded with the policy validation and simple evaluation. The debugger output shows the results of all artefacts which make up the whole policy. In the presented case the policy is atomic, as it contains only one Policy element *DefaultPolicy* and nine rules. The result of each is presented in the *DECISION* column. In the last column it is marked whether a rule was used in computing a final policy evaluation result (one star) and whether a rule had a key role for the result (two stars). In the presented example the combining algorithm was *first-applicable* and therefore a first rule which returned Deny or Permit was a key one. It can be noted that the debugger evaluates all defined rules, while in the real evaluation the XACML engine may stop processing the rules, when the final answer can be determined (in this case just after evaluating the rule marked with two stars).

3.2 Web Interface

The web interface allows to use the tools without installing or downloading any additional software, using a web-browser. A screen-shot of the policy debugging is presented in Figure 1.

The Web-based tool requires the user to register and login. This step was added as users can store their tested artefacts within their web account. It is possible to store policies, test requests and meta-tests which are described later.

The login process is based on the OpenID protocol⁶. The usage of OpenID eliminates the need for extra development of account management subsystem and what is probably more important, the users who already have OpenID account can use their existing credentials.

The unique feature of a web based tool is its ability to define meta tests. Meta tests are user defined test suites containing a set of requests and their respective valid results. Meta

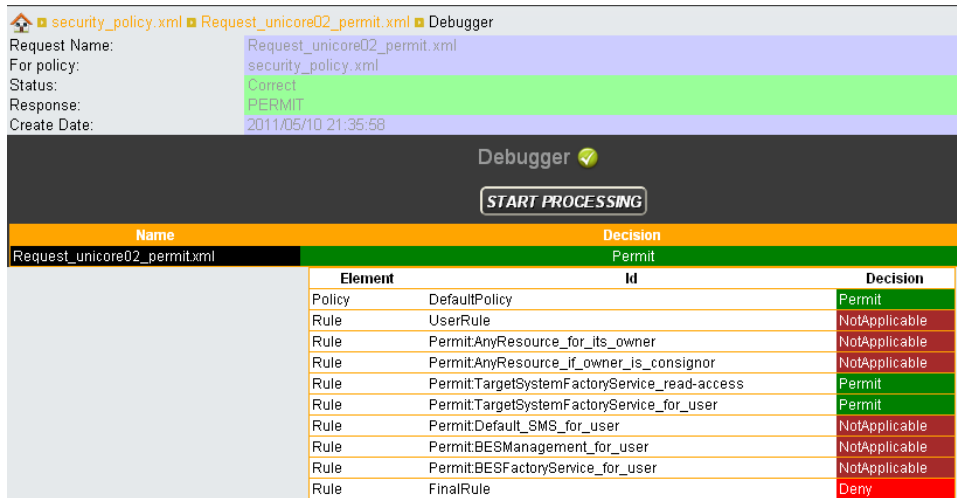


Figure 1. Fragment of a screenshot presenting results of policy debugging. Result of each policy rule is shown after evaluation with a given test request.

test can be applied to any policy to perform a complete and quick validation of its several properties. A careful reader can notice that UNICORE test suite which was previously described is a special case of a meta test.

4 Summary and Future Work

The presented tools simplify extremely complicated XACML policy authoring. The especially important one is the debugger allowing for a fine grained testing of all policy rules. It is impossible to gather such a detailed information by performing tests with other, standard XACML evaluation tools. What is more important, the meta-tests provide a possibility to create test suites for automated and complete policy testing.

While policy authoring is significantly improved by the proposed tools there is still place to make a progress. We can point out two directions. The first one is related to creation of the test policy requests. The XACML requests are written in XML and have a quite verbose syntax on their own. It would be good, to be able to specify them in a simplified way. This development seems to be a quite natural next step for the presented tool.

The more ambitious goal is be to provide a XACML graphical editor. Such editor, preferably based on a rich client platform like Eclipse RCP⁷, can be integrated with the tools described in this paper. Additionally it should allow for quick selection of XACML constructs, provide templates, support various XACML profiles and present a semantic view of the policy. Currently there is known, open source XACML editor⁸ but is fairly simplistic and offers only basic functionality.

Eventually support for the upcoming XACML 3.0 revision should be provided. However such effort is of lower priority as the new version of the language should be adopted first by services.

References

1. Linux POSIX Access Control Lists, URL: <http://www.suse.de/~agruen/acl/linux-acls/online/> (09.07.2011).
2. OASIS XACML Committee web page, URL: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml (3.05.2011).
3. B. Schuller: General Introduction to UNICORE, URL: http://www.unicore.eu/documentation/presentations/unicore6/files/General_Introduction_to_UNICORE.pdf (10.05.2011).
4. S. Graham, A. Karmarkar, J. Mischkin, I. Robinson, I. Sedukhin (eds.): Web Services Resource 1.2 (WS-Resource), OASIS Standard, 1 April 2006, URL: http://docs.oasis-open.org/wsrp/wsrp-ws_resource-1.2-spec-os.pdf (20.07.2011).
5. Holistic Enterprise-Ready Application Security Architecture Framework, URL: <http://www.herasaf.org/> (09.07.2011).
6. The OpenID specifications, URL: <http://openid.net/developers/specs/> (09.07.2011).
7. Eclipse RCP, URL: <http://www.eclipse.org/> (09.07.2011).
8. UMU XACML Editor, URL: <http://sourceforge.net/projects/umu-xacmleditor> (09.07.2011).

Distributed Storage Management Service in UNICORE

Tomasz Rękawek^{1,2}, Piotr Bała^{1,2}, and Krzysztof Benedyczak^{1,2}

¹ Interdisciplinary Center for Mathematical and Computational Modelling,
University of Warsaw, Warsaw, Poland

² Nicolaus Copernicus University
Toruń, Poland

E-mail: {newton, bala, golbi}@mat.umk.pl

In this paper we are presenting a UNICORE Distributed Storage Management Service (DSMS). The system provides a native UNICORE solution for exposing storage resources distributed between different UNICORE sites as a single logical space. Such a feature is highly desired by end-users, allowing them to forget about the actual location of their files. Additionally, the system allows for a better Workflow System experience as a larger storage space does not limit the size of files used in user workflow jobs. The paper also compares the DSMS system to the other existing solutions, both available in UNICORE and other Grid middlewares.

1 Introduction

The experience with Grid deployment shows that storage resources are similarly important as compute resources. The UNICORE middleware provides possibility to uniformly expose distributed compute resources but in the case of distributed storage either separate solutions must be deployed and then integrated or the system is limited to use separate storage resources independently.

There are two possible approaches which can mitigate this situation. The first one is to implement a new solution from scratch, integrating it as good as possible with the existing UNICORE stack. Another option is to provide a UNICORE interface to another existing solution from a different Grid middleware. The later approach brings all the problems of different security mechanisms and can be painful for administrators as they have to install and maintain an additional (usually very complex) software. Furthermore, it is highly possible that despite of significant effort the result might have some other limitations resulting from different middleware paradigms. As we believe that the cost of the second approach is higher than the implementation of a dedicated UNICORE solution we have decided to implement the first approach.

The next two sections of this paper present existing storage capabilities in UNICORE and other grid middlewares. Section 4 precisely defines the project aims. Next, the DSMS system architecture is described along with explanation of its idea and realisation. The subsequent section 6 discuss the synchronisation issues and respective solutions implemented to overcome them. Finally the comparison with other systems is given with a description of the current and future work.

2 Existing Storage Capabilities in UNICORE

UNICORE Storage Management Service (SMS) provides an abstract file system-like view on a storage resource. It has common POSIX-like operations (e.g. `mkdir`, `delete`,

`listDirectory`) and a few more to initiate file transfers¹. The standard UNICORE distribution includes the following SMS implementations:

- **FixedStorage** — stores files in a directory defined in a configuration,
- **HomeStorage** — stores files in a user's home directory,
- **PathedStorage** — a destination path is resolved at runtime so environment variables can be used to parameterise it (e.g. `/opt/uni-storages/$JOB`).

Unfortunately none of these implementations allows client to distribute files throughout the Grid. There are, however, three solutions that extend standard storage capabilities: UniRODS, UniHadoop and Chemomomentum Data Management System.

UniRODS²³ is a Storage Management Service implementation providing access to iRODS⁴ distributed data storage system. iRODS provides a shared disk space consisting of multiple physical storages. Metadata and storage information are persisted in iCAT service based on PostgreSQL, MySQL or Oracle database engines. Advantage of the UniRODS is the usage of the standard UNICORE Storage Management Service interface, so the client software can work with it without any modifications. UniRODS is also very easy in installation, kept simple and integrates well into the internal UNICORE/X architecture.

Because authorization mechanisms in UNICORE and iRODS are completely different, UniRODS has to use static files to map users. Another disadvantage is the necessity to install and maintain yet another, complicated service on Grid nodes.

