UNICORE and GRIP: Experiences of Grid Middleware Development

Denis A. Nicole
School of Electronics & Computer Science
University of Southampton
Southampton, U. K.

Abstract—*We describe our experiences with the UNICORE Grid environment. Several lessons of general applicability can be drawn in regard to user uptake and security. The principal lesson is that more effort should be taken to be made to meet the needs of the target user community of the middleware development. Novel workflow strategies, in particular, should not be imposed on an existing community.*

Keywords: GLOBUS, GRID, HPC, UNICORE, Scientific Computing

1 Introduction

The University of Southampton was a participant in the European Union Information Society Technologies project titled *GRid Interoperability Project* (GRIP IST-2001-32257) which ran from 2002 to 2004. Our primary task was to work with the German weather service *Deutscher Wetterdienst* (DWD) to ensure that their *Relocatable Local Model* (RLM) could be executed effectively on mixed UNICORE [1,2,3], GLOBUS2 and GLOBUS3 (OGSI) servers.

1.1 The Meteorological Application

The RLM is a local area weather forecasting model that takes two types of input. It uses as its initial value data the results of a forecasting run of the DWD global model GME. This is computed on a novel icosahedron-based grid and has to be interpolated onto the conventional grid used by the RLM. It also requires high resolution static data: orography, vegetation type, soil texture etc. for the area under simulation. A large calculation is performed on a parallel supercomputer of the Cray T3E or IBM SP2 class at a major centre such as CSAR (Manchester, UK), ECMWF (Reading, UK) or DWD (Offenbach, Germany). The data used in, and output from, this run is in the form of GRIB files, a self-describing format used internationally. The output data has to be rendered into usable graphical form and transmitted to the user scientist for interpretation or, ultimately, for viewing by an end customer. There is the expectation that this technology may eventually be used to provide highly accurate on-demand local forecasts to *pay-per-view* end users.

From the description above, it will be seen that the RLM requires a four-step workflow:

1. *Data collection.* All static data—orography, vegetation type and soil texture—and all data sets of a foregoing run of the Global Model (GME), which are necessary for the designed RLM run, have to be extracted from the CSO Oracle database located on DWD's data server. This is regarded as high-value data and proper precautions must be taken to secure it.
2. *Data interpolation.* Before using the GME data as initialisation and boundary data for the LM run, it has to be interpolated from the icosahedron grid used in the GME into the rotated latitude-longitude grid of the high resolution LM. This step represents a relatively small computational load and can run on the DWD facilities, although it can generate large volumes of output data.

3. *LM run.* The main step is to run the Local Model on a high performance supercomputer using the data from step two. This represents the principal computational load and it is desired to offload this to a suitably brokered service.
4. *Visualisation.* After the LM run is finished, the user can transfer the data files back to their own system and post-process them at will or use a visualisation facility on the high performance system to generate graphical data and transfer this back for further use.

The bulk of this paper describes the lessons learnt during the GRIP work which might hopefully be of value to other Grid software developers. In most cases they are rather generic and apply equally to GLOBUS and OMII [5] as they do to UNICORE. We can group the issues into two principal domains: user uptake and security. Within each domain we make observations about the problems found and the solutions adopted or proposed.

1.2 UNICORE

The core software technology of GRIP was UNICORE. This is a grid workflow system that has been in development from 1997, initially under funding from the German Ministry for Education and Research (Bundesministerium für Bildung und Forschung, BMBF) and more recently the European Union Information Society Technology programme. It has been described elsewhere[1]. From the user's perspective, a Java application, the UNICORE Client, is provided and all interaction with the system takes place through this client. The user may use a graphical component of the client to construct a workflow of job steps to run on a variety of remote server systems. These may be sequenced and may transfer data between steps. Each step can itself be constructed through other GUI interfaces; these can either represent standard facilities such as shell scripts and FORTRAN compilations or can be dedicated GUIs for specific end-user applications such as CPMD Gaussian or Amber. The Client provides mechanisms to display the progress of a job through the various servers and to return results to the user.

The server side of UNICORE is three-layered. The Gateway provides a simple HTTPS portal for each site which allows only authorised users access to the rest of the UNICORE system behind the firewall. The Network Job Supervisor (NJS) is a larger Java code which normally runs on a small dedicated system within the site firewall. There is one NJS instance for each compute server at the site. This uses templates (the so-called Incarnation DataBase, IDB) to translate the local component of a job into a sequence of small shell scripts which are passed on via a socket to the Target System Interface (TSI); this is normally a small perl script running on the compute server itself.