UniHadoop⁵ is a similar solution to UniRODS, but uses a different backend — Apache Hadoop⁶. Hadoop is an open-source implementation of Google File System⁷ (GFS). GFS is optimised for using a large number of cheap disk servers, therefore replication and monitoring are priorities for the system. Files stored in GFS are split into fixed size chunks which are distributed into *chunk servers*. Each chunk has 64 MB and is replicated on three servers. UniHadoop, like UniRODS, exposes the Storage Management Service interface and is available in the standard UNICORE distribution. An additional advantage is that Hadoop uses system user name for authorization, so it's possible to apply standard Storage Management security model in which Grid user (identified by his X.509 distinguished name) is mapped to the system user (identified by her login). This mapping is done by attribute source service existing in a Grid (e.g. XUADB or UVOS). Unfortunately, with UniHadoop there is again a requisite for an additional complex system to maintain — Apache Hadoop. Moreover, the Apache Hadoop internal security is not yet matured and misses features which are crucial for the Grid. There is no support for transport level encryption and proposed authentication solutions are going to be based on Kerberos which is not the best option for X.509 certificates based UNICORE.

The Chemomomentum Data Management System⁸ was designed as a native UNICORE distributed file storage with support for advanced metadata, ontologies and support for retrieving data for external storages like scientific databases available on-line. The system is comprised of a single entry point service called *Data Management System Access Service* and several back-end services providing the actual functionality. The physical storage of files is performed using the standard UNICORE SMS service.

The Chemomomentum DMS is a powerful system but bears a significant amount of problems for its potential users:

- The single service being an entry point to the whole system is a bottleneck.

- The whole system is complex and therefore hard to maintain. It seems that support for the system was dropped and it is not updated to the newest UNICORE releases. The effort to perform such an update can be assessed as quite high as there are many services, some of them require a domain-specific knowledge (like external database access tools).
- The system requires extensions on the client side. This is especially hard for UNICORE users as they have to install the extension and for developers who have to maintain plugins for each UNICORE client.

3 Data Management Solutions in Other Grid Middlewares

Beside the UNICORE world, data management solutions are usually called Storage Resource Managers (SRM) what is at the same time a name of the popular SRM protocol⁹ used to access a storage. In this section the four most popular managers are shortly introduced.

StoRM¹⁰ is a simple implementation of SRM. It exposes a selected POSIX file system. If the distribution is needed, then a distributed file system (like IBM GPFS) has to be used. Authorization is done by the POSIX Access Control List and external mapping (from Grid users to the system login) is necessary. StoRM uses an internal database for storing metadata and file locations. StoRM has also plug-in mechanism which can be used to utilise additional capabilities of the file system¹⁰.

Disk Pool Manager (DPM) is a *lightweight solution for disk storage management*¹¹. Despite its advertised „lightweightness”, DPM is a more complex system than StoRM. Usual installation is distributed into two type of servers: *head node* and *disk node*. Head node manages metadata and is an entry point for clients. Disk nodes store the physical data. The following services run on the head node:

- **DPM nameserver daemon** — manages metadata and directory structure,
- **SRM daemon** — a client entry point with the SRM interface implementation,
- **DPM** — the main daemon which executes requests.

These services do not have to be on the same server, but can not be replicated.

dCache¹² is a much more sophisticated solution than StoRM and DPM. Additional to disk storage, it supports tertiary memory (usually magnetic tapes). That kind of memory is very slow but cheap. dCache has mechanisms to copy (cache) recently used files from tapes to the disks. It also can recognise very often opened files (so called *hot spots*) and replicate them throughout the disk servers in order to balance the load. There are three types of servers in dCache:

- **head node** — unique node that runs the database, name server and other not replicable services,
- **door nodes** — entry points for clients (can be replicated),
- **pool nodes** store the data.

dCache's name server is called Chimera. It uses PostgreSQL database for storing files metadata and directory structure.

CERN Advanced STORage manager¹³ (CASTOR) is similar to dCache. It also supports tertiary storage and uses disks as a cache memory. CASTOR consists of many stateless daemons querying the central database and such architecture increases scalability. The most important components are:

- **request handler** is a lightweight service adding new requests to the database,
- **request scheduler** schedules requests for execution, it is possible to use an external scheduler such as LSF or Maui Cluster Scheduler,
- **Stager** is executing clients' requests,
- **resource monitoring daemon** gathers information for the scheduler,
- **name server** stores directory structure,
- **Distributed Logging Facility** is responsible for logging.

The SRM resources are able to provide distributed storage, however distribution is practically limited to the hosts at one site. The Logical File Catalogue (LFC) service¹⁴, available in gLite middleware, provides a capability to maintain the global file space. The LFC uses an Oracle or MySQL database and exposes a proprietary interface to control its contents. The LFC also implements an internal authorization system. It is integrated with other gLite tools and serves as one of the principle components of the Worldwide LHC Computing Grid.

4 The Aim of the Project

Purpose of the system presented here is to combine multiple UNICORE storage resources into a single logical resource. This logical space should be usable by both Grid client software directly and Grid services (for instance by the UNICORE Workflow Service). Users should not have to know where their files are stored physically. Optionally advanced users should have an opportunity to be aware of the distributed nature of the storage and physical files locations. As it can be learnt from the Chemomomentum DMS lesson, the system should not be too complex — otherwise its future maintenance would be problematic.

The requirement which can be directly derived from the above paragraph is that the standard and existing UNICORE client software should be able to access and operate on the newly created solution. Therefore the interface to the distributed file space must be the UNICORE Storage Management Service. The service should not limit the standard capabilities of the Storage Management Service as for example supported file transfer protocols.

To achieve high performance the system should be ready for providing an arbitrary number of entry points. Additionally file transfers must be performed directly between source and destination storages, without employing any proxy.

Last but not least the system must use the standard UNICORE authorization and authentication mechanisms to enable smooth and natural integration with the whole middleware.

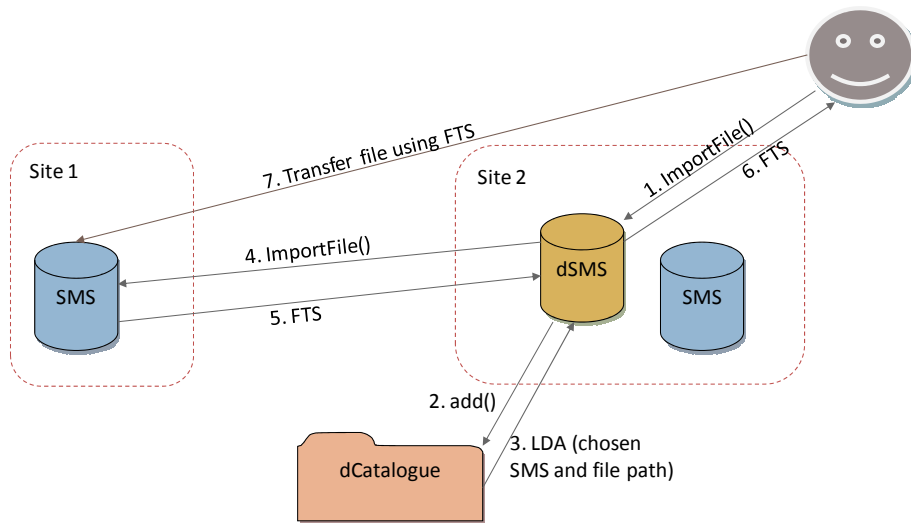


Figure 1. `ImportFile` operation invoked on DSMS.

5 DSMS Approach

DSMS consists of following components:

- Data is stored on existing, standard Storage Management Services.
- File locations and directory structure is stored on a service called **dCatalogue**.
- Client entry point is a special Storage Management Service implementation called **DSMS**.

The presented system can use an unlimited number of storages, which are available from a UNICORE Registry. Also the number of entry points is unlimited. **dCatalogue** is a central service. It is storing only a minimal amount of information about the files to minimise synchronisation and performance issues. **DSMS** despite of exposing Storage Management interface also connects to the **dCatalogue** and physical storages.

The overall idea is to mask the distributed nature of the system with the standard Storage Management implementation. We will show the way we have achieved this using two representative examples of operations. Figure 1 presents the `ImportFile` operation invoked on **DSMS**.

`ImportFile` operation allows a user to add a new file to the storage. At the beginning the user connects to the **DSMS** using a standard UNICORE client. The client invokes function `ImportFile` (step 1) and waits for an endpoint reference of File Transfer Service (FTS). **DSMS** connects to **dCatalogue** (2) asking for a physical SMS where the file should be saved. **dCatalogue** chooses one SMS out of those existing in the Grid (using a special algorithm) and returns its `EPR`^a and physical file name (3). This pair - SMS `EPR`

^a`EPR` stands for an *Endpoint Reference Address*, which precisely defines a network address of a UNICORE service and a concrete resource accessible through this service.