The NJS processes only the component of a job which has been targeted at its own server; other components are forwarded at the appropriate time to the NJSs associated with the various requested servers.

The UNICORE code base contains three substantial components. The Client and NJS have been described above. The shared data structures of UNICORE are represented by the Abstract Job Object (AJO) classes. These 256 classes constitute a Java [4] ontology for supercomputer job submission, encompassing file transfer, various standardised job steps and resource allocation.

2 User Uptake

As mentioned in the introduction, the UNICORE activity has been funded continuously since 1997 through a variety of projects: UNICORE (1997–1999), UNICORE Plus (2000–2002, €2,400,000), EUROGRID (2000–2004, €3,450,000) and GRIP (2002–2004, €1,920,000). A fortunate by-product of the UNICORE architecture is that we can relatively easily estimate usage of the UNICORE grid by gathering statistics at one of the sites. When a UNICORE Client starts up it connects to each site on its Grid, as represented at the UNICORE site server, in order to check for output. We can thus gather usage statistics from the Gateway logs at just one of the sites. The statistics for late 2003 are shown in figure 1. The results are salutary; an average of three people per day used the Grid. Detailed examination of the logs show that about half of this usage is by software developers engaged in the funded UNICORE projects; the remainder represents a small group of end users.

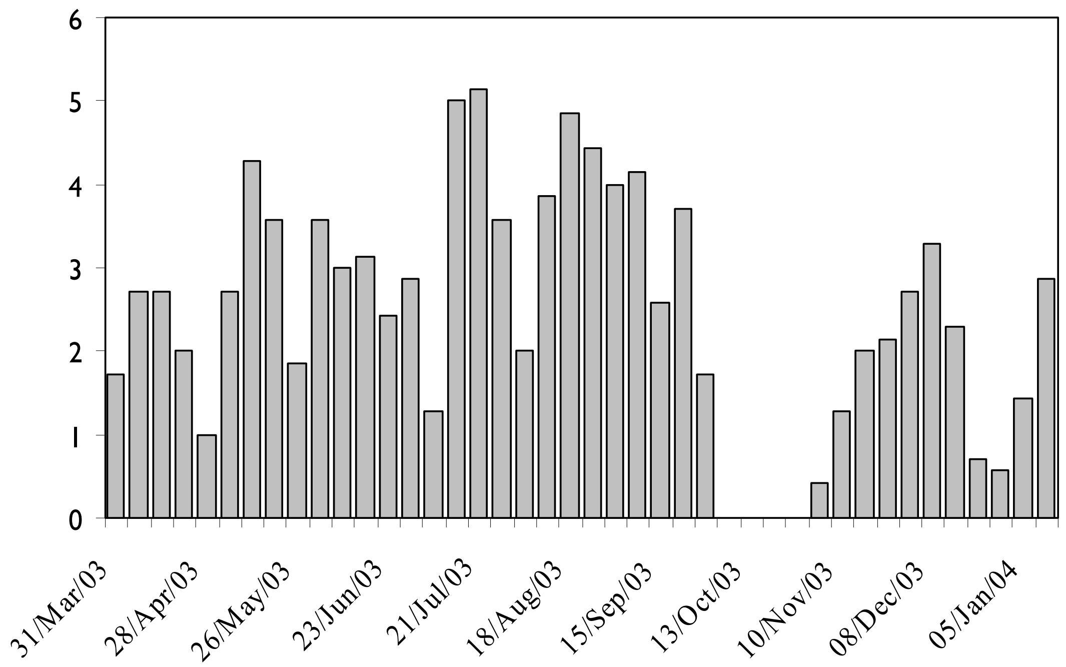

Figure 1: *UNICORE Grid usage: A plot of the average number of distinct Unicore users running with DWD in their Unicore site server each day. The October data is missing.*

User uptake of the UNICORE system has been disappointing. In some cases, administrative decisions at supercomputer sites have required uses to use UNICORE as the only permitted access mechanism; this has included academic users at DWD. In others, users have produced specific arguments as to why they must be permitted alternative supercomputer access mechanisms; this has included operational users at DWD. We have established a number of issues that have led to this poor uptake but, unsurprisingly, they are all associated with a perceived or actual failure to meet user needs.

2.1 Extensibility of the UNICORE Grid

Several factors conspired to make it difficult to add new compute servers to the UNICORE Grid. The first issue is associated with the PKI X509 certificates used by UNICORE. Personal user X509 certificates have become the *lingua-franca* of international eScience. They can freely be used for authentication in the Globus, UK eScience, EGEE and other communities. UNICORE user certificates are also of this standard form and can be used interchangeably with the other Grids, so long as the required root certificates are installed at the Gateway.

The UNICORE system requires additional certificates for the server side. The gateway certificates do not need any special properties, but each NJS also requires a certificate. These are required to hold a special extension object identifier (OID) assigned by the IANA to the Leibniz Computer Centre of the Bavarian Academy of Sciences (LRZ). This OID, named "unicoreKeyUsage" (1.3.6.1.4.1.7650.1) is required to have the value "unicoreNJS" for an NJS certificate. There are other assigned values for other UNICORE roles, but they are currently ignored. The NJS OID is necessary to prevent an ordinary user from spoofing an NJS and conducting a replay attack using another user's job. Unfortunately, many nationally accredited CAs are unwilling to issue certificates carrying unusual privileges and this has made it difficult to leverage existing CA facilities; apart from the excellent service provided by the University of Manchester to EUROGRID and associated projects, arrangements for UNICORE CAs have been a continuing problem. *Remedy: Only use completely standard PKI facilities.*

There is a second problem associated with the flow-through nature of the UNICORE workflow. Each NJS in the workflow sequence has to connect to the NJS associated with the next host server. This requires the NJSs in a workflow to recognise and accept each other's certificates. Thus even if a user is authenticated into two Virtual Organisations (VO), they cannot operate a shared workflow on both VOs computers unless the VOs are configured to authenticate each other. The same problem makes it impossible for a user to incorporate their own local systems into a workflow. *Remedy: Sequence workflows from a central workflow server, rather than by flow-through. This is the natural way to operate modern BPEL workflow and also makes it much easier to track deviant workflows associated with failed job steps.*

A final problem with extensibility arises from the need to establish trust in the UNICORE system. Major supercomputer sites take security very seriously and it is difficult to persuade them to abandon their tried and tested strategies. The European Centre for Medium-Range Weather Forecasts (ECMWF), for example has a sophisticated strategy, ECAccess which is based on temporary PKI certificates whose issue is authenticated using SecurID cards; actual access is then conducted using a modified FTP client. The site would not willingly give up this security and has insisted on special modifications to the UNICORE Client. *Remedy: The middleware provider must provide clear evidence of sound software engineering and must establish a bond of trust with major site managers. Perl scripts running as root do not inspire confidence.*

2.2 Workflow

UNICORE has its own workflow model, allowing a variety of sequential, concurrent, test and loop constructs. Through the normal client, this can only be accessed by manipulating job step *blobs* and workflow *arrows.* Unfortunately, it completely fails to meet the expectations of users. The compilation of a single FORTRAN source file, for example, represents a job step in the UNICORE ontology. Thus, if the abstract support for the compile step is to be used, the entire build process for a user application must be crafted, file by file, from blobs and arrows. Otherwise, the automatic mapping of compilation onto different hardware platforms is lost. The users, of course, expect to use their existing techniques to manage a build. At its simplest, this means shell scripts and Makefiles. More complex builds tend to use perl or, in some cases, tcl/tk and python. *Remedy: Middleware must show sensitivity to existing user practice. If a portable build facility is to be provided, it must offer a natural way of incorporating the target-specific information into the existing build process, say by providing string substitutions in the existing scripts.*

There is a much more serious problem with scheduling. The operational forecasters at DWD were completely unable to use UNICORE because its simple workflow between compute hosts did not meet their needs. They use SMS [6], a sophisticated scheduler developed at ECMWF which submits tasks (jobs) and receives acknowledgments from the tasks when they change status and when they send events. SMS knows the relationships between tasks, and is able to submit dependent tasks when a given task changes its status, for example when it finishes. The UNICORE workflow cuts across the SMS functionality. Researchers at DWD use another experimental system, NUMEX, not supported by Unicore. *Remedy: This was a requirements capture problem. If UNICORE was to offer workflow support, it needed to be in empathy with the existing facilities.*

2.3 Job Preparation

A major feature of UNICORE is its application-specific Job Preparation Agents (JPA). These are implemented as signed java plugins which a user can configure to be loaded into their UNICORE Client. Unfortunately, writing a JPA is difficult. The author has to write in Java and has to have a deep understanding of the UNICORE AJO ontology as well as being aple to program a Swing GUI. To make matters even worse, these JPAs have to be updated on each evolution of the UNICORE AJO *and* as the client applications evolve. This latter point is important; much of the *business logic* of a JPA is provided to shield the end user from various input restrictions in the application. These restrictions typically change with each version.