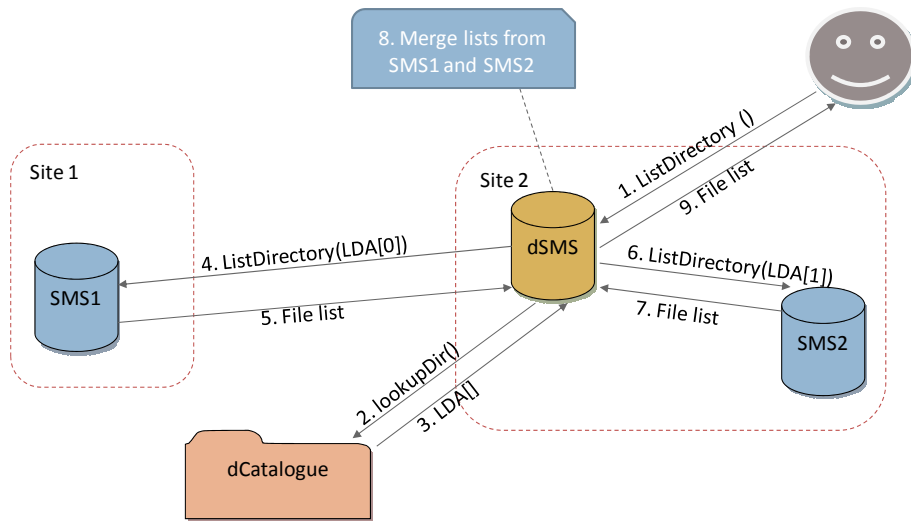


Figure 2. ListDirectory operation invoked on DSMS.

and file path is called Location Dependent Address. DSMS connects to the received SMS (4) calling an `ImportFile` operation. SMS returns FTS EPR (5) which is returned to the client (6). Further communication is performed as usual: the client sends the file using the received EPR (7). Operations 2 and 4 are done on behalf of the client using trust delegation mechanism and hence the client is an owner of the file on the SMS. The user connecting directly to the SMS has access only to his or her files. dCatalogue authorise all operations using the client's X.509 distinguished name.

A `ListDirectory` operation presented in the figure 2 works slightly different. Although the directory structure is stored in dCatalogue, file metadata (such as size, privileges, etc.) has to be retrieved from appropriate storages. So DSMS (in step 2) invokes the `lookupDir` operation and gets the list of storages which contains files from the given directory (3). After that DSMS connects to each storage and invokes the `LookupDirectory` operation (4–7). When all lists are collected they are merged (8) and sent to the user (9).

The Storage Factory is a service introduced with UNICORE 6.3. It can create separate SMS instances for each workflow job (other use cases are also possible). An advantage of the Storage Factory is that it can remove all files created by a workflow job by removing the whole SMS instance. The Factory can be configured to create a new DSMS within a separate namespace. UNICORE Storage Factories can be also used by the system to dynamically create instances of DSMS service. A DSMS which was created using the Storage Factory can subsequently create new physical SMSes and store files in them.

Of course usage of the Storage Factory is not limited to the UNICORE Workflow System: it can be used for any other application which can take advantage of a dynamic creation of an SMS instance.

6 Handling Errors and Synchronization Issues

Synchronization between the dCatalogue database and a real file state in the SMS resource is a very important issue. There are two main possible causes of losing this synchronization:

- A) a file, which does not exist in dCatalogue, appears in an SMS resource,
- B) a file which belongs to DSMS is removed from SMS, but not from dCatalogue.

DSMS operations which could suffer from the above problems can additionally be divided into three main groups:

1. directory listing operations (methods `ListDirectory` and `Find`),
2. operations which add new file (methods `ImportFile`, `ReceiveFile`, `Copy`, `Rename`),
3. operations that get file or file information (methods `ExportFile`, `SendFile`, `ListProperties`, `ChangePermissions`, `Copy`, `Rename`).

During `ListDirectory` and `Find` methods DSMS gets file lists from each used SMS, merges them and sends back to the client, so there is no synchronization problem possible here: even if data in dCatalogue is invalid the client will not notice it.

If we consider operations which add a new file, synchronization problem A) may appear if a file with a given name exists on an SMS (but not in dCatalogue) and user tries to send that file again in order to replace it. To avoid A) the DSMS checks each used SMS looking for the file with the given name. If the file exists, it will be added to the dCatalogue and replaced by the new file. Synchronization problem B) can not appear with this kind of operation.

The last group of operations are these which read file information. If a user wants to fetch a file which does not exist in the dCatalogue, then before throwing the `FileNotFoundException` exception, that file will be searched in each SMS. If it is found then it will be added to the dCatalogue and returned to the user (and the A) problem is resolved). If the file exists in the dCatalogue then DSMS also checks if the file exists in the proper SMS. If not, file will be removed from the dCatalogue and the user gets the mentioned exception.

It must be noted that under standard circumstances none of the synchronization issues should appear. To cause A type of problem, a user must use the physical SMS directly (what can be made difficult by administrators, by hiding the SMS from the Registry) or copy the files with other tools like FTP or SCP. Additionally the files must be placed in a special, hidden folder created by the DSMS. The situation with the B type of issue is analogous. Because of this fact and the performance penalty introduced by the above described auto-synchronization methods, they can be disabled in the DSMS configuration.

7 Summary

The DSMS can be seen as a direct successor of the Chemomentum DMS: it is a native UNICORE solution. In comparison with its predecessor the DSMS is more lightweight as

it does not provide some of the more advanced features (metadata, support for ontologies and external databases). Additionally DSMS removes the original bottleneck of the system and necessity for usage of a client's extension. Therefore we hope that DSMS will be much easier to maintain.

Compared to UniHadoop and UniRODS, DSMS does not require installation of an additional storage system and does not introduce security mismatches between UNICORE and on other system. The DSMS is also quite different than typical SRM solutions: it is designed to support storage which is distributed between different sites, while solutions like DPM are designed to support disks exposed by different machines located in a single administration domain. DSMS exploits arbitrary UNICORE SMS implementations. It may be even used to merge Hadoop SMS and iRODS SMS as a single logical space. Therefore DSMS can be seen more as a LFC counterpart. However LFC is designed differently as it stores more data about files centrally, what brings more complicated synchronization issues. Also the LFC requires usage of special operations to manage files in its context.

At the current project stage all basic requirements for the distributed storage are fulfilled. The system was recently integrated into the standard UNICORE distribution as an experimental type of storage. It can be seen as a perfect solution for a Grid system comprising of several sites, where each of them is providing their own storage resource.

Our current work is concentrated on measuring the performance of the system. We are paying a special attention to an influence of the automatic dCatalogue synchronization feature on the overall performance. Of course system scalability is another crucial aspect that must be carefully determined. In the future more advanced strategies for choosing the optimal physical storage resource are planned. Currently a round-robin strategy is the default one and another one which is available chooses the storage resource with the most free space.

References

1. Schuller, B.: General Introduction to UNICORE, URL: http://www.unicore.eu/documentation/presentations/unicore6/files/General_Introduction_to_UNICORE.pdf (10.05.2011).
2. Derc, K.: *Integracja systemu iRODS ze środowiskiem gridowym UNICORE 6*, Master's thesis, NCU Toruń 2008.
3. Janiszewski, K.: *Rozproszone bazy danych w systemach gridowych*, Master's thesis, NCU Toruń 2010.
4. The iRODS project, <https://www.irods.org> (16.05.2011).
5. Bari, W., Memon, A. S., Schuller, B.: Enhancing UNICORE Storage Management using Hadoop Distributed File System. *Euro-Par 2009 Parallel Processing Workshops*, LNCS, vol. 6043, Springer, Heidelberg 2010.
6. The Apache Hadoop project, <http://hadoop.apache.org> (16.05.2011).
7. Ghemawat S., Gobiuff H., Leung S.-T.: The Google file system. *Proceedings of the 19th ACM Symposium on Operating Systems Principles 2003, SOSP 2003, Bolton Landing, NY, USA, October 19-22, 2003*. ACM 2003.
8. Rasch, K., Schöne, R., Ostropytsky, V., Mix, H., Romberg, M.: The Chemomentum Data Services — A Flexible Solution for Data Handling in UNICORE. *Euro-Par 2008 Workshops - Parallel Processing*, pp. 84–93, Springer-Verlag, Berlin, Heidelberg

2009.

9. Sim, A., Shoshani, A. et al. (eds.): The Storage Resource Manager Interface Specification Version 2.2. OGF 2008.
10. Corso E., Cozzini S., Forti A., Ghiselli A., Magnoni L., Messina A., Nobile A., Terpin A., Vagnoni V., Zappi R.: StoRM: A SRM solution on disk based storage system. *Proceedings of the Cracow Grid Workshop 2006*, Kraków 2006.
11. Grid Data Management — DPM, URL: <https://svnweb.cern.ch/trac/lcgdm/wiki/Dpm> (19.02.2011).
12. The dCache Book 1.9.5, URL: <http://www.dcache.org/manuals/Book-1.9.5/Book-a4.pdf> (25.02.2011).
13. The CASTOR project, <http://castor.web.cern.ch/castor> (16.05.2011).
14. The LFC webpage, <https://svnweb.cern.ch/trac/lcgdm/wiki/Lfc> (16.05.2011).

UFTP: High-Performance Data Transfer for UNICORE

Bernd Schuller and Tim Pohlmann

Jülich Supercomputing Centre,
Research Centre Jülich, Jülich, Germany
E-mail: {b.schuller,t.pohlmann}@fz-juelich.de

The UNICORE middleware is increasingly used for data-intensive applications where large data volumes need to be transferred at adequate data rates. UNICORE's default built-in file transfer tools (HTTPS and OGSA ByteIO) are very convenient to use, but lack the required throughput for large (i.e. on the order of gigabytes) data files. So far, UNICORE is lacking a convenient, fully integrated high-performance file transfer solution.

A new file transfer tool called UFTP was developed and integrated in the UNICORE stack. The UFTP server requires only a single open port in the firewall. Similar to GridFTP, UFTP uses the available bandwidth more efficiently by employing multiple TCP data connections. Both the UFTP server and the UFTP client are implemented in Java, guaranteeing platform independence, simple deployment and secure operation. The UFTP client can be integrated easily in the existing UNICORE clients.

This paper presents the architecture of UFTP and its integration into UNICORE. The UFTP server (uftp) and its deployment are described. We give a performance comparison in several typical scenarios, comparing UFTP to UNICORE's default HTTPS-based mechanism.

1 Introduction

Clearly, efficient handling of data is an important ingredient for the performance of most distributed computing applications. The standard job processing model used in UNICORE requires that any required input data is staged into the job working directory before the job is started, and result data is staged out to a permanent storage. It turns out that in many cases the performance and the job throughput is dominated by the time required for data movement.

Many application scenarios of distributed computing middleware are data-oriented, and the volumes of data that are produced and need to be moved over the network keep increasing. Thus there is a requirement for high-performance data transfer, that can efficiently use available network bandwidth. This transfer protocols and tools should be seamlessly integrated with UNICORE. Especially integration with the UNICORE clients should be in accordance with established UNICORE design goals such as platform independence and ease of use.

So far, high-performance data transfer was difficult to achieve in UNICORE. The default BFT protocol uses the HTTPS channels between client, gateway and the UNICORE/X server. Thus, it requires multiple socket connections to get the data from the client to the file system. For example, in a data upload from client to server, the data is first sent to the gateway and from there to the UNICORE/X server. To write the data to the file system, the UNICORE/X server needs to open a third socket connection to the TSI. Under favourable circumstances, the BFT transfer achieves a typical performance of several megabytes per second.

The other built-in mechanism is OGSA ByteIO¹. UNICORE supports only the “simple” transfer mechanism, where the transferred data are not directly streamed via TCP, but Base64 encoded into the SOAP messages. The performance is even worse than in the BFT case due to the higher overhead due to XML parsing and Base64 decoding. Typically, ByteIO performance is below one megabyte per second.

1.1 Requirements on a Data Transfer Solution

To achieve high-performance, direct transfers from client to a process running on the cluster login node is required. However, this is generally not possible due to the site firewall. If one wants to avoid statically configured open ports in the firewall, a solution for dynamic firewall transversal is required.

Furthermore, the data transfer protocol has to cope with the performance problems encountered when using TCP connections in the presence of packet loss². For example, multiple TCP streams may be used, or alternatives to TCP might be explored.

UNICORE has two modes of data transfer, both of which must be supported by the data transfer solution. On the one hand data needs to be moved between client and server, on the other hand data staging is done between two UNICORE sites. For the client to server transfer, good integration into the UNICORE clients is required. For the server to server transfer, the data transfer process should be running on the TSI node (cluster login node), where the required file systems can be accessed directly.

Last not least, the solution should be easy to deploy, configure and operate in a secure manner.

1.2 Related Work

Two approaches for achieving high-performance data transfer in UNICORE were available so far: GridFTP and UDT. Both have specific drawbacks, and fail to fully solve the problems.

The first option is to use the well-known GridFTP³ from the Globus Toolkit. Unfortunately this integrates rather badly with UNICORE, for multiple reasons. Since GridFTP uses proxy certificates for authentication, it is not compatible with the UNICORE security model. Furthermore GridFTP requires open port ranges in the site’s firewall, which is widely regarded as a serious security flaw. Most importantly, GridFTP cannot be readily integrated into UNICORE clients since no native Java implementation is available. Therefore, using GridFTP violates the core UNICORE principle of platform independence. GridFTP can currently only be used for data staging, not for general UNICORE data movement from client to server or from server to server, and it requires special client setup so that the required proxy certificate is generated.

The second option, UDT (UDP based data transfer)⁴, is a protocol specifically designed to overcome the performance problems of TCP by implementing a custom layer on top of the UDP transport protocol. UDT has its own algorithms for dealing with network congestion. The reference implementation is written in C++. The performance of UDT is outstanding, clearly outperforming plain TCP even when multiple TCP streams are used. Unfortunately, the UDT reference implementation is not platform-independent, and full integration on both server and client side is difficult. A native Java implementation of UDT

has been created⁵, however the performance and stability are not yet satisfactory. However, it may be that these issues can be solved in the future. A UDT integration module⁶ is available in UNICORE, which provides server-server transfers. Firewall transversal is provided using a technique called UDP hole punching using the UNICORE/X server as a relaying party. Summarising, UDT is very promising due to its superior performance, but deployment of the C++ variant of the server may be non-trivial, and full client integration has not been done due to the performance and stability issues of the Java UDT implementation.

1.3 Structure of the Paper

The remainder of this paper is structured as follows. In the next section, we present the UFTP architecture, its features and how it is integrated into UNICORE. Then, some performance examples are presented, showing how UFTP compares to the default BFT mechanism and to the “scp” data transfer. A summary and an outlook concludes the paper.

2 The UFTP Data Transfer Protocol and its Integration into UNICORE

The central idea of UFTP is to leverage the well-known FTP protocol⁷ for negotiating new data connections between a client and a server. An FTP server listens on a port which clients can connect to. FTP is plain text, enabling a firewall to monitor the traffic and dynamically alter its configuration to allow the required data connections between client and server. However, UFTP does not use the login procedure of the FTP protocol, instead it relies on its own authentication scheme. Thus we say that UFTP is “pseudo-FTP”. The FTP protocol is used only to communicate the IP addresses and ports to be used as data connections.

The firewall needs to be configured in order to recognise a certain port as an FTP port. For the *iptables*⁸ solution which is often used on Linux servers, this feature is called “connection tracking”. We have also tested UFTP in conjunction with a commercial firewall (CISCO).

The UFTP library includes a server daemon, named `uftpd`, and client code. The library is written in Java, version 1.6 (from Oracle, OpenJDK, or IBM) is required. The `uftpd` server is deployed on a machine which has access to the required file systems, usually this will be the cluster login node where the UNICORE TSI daemon is running. The pseudo-FTP port, named *listen port* has to be accessible from outside the firewall.

For authentication, the `uftpd` server listens on a secure second port, the *command port*, which is not accessible to clients. The command port can be protected using SSL, which is mandatory in a production deployment for security reasons.

The command port is used by a trusted party (such as the UNICORE/X server) to announce an upcoming data transfer. Such a transfer request contains the following information:

- the client’s IP address
- the file name
- the mode (whether to send or receive data)

- the requested number of parallel data streams
- Unix user and group to be used for accessing the file
- an authentication secret (a string, usually a 128 bit GUID is used)
- (optional) a 64 bit key to be used for data encryption
- (if receiving data) whether to append to an existing file

This information is available to the UNICORE/X server when the file transfer is created. For testing purposes, a UFTP transfer can also be initiated using a tool such as telnet or netcat, provided the command channel is not secured with SSL.

After a transfer request has been sent to `uftp`, a client connection to the *listen port* of `uftp` is accepted, provided several conditions are met. The client's IP address and the provided secret are checked and matched against the existing transfer requests. This is necessary since multiple clients may share the same IP address. If no matching client connects in a time span of five minutes after the request is received by `uftp`, the request is removed.

After authentication is completed, client and server negotiate the parallel data connections using the FTP protocol. The server may limit the number of streams. After all the data sockets have been created, the actual data transfer is started. Here, the data can be encrypted, if the client so desires. Encryption is done using a symmetric algorithm, and as expected it entails a significant performance drop.

In production, `uftp` will need to run with superuser privileges, since it has to read or write files for arbitrary users. To prevent unauthorised access, the effective user and group is switched to the requested values before accessing files. Attempts to access files as root are rejected by `uftp`.

2.1 Integration into UCC and UNICORE/X

Apart from the `uftp` daemon deployed on the TSI node, integration code is part of the UNICORE/X server and UCC since version 6.4.1.