It is frequently the case that major applications already have their own job preparation GUI. An example would be the UK Meteorological Office Unified Model [7]. This has the Unified Model User Interface (UMII) which is based on the Generic Hierarchical User Interface (GHUI) which in turn is written in tcl/tk. This interface has 200 windows and 1000 questions; rewriting it in Java would be a major undertaking. In general, these existing application-specific job preparation GUIs are rarely written in Java as scripting languages are much more popular: perl, tcl/tk and Python. *Remedy: The client must support job preparation using existing popular scripting languages. Again, template substitution should limit the amount of rewriting required to support remote job preparation.*

In addition to the support for scripting languages, something needed to be done to simplify Java JPA creation. Early in the UNICORE project, a team at one of the application sites developed GuiGen [8], a GUI generator which could rapidly produce simple JPAs. Unfortunately, it was not adopted by the core UNICORE developers and quickly became outdated. Within the GRIP project, we were able to revive this approach at much lower cost by leveraging standard Java technologies: Beans and XML serialisation.

The primary aim of the new GUI tool is to allow useful Grid results to be delivered to end users without anyone in the application team having to develop a deep understanding of a complex programming model such as the UNICORE Abstract Job and its associated classes. It is nevertheless straightforward to incorporate complex business (scientific) logic in the client GUI by writing and deploying conventional, well-documented, Java Beans. Indeed, the whole GUI may readily be constructed non-programmatically in a standard Bean Box such as Borland Jbuilder. The jobflow is easily initially constructed from an exemplar job built with the original scripting and other plugins for UNICORE. This exemplar job may then be edited under advanced mode in the new GUI to introduce fields substituted by the bean components. Not only does this provide a quick and easy way to capture a jobflow; it also completely deskills the process of updating the GUI to accommodate new software versions of the target applications. The two application-specific parts of the GUI reside in XML files. This provides insulation from version changes in UNICORE. We would normally expect that *only* the application-neutral plugin code will have to change as UNICORE evolves. This decoupling solves the familiar *languages-times-targets* problem from compiler engineering; modifications to the application and to UNICORE are decoupled.

2.4 Interactive Access

During the GRIP project, it rapidly became clear that the user community required full interactive access to the compute hosts. This was partly for reassurance; interactive examination of the queues and filestore would give confidence that the submitted jobs were proceeding as expected. It was also essential for program development and particularly for access debuggers and performance tools such as Vampir and TotalView. The absence of this basic facility highlights a persistent problem with Grid environments—certainly including both Globus and UNICORE—that they are not *used* by either middleware or application developers. The developers use traditional tools such as SSH and X-windows rather than the end-user middleware. Indeed, for those developing scientific applications, there is little alternative as most sophisticated debuggers insist on X-windows. The effect is to create a "*them and us"* situation in which the middleware development tends to proceed largely disassociated from any immediate user community.

At DWD, interactive access to the supercomputers was via a two-step login. SSH2 was used to access a bastion host; from there the user could Telnet into the required system. A side effect of this mechanism was that remote X-windows access was impossible. Within GRIP, we were able to subvert the UNICORE Gateway to provide a secure channel into the main systems. This effort, like that in the Universal (RLM) GUI should reduce the damaging disconnect. Users enjoy all the security and ease-of use advantages of *single signon* while retaining their access to familiar host-based tools. Furthermore, this access is maintained in a manner totally independent of the client platform, and without compromising security on the client by exposing open sockets at a client-side X-server.

There are four key components to this new interactive capability: an ssh2 implementation, a vt100/Xterm emulation, a stand-alone X server in Java and a special technique to allow us to use the Java keystore (JKS or JCEKS) inside the Pallas/Intel client to authenticate an SSH connection. The details are explained below; it provides something of a salutary lesson about the dangers of adopting open-source security code. The Bouncy Castle software having this vulnerability is used by both UNICORE and Globus.

The combination of these components provides a facility of considerable utility. Firstly, we have a truly platform independent secure login facility with responsive full-screen access: entirely suitable for pine, vi, emacs etc. Secondly, the integrated X-server avoids the need to expose vulnerable sockets on the client; the SSH, vt100 and X-server all communicate as Java threads. A screenshot is shown in figure 3.Thirdly, the integration with the UNICORE client and its keystore allows a genuine single-sign-on and permits uses to use their standard X509 certificates to access *unmodified* `ssh` servers, using a tool within the user interface to generate public keys in a suitable format. Finally, the UNICORE client is able to use the plugin feature of the standard unmodified UNICORE Gateway to provide a portal through the site firewall.

2.5 Performance

If we take the more trivial first, then an immediate complaint of users in 2002 was the unresponsiveness of the whole system. Client startup was slow for users running older workstations and job submission also required progress through layers of Java servers. This has naturally improved with time as workstations are upgraded and Java-based systems are now wholly acceptable. It nevertheless generated significant resistance to early adoption; work proceeded much more quickly within DWD using ftp and telnet.

There is a more subtle performance issue exposed by our deployment of a special TSI as an interface to remote Globus servers. It turns out that the NJS maps the complex tree structure of an Abstract JOB into a long sequence of short shell scripts which create directories, move files etc. No attempt has been made to optimise this sequence as there is the assumption that the NJS and TSI are closely coupled. Each script requires a completion acknowledgement before the next can be sent. Unfortunately, infelicities in the Globus installation at some supercomputer sites, partially associated with attempts to validate the remote site by making a reverse connection, mean that it can take ten seconds to start up a script under Globus as it waits for a firewall-blocked connection to time out. The overall responsiveness of the UNICORE/Globus system becomes unacceptable.

3 Security

In the process of developing the UNICORE system, we exposed a number of security[9] issues. Two are technical and one is simply the Grid incarnation of a well known problem in user management. These real vulnerabilities compound the difficulty of persuading sites to adopt Unicore as a replacement for established mechanisms such as SecureID (at ECMWF), SSH (at DWD) and Globus (at CSAR, Manchester).

3.1 Code Injection

Code injection is typically associated with attacks on Web-based e-Business systems. Carefully crafted query strings sent from a web browser are misinterpreted as SQL commands when the server naively presents them to the database engine. Unfortunately, the same problem can easily arise with perl scripts; communication between the UNICORE NJS and TSI was insecure in this regard.

Messages over the socket from the NJS to the TSI are delimited by

```
ENDOFMESSAGE\n.
```

These messages are untainted simply by removing the trailing `ENDOFMESSAGE`, without attempting to parse their contents. This action is accompanied by the comment:

```
# I trust the source! and the setuid/setguidis downgrading!
```

Such possible problems were well understood by the authors of perl; the tainting feature is especially designed to reduce these risks.

Unfortunately, an end user was free to inject whatever string they wish into this protocol by, for example, asking to access a file with a carefully crafted name. A particular case, when talking to a real NJS, which frightened us was the possibility of a malicious client generating an AJO that contains file imports, where the filename has embedded within it something like:

```
ENDOFMESSAGE\n#TSI_IDENTITY victim NONE\nENDOFMESSAGE\n#TSI_EXECUTESCRIPT\n
 ...hostile script...\nENDOFMESSAGE\n
```

This would have the effect of executing `hostile script` under the privileges of any chosen user. *Remedy: Careful code audits are required* before *deployment to live operational systems. Explicit cases where language security features are circumvented call for special attention*

It is interesting to observe that several system administrators were nervous about installing the TSI script on their operational supercomputers; as it turned out, their caution was justified.

3.2 Stealing Private Keys

The UNICORE Client makes use of the Bouncy Castle library. Unfortunately, this library was not designed with the expectation that it could be executed by code that is not wholly trusted.

It is well known that passing objects back to trusted code from untrusted routines can be a general source of difficulty. The key point is that, if trusted code allows untrusted code to “handle” one of its objects, then it is usually essential that the object be “final” so that the untrusted code cannot subclass it to introduce misbehaving methods.