On the UNICORE/X server, the “UFTP” protocol and file transfer service has to be enabled, together with configuration of the `uftp` host and the control and listen port numbers. Full configuration details are available in the UFTP documentation⁹. It is important to note that when the UNICORE/X server acts as UFTP client (e.g. when doing data staging), the actual UFTP client process is started on the TSI node, where it has direct access to the file system.

For the UCC client, no configuration is required. However the user may wish to specify the number of streams (which is 2 by default) and whether data should be encrypted. It is also possible to explicitly control which client IP address is communicated to the server, which can be useful in network configurations such as NAT or VPN.

For better understanding, we want to spell out how the file transfer procedure works in case of a file import (i.e. data upload).

1. The UCC requests the creation of a new file import resource by invoking the `Import()` method on a storage. The file path, protocol, and the required extra parameters (client IP, number of streams, encryption yes/no) are given as parameters.

2. The UNICORE/X server creates the new resource. It contacts the UFTP server via the command socket, and sends a transfer request containing all the required parameters. If encryption is requested, a key is created and included in the transfer request. The resource properties of the new file transfer contain the information required by the client, such as the UFTP server's listen address, the number of streams and the encryption key. The address of the new transfer resource is sent back to the client.
3. The UFTP client code in UCC accesses the WSRF resource properties of the file transfer resource, and retrieves the parameters.
4. The UFTP client contacts the UFTP server and performs the data transfer.

Of course, all this is completely transparent for the user, and she only needs to specify that UFTP shall be used for the transfer.

3 Performance Examples

We have evaluated UFTP performance in two scenarios. Both are somewhat artificial, but since UFTP is a new tool, realistic deployments were not yet available.

The first scenario is a purely local setup with UNICORE Gateway, UNICORE/X, TSI and UCC all running on the same machine, a Lenovo laptop with 4GB main memory and an Intel Core2 Duo CPU (P8600) running at 2.40GHz.

We compared UFTP file uploads (using UCC's `put-file` command) with the BFT transfer mechanism while varying file size from 1 to 400 Megabytes. We used 2 parallel streams. Both encrypted and unencrypted UFTP was tested.

As Figure 1 shows, UFTP outperforms BFT by a very wide margin, which is smaller for small files, since the overhead of setting up and starting a UFTP transfer is higher than in the BFT case. As expected, UFTP data encryption entails a large performance hit of about 75%.

As a second scenario, client-server transfers were tested on two server machines. The servers were AMD Opterons (246 model) running at 2GHz. The servers are connected by a local network. The network performance was estimated by copying data using *netcat*¹⁰ to be 78 megabytes/sec. Data transfer was from local disks. In this case we compared the performance of UFTP to `scp` for file sizes ranging from 1 MB to 2000 MB. The results are shown in Figure 2. The `scp` performance levels at around 30MB/sec, and outperforms the encrypted UFTP by a factor of 2. However, plain UFTP outperforms `scp` by a factor of about 2, reaching up to 70MB/sec.

Further performance evaluations are required, especially in realistic wide-area networks. However, we can already say that UFTP outperforms the default UNICORE file transfers by a wide margin, and is (in the unencrypted case) much better than the widely used `scp`.

4 Summary and Outlook

We have presented a new file transfer protocol, UFTP, and its integration into UNICORE. UFTP solves the performance and deployment issues associated with file transfer in

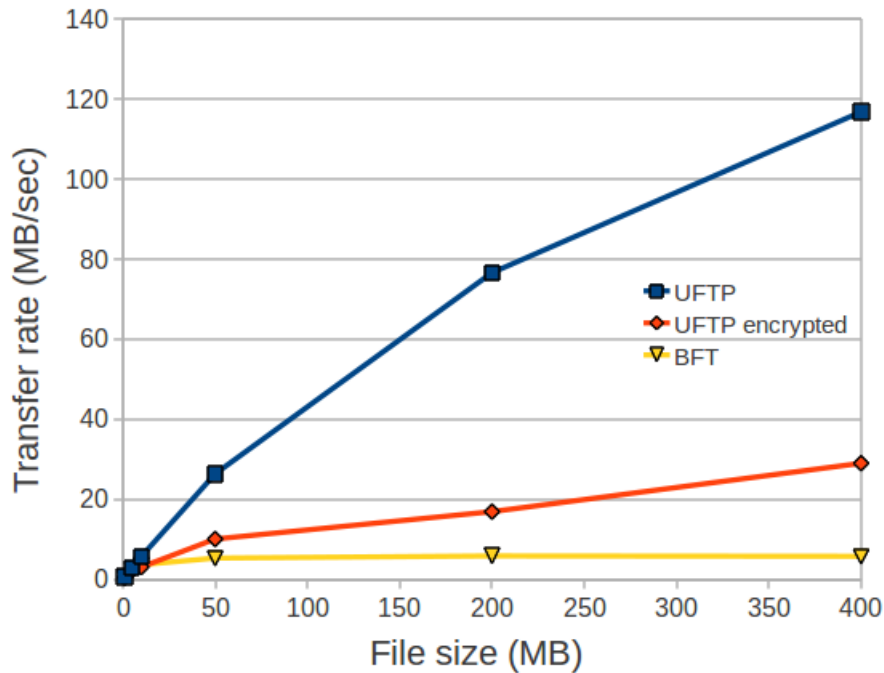


Figure 1. UFTP performance comparison on localhost.

a secure and easy-to-use fashion. UFTP offers superior performance over the default UNICORE protocols BFT and ByteIO. Each file transfer can use multiple TCP streams, optionally the data streams can be encrypted. Written in Java, UFTP is platform independent and easy to deploy. It is firewall friendly, requiring only one single open port, and allowing firewalls to monitor and log the data connections.

The first release UFTP 1.0 can be downloaded from SourceForge¹¹, and the UNICORE/X server and the UCC client in version 6.4.1 with UFTP support are available. The UNICORE rich client will offer UFTP support as well with its 6.4.1 release.

With UFTP deployed in production, some enhancements will certainly be necessary. For example, many hosts have multiple network interfaces accessing different sub-nets with different performance characteristics whereas UFTP currently only supports a single listen address. Another requirement will be to be able to deploy multiple UFTP servers on multiple login nodes for load balancing and failover.

In general, the data area remains a challenge for future enhancements of UNICORE, driving its further evolution as a universal tool for distributed applications. For example, reliable file transfer with success notification is not yet available. We are confident that UFTP will play an important role in future UNICORE installations and application scenarios.

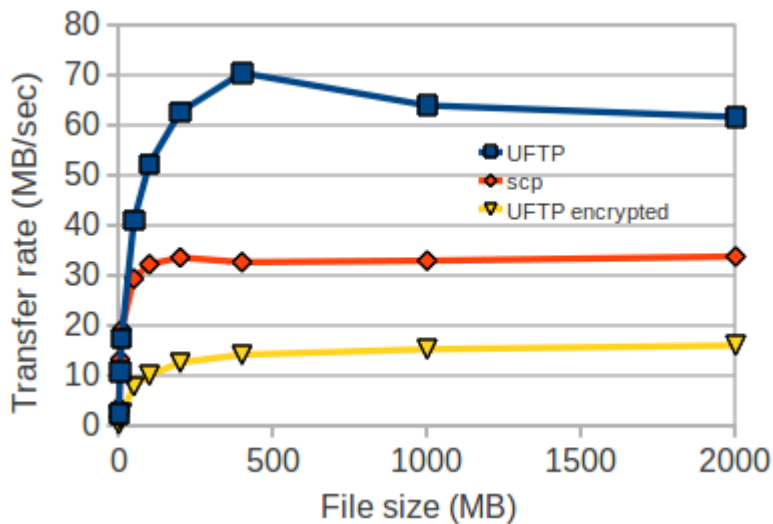


Figure 2. UFTP performance comparison in a server to server scenario.

Acknowledgements

The authors wish to thank Krzysztof Benedyczak, Björn Hagemeier, Michael Rambadt and Michael Stephan for early testing of UFTP and for giving important feedback.

References

1. M. Morgan, *ByteIO specification 1.0*, Tech. Rep. GFD-R-P 087, Global Grid Forum, 2006.
2. W. Feng and P. Tinnakornsriruphap, *The failure of TCP in high-performance computational grids*, in Supercomputing '00: Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM), (Washington, DC, USA), IEEE Computer Society, 2000.
3. W. Allcock, *GridFTP: Protocol extensions to FTP for the Grid*, Tech. Rep. GFDR-P 020, Global Grid Forum, 2003.
4. Y. Gu, X. Hong, and R. L. Grossman, *Experiences in design and implementation of a high performance transport protocol*, in SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing, (Washington, DC, USA), IEEE Computer Society, 2004.
5. UDT Java project. <http://sourceforge.net/projects/udt-java/>.
6. T. Oistrez, E. Grünter, M. Meier, and R. Niederberger, *A reliable and fast data transfer for grid systems using a dynamic firewall configuration*, in Proceedings of the 2008 UNICORE Summit, LNCS 5415, pp. 94 - 102, Aug. 2008.
7. J. Postel and J. Reynolds, *File Transfer Protocol*. RFC 959 (Standard), Oct. 1985.
8. Iptables. <http://en.wikipedia.org/wiki/Iptables/>.

9. **UFTP Documentation.** <http://unicore.eu/documentation/manuals/unicore6/files/uftp/>.
10. **Netcat.** <http://netcat.sourceforge.net/>.
11. **UNICORE downloads.** <http://sourceforge.net/projects/unicore/files>.

UNICORE Data Management: Recent Advancements

Krzysztof Benedyczak^{1,2}, Tomasz Rękawek^{1,2},
Jędrzej Rybicki³, and Bernd Schuller³

¹ Faculty of Mathematics and Computer Science,
Nicolaus Copernicus University, Toruń, Poland
E-mail: {golbi, newton}@mat.umk.pl

² Interdisciplinary Center for Mathematical and Computational Modeling,
University of Warsaw, Warsaw, Poland

³ Jülich Supercomputing Centre,
Research Centre Jülich, Jülich, Germany
E-mail: {j.rybicki, b.schuller}@fz-juelich.de

Continuous growth in data-oriented research activities makes an efficient handling of large amounts of data increasingly important. The scientific data are often distributed, heterogeneous and shared between various research communities. Thus not only an increase in data volumes but also in their complexity can be observed. All that together makes a means for storing, accessing, transferring and searching the data important parts of the modern researcher's arsenal. In this paper we present how the UNICORE Grid middleware solution can propel data-oriented research activities. In particular we will show that UNICORE is able to provide seamless access to large amounts of data stored in a distributed fashion. We will also present a novel protocol for efficient transfer of large amounts of data. Last but not least, new features for organising and searching the data will be presented. By providing a holistic view of the new UNICORE features supported by the preliminary performance evaluations and usage examples, we underpin the UNICORE capabilities to efficiently support data-intensive research.

1 Introduction

A recent report of the European Commission¹ suggested that the coming years will be marked by increasing amounts of data produced and processed. This emerging “wave of data” is accompanied by a paradigm change in science: from computer- to data-oriented processing. Although the UNICORE middleware is well established in supercomputing-oriented Grid infrastructures, it was mainly confronted with compute-intensive processing so far. Thus it is a proven fact that UNICORE can provide a secure and user-friendly access to computational resources. Its capabilities in regard to data-intensive computations, on the other hand, are not so well-known. This paper will show that UNICORE is in fact well prepared for the announced paradigm change. We intend to present a holistic overview of the UNICORE capabilities for storing, transferring and managing large amounts of data.

In order to cope with large amounts of scientific data, first and foremost a scalable way of storing data must be provided. For this purpose a new distributed storage facility, called Distributed Storage Management Service (DSMS) was introduced into the UNICORE architecture. The subsystem provides end-users with a single access point for the whole Grid storage, without a necessity to be aware of the physical location of files. To enable such a global view, the DSMS combines storage resources exposed by other, “normal” UNICORE SMS instances. Such an approach allows for a straightforward integration of the DSMS with the rest of the UNICORE stack.

The foreseen increase in the amounts of data that are produced and processed in the scientific process poses high demands on the efficient transport of the data from one place to the other. So far UNICORE is lacking in a convenient mechanism for high-performance, high-volume data transfer. The built-in solutions (HTTP based) do not provide the required transfer rates. In the Grid world, GridFTP² has been by far the most successful tool for efficiently transporting large amounts of data with adequate performance. While it is possible to use GridFTP in UNICORE, it is not well integrated, primarily because it relies on proxy certificates. Furthermore, GridFTP requires an open port range on site firewalls, which is widely considered to be a security weakness. To overcome these shortcomings, a new file transfer tool called UFTP was developed and integrated in the UNICORE stack. UFTP enables quick and reliable data transfers by employing multiple, parallel TCP data connections. In addition, it requires only a single open port in the firewall and will open more required ports on demand using the FTP protocol. UFTP is implemented in Java, thus it integrates smoothly with UNICORE clients and services.

Finally, in order to make the data more usable for the users, an urgent need of services for organising, indexing, and searching scientific data has been recognised and addressed. UNICORE already provided an experimental implementation of a metadata service: Metadata Management Framework (MMF)³. The service has been recently updated and extended by a new means for extracting and searching through the metadata. The new feature of automatic extraction of the metadata realised by a storage crawler and Apache Tika toolkit⁴, supports users in the process of describing scientific data.

The rest of the paper is structured as follows. In Section 2 we present technical details of the distributed storage element and some performance evaluation of the proposed solution. Later, in Section 3, the new data transfer protocol: UFTP is presented. Section 4 explains the framework for managing and accessing metadata in UNICORE. We also provide some usage examples. We conclude our paper in Section 5.

2 Distributed Storage

In order to cope with large amounts of data, first and foremost a scalable, performant solution for storing the data is needed. Generally, one way of achieving scalability and performance is to use distribution and divide the workload between multiple parties. Examples of such an approach to data management are iRODS⁵ and Hadoop⁶. UNICORE can access this third-party distributed storages via adaptors: UniRODS⁷ and UniHadoop⁸. Adaptors implement classical UNICORE Storage Management Service interface and translate its methods to operations on the respective backends: iRODS and Hadoop. The shortcomings of such an approach are twofold. The internal security of both backends differs from the security model used in UNICORE, which makes full, transparent, and secure integration impossible. Subsequently, the usage of third-party solutions increases administration overhead; two systems need to be maintained: UNICORE and the respective distributed storage system. Thus, recently a new development of simple, native, integrated distributed storage for UNICORE was undertaken. In this section we will present only a short overview of the solution more interested readers are referred to the full paper on distributed storage⁹.

2.1 Design

Before we dive into technical details and design principles of the distributed storage manager, let us first shortly explain how access to storage resources is typically realised in UNICORE. The Storage Management Service (SMS) provides an access point to a storage resource (directory on a disk). It embraces operations for file and directory manipulation and for transferring files to and from the storage. In a typical installation of UNICORE there are multiple instances of SMS, each of which is accessed via separate service instance. Usually with increasing demands on storage place, new instances are added, making a seamless access to the resources much harder for the users.

In short, DSMS is designed as an additional layer above existing SMSes which provides a uniform view of the combined, multiple storage instances and constitutes a single access point for all storage interactions. DSMS does not store files itself, but rather facilitates easy access to files stored in standard SMS storages. It also keeps track of the file location in a catalogue. The subset of underlying SMS storages is dynamic: it is possible to scale-out an existing service by adding new resources.

DSMS aims at providing its functionality via the SMS interface so that no changes in the Grid client software and Grid services are needed. When interacting with such a distributed storage, the user has an impression of interacting with a single SMS. Moreover it employs the same security mechanism and thus it can be easily integrated in existing Grid infrastructures.

The client interacts with the system via instances of DSMS, so called entry points. To achieve maximal performance, DSMS can provide an arbitrary high number of entry points. Each of which can be used to access distributed resources independently.

2.2 Implementation

The most important part of the proposed solution is probably the *dCatalogue*. It is responsible for mapping the logical file names, used in DSMS and visible to users, on physical file locations in the underlying storages. The user should not access the underlying instances directly but rather via the DSMS interface. Please recall that the entry point of a DSMS exposes exactly the same interface as a classical SMS. It makes the implementation of DSMS quite straightforward: DSMS only serves as a kind of *proxy* between client and the (hidden) SMSes. How this works in detail can be best explained with an example. Let us assume that a user wants to download a file from the distributed storage. For that, she calls `ExportFile` method of a DSMS entry point. DSMS connects to `dCatalogue` and resolves the logical file name provided by the user into a physical SMS where the file is stored. Next, DSMS connects to the given SMS and calls `ExportFile` function. The function, according to the SMS interface contract, returns an Endpoint Reference Address (EPR) of the newly created `FileTransfer` instance. The EPR is then passed by DSMS back to the client, which can start the transfer. The solution is transparent both for the client software and for the SMS instances, no changes are required to make them compatible with DSMS.

The implementation of methods accessing multiple files in one go, is a little bit tricky. An example of such a method is a request for a list of files in the storage. In this case DSMS not only passes the original request to the underlying storages but also collects and

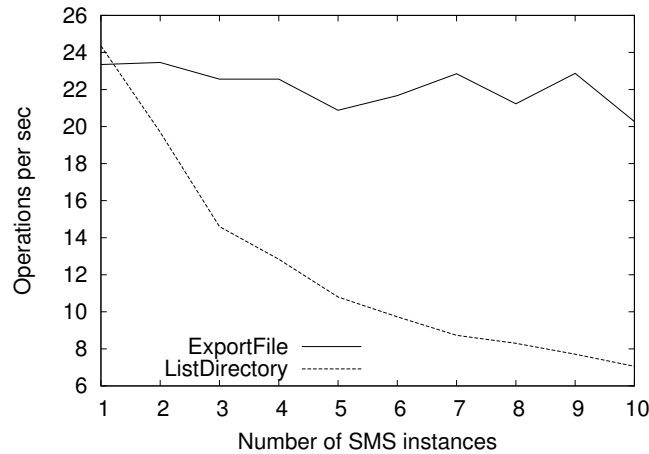


Figure 1. Scalability estimations of distributed storage.

aggregates the responses of single storages and afterwards sends the aggregated answer back to the user.