It turns out that the Bouncy Castle [10] package (used by Globus and Unicore) has just the above vulnerability. This turns out to be useful to us, but perhaps alarming to system administrators. The Interactive Job facility has to authenticate an SSH, not SSL, channel. The protocols differ and it does not seem to be possible to authenticate an SSH channel without direct access to the private key. This is achieved in our interactive plugin by subclassing the `X509V3CertificateGenerator` class. The code exploits the fact that `X509V3CertificateGenerator` is not a final class; it simply subclasses it to introduce a key-stealing method which, in this case, is used only for SSH authentication.

Of course, the real problem is that any plugin can routinely harvest private keys in this way. UNICORE plugins have to be signed, but the user might reasonably expect that the exposure associated with a bad plugin is to submitting a badly formed job; there is no reason that plugins should be allowed to steal keys in this way.

3.3 General key security

The UNICORE client improves on many other eScience Grid environments by providing a closed mechanism for creating, certifying and storing user private keys. Other systems commonly use Internet Explorer, Netscape (often Netscape 4) or `openssl` scripts to manage the key issue process. The effect is to leave private keys in small files scattered over the user's workstation, often protected by weak passwords that were intended only to move temporary files. In contrast, the UNICORE Client handles the entire process, keeping the users secret data locked in a Java keystore. Furthermore, the Client can be used to insist on a reasonable choice of keystore password. In aggregate, user key handling in UNICORE is typically better than the norm for, say, Globus.

Unfortunately, there remain serious threats to the private keys. UNICORE encourages an nomadic style of use; a user may freely roam to any workstation onto which they can download the Client, taking their keystore with them. As a result, private keystores are easy to harvest and many would succumb to a brute-force dictionary attack. Perhaps the biggest risk, however, is associated with spyware. It is relatively easy to cause keyboard logging software to be installed on a workstation; this easily enables both keystore and password to be stolen.

Olle Mulmo, Security Architect for the EGEE project will have us believe that the only remedy is to "Remove credentials from users' desktops" as "On the client side, we have surrendered the desktop" [11, 12]. *Remedy: There are two partial solutions: secure hardware keystores or retaining all user keys* only *on trusted key servers.*

References

[1] D Breuer, D Erwin, D Mallmann, R Menday, M Romberg, V Sander, B Schuller and P Wieder, *Scientific Computing with UNICORE*, NIC Symposium 2004 Proceedings (2004), 429–440.

[2] D Erwin, *UNICORE - a Grid computing environment,* Concurrency and Computation: Practice and Experience, **14** (2002), 1395–1410.

[3] A Streit, D Erwin, Th Lippert, D Mallmann, R Menday, M Rambadt, M Riedel, M Romberg, B Schuller and P Wieder, *UNICORE - From Project Results to Production Grids,* arXiv.org **cs.DC/0502090.**

[4] V Sander, D Erwin and V Huber, *High-performance computer management based on Java*, Future Generation Computer Systems, **15** (1999), 425–432.

[5] M Atkinson, D DeRoure, A Dunlop, G Fox, P Henderson, T Hey, N Paton, S Newhouse, S Parastatidis, A Trefethen, P Watson, and J Webber, *Web Service Grids: An Evolutionary Approach*, (2004). http://www.omii.ac.uk/paper web service grids.pdf

[6] *SMS—Documentation*, ECMWF. http://www.ecmwf.int/publications/manuals/sms/documentation/index.html

[7] *Unified Model User Guide*, Met Office, (2004).

[8] AReinefeld, HStüben Florian Schintke and G Din, *GuiGen: A Toolset for Creating Customized Interfaces for Grid User Communities,* Future Generation Computer Systems, **18** (2002), 1075-1084

[9] T Goss-Walter, R Letz, T Kentemich, H.-C. Hoppe and P Wieder, *An Analysis of the UNICORE Security Model,* Published Global Grid Forum Document **GFD.18** (2003).

[10] *Bouncy castle crypto APIs*, The Legion of the Bouncy Castle, (2004). http://www.bouncycastle.org/

[11] Olle MULMO, *Scaling up, surely. But scaling up securely?,*Presentation given at the UK e-Science Workshop on Grid Security Experiences and Practices, Oxford, July 2004, EGEE Document. https://edms.cern.ch/file/503080/1/2004-07-09-security-workshop-oxford.ppt

[12] Olle Mulmo et al., *Global Security Architecture*, EGEE Deliverable **DJRA3.1**, (2004). https://edms.cern.ch/file/487004/1.1/EGEE-JRA3-TEC-487004-DJRA3.1-1.1.pdf