It is crucial for the system that dCatalogue is in sync with the content of the underlying storages. We might differentiate here between two kinds of potential inconsistency: a file is in a physical storage but not in the dCatalogue, or a file is present in the dCatalogue but not in physical storage. Such situations should not occur in the normal operation of the service as the DSMS “hides” the storages from the user. Still, it is possible to enable an explicit synchronisation of the storage content with the dCatalogue. It yields, however, a performance penalty. Especially, in cases when the user requests access to a file which is not listed in the catalogue. Instead of returning `FileNotFoundException` instantly, DSMS assumes an inconsistency, and first tries to resolve the problem by searching in underlying storages for the requested file. If the search fails, an exception is generated.

2.3 Performance Evaluation

In a simple experiment we estimated the scalability of the proposed solution. For the tests two production machines were used. One machine ran DSMS and the other hosted dCatalogue. Underlying storage instances were divided equally on both machines. The results are depicted on Figure 1, where the mean number of recorded operations per second versus the number of underlying storage instances is depicted. We took an average from ten runs. No explicit synchronisation was used. The performance of the simple operations like file transfer is independent from the number of instances. The performance of an “aggregating” method (`ListDirectory` in this case) decreases with the number of instances. This is caused by the fact that in the current implementation the queries to underlying storages are sent sequentially. An obvious improvement would be to send them in parallel, we are already working on that.

3 UFTP

So far, high-performance data transfers were difficult to achieve in UNICORE. The default transfer protocols (BFT and ByteIO) struggled in accomplishing this goal. BFT relies on the available HTTPS channels and consequently requires multiple socket connections to get the data from the client to the file system. For example, in a data upload from client to server, the data are first sent to the gateway and from there to the UNICORE/X server. Finally the UNICORE/X server opens a socket connection to the TSI. Overall, three independent connections are used. Under favourable circumstances, BFT achieves a performance of several megabytes per second. ByteIO is even worse because the transferred data are placed into SOAP messages, resulting in high overhead.

To achieve high-performance, direct transfers from client to a process running on the cluster login node is required. However, this is generally not possible due to the site firewall. To overcome this limitation two approaches were used in UNICORE so far: GridFTP and UDT. UDT (UDP based data transfer)¹⁰ relies on a library written in C++, which is not platform independent and makes full integration on both server and client side hard, if not impossible. The second option is to use the well-known GridFTP² from the Globus Toolkit. Unfortunately this integrates rather badly with the UNICORE, for multiple reasons. Since GridFTP uses proxy certificates for authentication, it is not fully compatible in UNICORE security model. Furthermore it requires open port ranges in the site's firewall, which is regarded as a serious security flaw. Last but not least: GridFTP cannot be readily integrated into UNICORE clients due to a missing native Java implementation.

3.1 Implementation

The newly developed UNICORE UFTP tool mitigates drawbacks of all the aforementioned protocols and a facilitates high performance, secure, well-integrated solution for data transfers in the Grid. The solution encompasses two parts: a server daemon and a set of libraries. The server, `uftp`, is deployed on the cluster login node and waits for incoming connections, mimicking a generic FTP server. Only one port for control connections, has to be kept open in the firewall. Clients can connect to this port and request a data transfer connection using the well-known FTP protocol. Modern state-full firewalls understand the FTP protocol and they dynamically open ports for data transfers when required. As soon as a data transfer connection is available, the client can communicate *directly* with the login node. The firewall takes care of closing the additional ports, after the data transfer ends. The usage of a direct connection between client and server results in a good performance of the data transfer, which can be further improved by using multiple, parallel data connections. This feature is supported by both server and client libraries and remains transparent for the firewall.

The FTP protocol does not provide sufficient security. Thus to prevent unauthorised access, UFTP relies on its own authentication and authorization mechanisms. First and foremost, it makes sure that only client connections that have been negotiated beforehand are accepted. To achieve this, an SSL connection between `uftp` and UNICORE/X server is established. Through this secured channel UNICORE/X informs the server about expected incoming requests of the authorised users. Connections which are not announced beforehand are discarded by the UFTP server.

Additional security measures are implemented to prevent unauthorised access to the file system. Also the data connections can optionally be encrypted to prevent man-in-the-middle attacks. A detailed description of this features together with a deeper explanation of the UFTP transfer protocol, its implementation and integration into UNICORE can be found in the full UFTP paper¹¹.

The UFTP base libraries and the `uftpd` server are written in Java, allowing for seamless integration into both UNICORE service stack and client software. Summarising, UFTP offers high-performance file transfer that is tightly integrated into UNICORE, platform independent, easy to deploy, and thus solves the problem of efficient data transfer in full accordance with the basic UNICORE principles.

4 Metadata Management

Presented solutions for storing and transferring the data, constitute a basis for handling data-oriented computation efficiently. However, one important problem remains unsolved. With increasing amounts of (distributed) data made accessible to the users, the process of locating particular pieces of content becomes harder and harder. This is amplified by the fact that the “data production” is often a joint effort of many parties spread all over the world. So that the resulting data are frequently stored in various file formats or file hierarchies and no common naming convention can be taken for granted. Locating a particular content under this circumstances is like looking for a needle in a haystack or, considering the far-reaching distribution, in multiple haystacks. Although DSMS can provide a uniform view of the multiple storage systems, and thus reduce the number of “haystacks”, an efficient way of searching in storages needs to be provided.

Since the amount and size of files in a single storage can be arbitrary high, it is not feasible to perform a full-text search in all files. A much better way of getting to grips with the data is to provide a way for describing data resources. Such “data on data” or metadata can describe the *content* of particular files independently from their representation (file format or naming convention used), physical locations, or origins. Since the metadata will be smaller than the content it describes, it can be indexed and searched much more efficiently.

4.1 Design

UNICORE includes a service for managing and searching metadata: the Metadata Management Framework. Within this framework it is possible to describe any given file stored in a UNICORE storage by a set of user-defined attributes. One of the most important design decision made when implementing MMF, was to give the user maximal freedom as to what attributes she can use to describe the resources. MMF does not impose a rigid set of attributes that must be provided, but rather allows to add user-defined key-value pairs describing the content. In other words: no metadata schema is used. The current solution addresses the most general scenario of users representing different research communities working together on a common set of data. Each of such communities can use its own semantic to describe particular content and sometimes the descriptions are not compatible with each other. With the current implementation of the metadata manager, there will be no conflict between the descriptions. Such flexibility and freedom come at the cost of their

own. The drawbacks of our approach are twofold. First, it is almost impossible to automatically validate the metadata provided by the user. Only syntax validation of the submitted metadata can be (and is) performed. Secondly, duplication and redundancy of the data can occur. We believe that the shortcomings are outweighed by the gained flexibility.

Further architecture decision made when developing Metadata Management Framework for UNICORE was to store metadata externally, that is metadata will not be included in the original data files. Again we aimed at the maximal flexibility of the solution. Although some of the common file formats allow to extend files with additional attributes, e. g., it is possible to describe a `jpeg` image with Exif metadata¹², not each file format supports such an inclusions of the metadata. Thus we followed a different approach and stored the metadata externally in separate files. We also used a very simple syntax to store the metadata: JSON¹³ representation of the key-value pairs. The main criterion here was to avoid vendor lock-in, and make possible migration of the data (and corresponding metadata) as easy as possible.

The possibility to describe the content of the UNICORE storages with metadata is only a first step in helping the user to organise their data. From their perspective the most important function of the metadata framework is the ability to search through the collected metadata. Therefore the metadata should not only be stored in the storage along with the original files but also indexed to enable efficient search.

4.2 Implementation

All the methods for manipulating and searching the metadata are implemented as a stand-alone web service part of the UNICORE Atomic Services. It is a generic service and it theoretically allows to create and manage metadata descriptions for any given resource (not only files in a storage). However, the metadata support for storage resources was most urgently needed. Thus we will focus on this particular use-case in the rest of the paper. Technically, the metadata service is implemented as a `WS-RF` web service. The metadata service and the storage access service (SMS) in UNICORE are closely coupled. For each UNICORE storage instance, a new instance of the metadata service is also created. Upon creation, the metadata service is provided with a reference to the storage so that the file access is realised exclusively via `IStorage` interface. An important ramification of such an implementation is the fact that the metadata service is compatible with all storage implementations available in UNICORE.

The metadata service includes all basic methods for manipulating metadata, in form of a *CRUD* interface: `CreateMetadata`, `GetMetadata`, `UpdateMetadata`, and `DeleteMetadata`. When creating or updating a metadata description of a file stored in a UNICORE storage, the user has to provide a file name and a set of key-value pairs. The description is stored in JSON format in a file `fileName.metadata`, where `fileName` is the file name of the described file. For deletion or retrieval of the metadata only a file name has to be provided.

Since the manual creation of the metadata is an error-prone and cumbersome activity, we provide a tool to support the process. The user can start automatic extraction of the metadata in a given storage, by calling the `StartMetadataExtraction` method of the metadata manger service. The service will then go through all the files in the given storage directory and try to extract metadata automatically by using the Apache Tika content analysis toolkit⁴. Tika supports a wide range of file formats, e. g., `xml`, `doc/odt`,

Table 1. Query examples.

Query type	Syntax example	Result
Wildcard	foo*	foot, foobar, . . .
Range	[bar TO foo]	bar, center, desk, . . . foo
Compound	foo OR bar	foo, bar
Fuzzy	foobar	foobar, flobar, foobar, . . .

pdf, jpeg and it is able to extract a lot of useful information. It can also be easily extended to support further formats. Please note that such an extraction needs to be performed only once for the storage, and not each time a user wishes to access the metadata.

Given the metadata describing files are available (as a result of either manual creation or automatic extraction), the searching functions of the metadata service can be employed. The respective web service function name is `SearchMetadata` and it takes two parameters: a query string and a boolean value indicating whether to perform an advanced query or not. The functionality is realised with help of powerful search engine: Apache Lucene¹⁴. Each of the metadata manipulating methods described above, beside storing the data in the storage, also passes the metadata to the index to keep it up-to-date. While searching, the query string is passed to Lucene and returned matching items are passed back to the user.

Lucene supports advanced query syntax e. g., wildcard queries, range queries and compound queries. Probably the most interesting query type supported by Lucene are the “fuzzy” searches, where not only exact matches but also all items semantically similar to the query string (ordered by the similarity grad) are returned. A quick overview of supported query types is provided in Table 1, for more examples we refer the reader to the documentation of the Apache Lucene project¹⁴.

Presented methods of the metadata service embrace most of the basic functionality a user might expect, including advanced searching and automatic metadata extraction. They probably do not cover all possible use-cases of metadata manipulation, but strike a good balance between simplicity and sophistication.

5 Conclusion

In this paper we have presented recent developments taking place in UNICORE with regard to data management functionality. In particular we presented how it is possible to store large amounts of data in a distributed fashion by using UNICORE distributed storage service. Furthermore a novel protocol for transferring large amounts of data with high performance (UFTP) was recently introduced in the UNICORE stack. The set of the data management functionalities is completed by the metadata management framework to manipulate and search metadata descriptions of the stored file resources.

The recent advancements and data-related features of UNICORE reinforce the statement that this middleware solution can cope with large amounts of data and support data-oriented scientific processes.

References

1. John Wood, Thomas Andersson, Achim Bachem, Christoph Best, Françoise Genova, Diego R. Lopez, Wouter Los, Monica Marinucci, Laurent Romary, Herbert Van de Sompel, Jens Vigen, Peter Wittenburg, and David Giaretta. *Riding the wave: How Europe can gain from the rising tide of scientific data*. European Union, 2010. Final report of the High Level Expert Group on Scientific Data: A submission to the European Commission.
2. William Allcock. GridFTP: Protocol extensions to FTP for the Grid. Technical Report GFD-R-P 020, Global Grid Forum, 2003.
3. Waqas Noor and Bernd Schuller. MMF: A flexible framework for metadata management in UNICORE. In *Proceedings of the 2010 UNICORE Summit*, volume 5, pages 51–60, May 2010.
4. Apache Tika Project. <http://tika.apache.org/>.
5. The iRODS project. <https://www.irods.org/>.
6. Apache Hadoop Project. <http://hadoop.apache.org/>.
7. Katarzyna Derc. Integracja systemu iRODS ze środowiskiem gridowym UNICORE 6. Master's thesis, Nicolaus Copernicus University, Toruń, Poland, 2008. In Polish.
8. Wasim Bari, Ahmed Shiraz Memon, and Bernd Schuller. Enhancing UNICORE storage management using Hadoop distributed file system. In *Proceedings of the 2010 International Conference on Parallel Processing (ICPP '10)*, pages 345–352, 2010.
9. Tomasz Tomasz, Piotr Bala, and Krzysztof Benedyczak. Distributed storage management service in UNICORE. In *Proceedings of the 2011 UNICORE Summit*, July 2011. to appear.
10. Thomas Oistrez, Egon Grünter, Marcus Meier, and Ralph Niederberger. A reliable and fast data transfer for grid systems using a dynamic firewall configuration. In *Proceedings of the 2008 UNICORE Summit, LNCS5415*, pages 94–102, August 2008.
11. Bernd Schuller and Tim Pohlmann. UFTP: high-performance data transfer for UNICORE. In *Proceedings of the 2011 UNICORE Summit*, July 2011. to appear.
12. Exchangeable image file format for digital still cameras: Exif version 2.3. Technical Report 008, Camera & Imaging Products Association, 2010.
13. Douglas Crockford. The application/json media type for javascript object notation (JSON). RFC 4627, July 2006.
14. Apache Lucene Project. <http://lucene.apache.org/>.

1. **Three-dimensional modelling of soil-plant interactions: Consistent coupling of soil and plant root systems**
by T. Schröder (2009), VIII, 72 pages
ISBN: 978-3-89336-576-0
URN: urn:nbn:de:0001-00505
2. **Large-Scale Simulations of Error-Prone Quantum Computation Devices**
by D. B. Trieu (2009), VI, 173 pages
ISBN: 978-3-89336-601-9
URN: urn:nbn:de:0001-00552
3. **NIC Symposium 2010**
Proceedings, 24 – 25 February 2010 | Jülich, Germany
edited by G. Münster, D. Wolf, M. Kremer (2010), V, 395 pages
ISBN: 978-3-89336-606-4
URN: urn:nbn:de:0001-2010020108
4. **Timestamp Synchronization of Concurrent Events**
by D. Becker (2010), XVIII, 116 pages
ISBN: 978-3-89336-625-5
URN: urn:nbn:de:0001-2010051916
5. **UNICORE Summit 2010 – Proceedings**
edited by A. Streit, M. Romberg, D. Mallmann (2010), iv, 123 pages
ISBN: 978-3-89336-661-3
URN: urn:nbn:de:0001-2010082304
6. **Fast Methods for Long Range Interactions in Complex Systems
Lecture Notes**
edited by P. Gibbon, T. Lippert, G. Sutmann (2011), ii, 167 pages
ISBN: 978-3-89336-714-6
URN: urn:nbn:de:0001-2011051907
7. **Generalized Algebraic Kernels and Multipole Expansions
for massively parallel Vortex Particle Methods**
by R. Speck (2011), iv, 125 pages
ISBN: 978-3-89336-733-7
URN: urn:nbn:de:0001-2011083003
8. **From Computational Biophysics to Systems Biology (CBSB11)**
Proceedings, 20 – 22 July 2011 | Jülich, Germany
editet by P. Carloni, U. H. E. Hansmann, T. Lippert, J. H. Meinke, S. Mohanty,
W. Nadler, O. Zimmermann (2011), v, 255 pages
ISBN: 978-3-89336-748-1
URN: urn:nbn:de:0001-2011112819

9. **UNICORE Summit 2011**

Proceedings, 7–8 July 2011 | Toruń, Poland

edited by M. Romberg, P. Bała, R. Müller-Pfefferkorn, D. Mallmann
(2011), iv, 150 pages

ISBN: 978-3-89336-750-4

URN: urn:nbn:de:0001-2011120103

The UNICORE Grid technology provides a seamless, secure, and intuitive access to distributed Grid resources. UNICORE is a full-grown and well-tested Grid middleware system, which today is used in daily production worldwide. Beyond this production usage, the UNICORE technology serves as a solid basis in many European and International projects. In order to foster these ongoing developments, UNICORE is available as open source under BSD licence at www.unicore.eu.

The UNICORE Summit is a unique opportunity for Grid users, developers, administrators, researchers, and service providers to meet and share experiences, present past and future developments, and get new ideas for prosperous future work and collaborations. The UNICORE Summit 2011, the seventh in its series, took place 7–8 July at Nicolaus Copernicus University, Toruń, Poland.

The proceedings at hand include a selection of 17 papers that show the spectrum of where and how UNICORE is used and further extended, especially with respect to Clouds and Portals.

This publication was edited at the Jülich Supercomputing Centre (JSC) which is an integral part of the Institute for Advanced Simulation (IAS). The IAS combines the Jülich simulation sciences and the supercomputer facility in one organizational unit. It includes those parts of the scientific institutes at Forschungszentrum Jülich which use simulation on supercomputers as their main research methodology.