A Service-Based Architecture for Integrating Globus 2 and Globus 3

Manuel Sánchez[1], Óscar Cánovas[2], Diego Sevilla[2], and Antonio F. Gómez-Skarmeta[1]

[1] Information Engineering and Communications Department
[2] Computer Engineering Department, University of Murcia, Spain
{msc, skarmeta}@dif.um.es
{ocanovas, dsevilla}@ditec.um.es

Abstract. During the past few years, Grid Computing has matured in terms of programming models and available tools. Some tools like the Globus Toolkit version 2 (GT2) are used in many international high performance distributed computing projects. Recently, the OGSA standard (*Open Grid Services Architecture*) has been defined, proposing a radically new philosophy compared to that of GT2. Analyzing the evolution of the scientific community working on Grid Computing, we foresee a progressive shift of current developments to this new standard, that already has a reference implementation: GT3. This paper describes the analysis and design of an architecture of OGSA Grid Services that aims to integrate both platforms seamlessly, allowing remote job invocation from GT3 to GT2 holding all the security properties, and transparent for the user.

1 Introduction and Motivation

The computing power and storage needed in scientific environments grow day by day, exceeding that offered by traditional computers. Thus, a new paradigm called *Grid Computing* emerged with the goal of sharing resources among dynamic organization coalitions in a coordinated, secure, and flexible way. Organizations belonging to the Grid can decide to share part of their resources in a controlled fashion, conforming *Virtual Organizations*[12].

Nowadays, the *Globus Toolkit*[2] is widely accepted as a *de-facto* standard for building Virtual Organizations, being GT2 the most widely used version. However, since the publication of *OGSA* (*Open Grid Services Architecture*)[11] in 2002, Globus is adapting to the emerging *Grid Service* concepts. Thus, a new version, GT3, appeared based on this paradigm, which implements all its functionality by means of *Grid Services* and standard interfaces, making the development of Grid applications easier.

In terms of security, as described in[13], the new version of *GSI* (*Grid Security Infrastructure*)[6], included in GT3, provides several advantages, such as the use of IEEE/GGF compliant proxy certificates[16], or the use of the SOAP standard

(*Simple Object Access Protocol*)[4] and the recent WS-Security specifications[3] to exchange authentication and authorization information.

Although both GT2 and GT3 try to ease the development of distributed computing applications, they have rather different mechanisms for accessing resources. While GT2 is based mainly on command line tools and scripts since it was designed for batch execution, in GT3, resources are not accessed directly, but by means of grid services, which are executed on the resources and perform a particular task.

Given the wide deployment of GT2, there are a lot of applications that rely on this Globus version. Although GT3 distributions contain an updated GT2 implementation, the migration process to GT3 does not seem to be a straightforward task: This new version involves a complete change in the operation manner which is incompatible with the previous version. Moreover, a number of organizations using both versions of Globus would want to make them interoperate. In order to achieve that integration, we have to define the elements needed to satisfy two main goals. First, the system must be able of processing the requests sent from GT3 clients, interpreting them, and building an equivalent request to be sent to the GT2 nodes. It is worth noting that this process must be transparent to the GT3 client: it should be able to use GT2 nodes as if they were OGSA grid services. Secondly, given that the request must be translated from GT3 into GT2 by, as we will show, an intermediate element that might not be trusted by both parties (for example, when the integration service is offered by a third party), it is necessary to protect the job submitted for execution to guarantee end-to-end security.

In this paper we propose an architecture based on grid services that achieve the described integration. This architecture is based on digitally signed jobs, allowing secure usage of GT2 resources by GT3 clients. It is composed, on one hand, by grid services local to GT3 clients, that is, hosted by their organization, and on the other hand, by external services hosted by third parties providing the integration service.

Similar work has been done in the GRIP project[15]. Among the main results of this project we can find some kind of interoperability between Globus and UNICORE, and the promotion of standards for interoperability in the GGF.

This paper is structured as follows. First, Section 2 provides a brief overview of the two different GT versions related to our work. Then, Section 3 describes the proposed architecture for performing the integration, that is, the different architectural elements, their relationships, and the mechanism based on digital signature that is used to protect the integrity of the submitted jobs. Section 4 shows the relevant details of the implementation. Finally, we conclude with our remarks and some future directions derived from this work.

2 Background

Nowadays, we can find several platforms for Grid Computing providing different sets of capabilities. Globus Toolkit is the most widely adopted, and is being used in several European research projects [5, 9, 8]. In the last two years, UNICORE[10] has also emerged as an alternative toolkit for Grid Computing, as can be seen from the several existing initiatives which are trying to integrate Globus and UNICORE in a seamlessly manner[15].

2.1 Globus Toolkit 2

Globus Toolkit[2] is a computing platform composed by applications and libraries for the management, discovery and monitoring of resources. GT2 provides an uniform access interface to the computing resources, either independent nodes or a whole cluster of workstations using different operating systems. One of the main elements of GT2 is *GRAM* (*Globus Resource Allocation Manager*), which is responsible for accepting job submissions and hiding the specific details of the platform executing those jobs. In an upper layer we can also find *DUROC* (*Dynamically-Updated Request On-line Co-allocator*), a meta-manager responsible for the coordination of several GRAMs in order to execute complex tasks (composed of other jobs).

Remote execution is guided by a resource specification language (RSL). By means of RSL, users specify the job to be launched and some related execution parameters, such as the number of processors involved or required memory.

2.2 Globus Toolkit 3

GT3 represents a completely different approach from GT2, as it uses *Grid Services* as its core. Grid services are specialized *Web Services* that include some new features, such as persistence, management of notifications, and use of *service data* elements. GT3 grid services must be run in a service container, and remote invocations make use of SOAP. Grid services are self-described by means of WSDL (*Web Service Description Language*) documents, which must be obtained by clients in order to access them. Complexity is reduced by means of client and server stubs, which are intermediate software elements derived automatically from the WSDL description. Therefore, details about SOAP and other technological elements are hidden from the developer's point of view.

Two of the most important novelties in GT3 are service data and notifications. The former allow the programmer to add *structured data* to the services, which can be accessed through a well defined interface. Then, service data can be used to expose the grid service internal state or metadata. Notifications are used by a grid service or *notification source* to notify changes in its state to any subscribed client or *notification sink*. Notifications are closely related to service data, because a client is not subscribed to a whole grid service, but to a specific service data belonging to the service.

The Java CoG Kit[1], in turn, offer a Java framework for accessing GT2 programs and services. In general, Commodity Grid (CoG) kits allow users, developpers and administrators to access the grid from a higher-level framework.

2.3 Security Mechanisms for Globus: GSI

Both GT2 and GT3 use the security services provided by GSI (*Globus Security Infrastructure*)[6]. GSI makes use of X.509 certificates for authentication purposes, and TLS for establishing confidential channels. It also supports different authorization policies, delegation of privileges, and can also be integrated with other local security systems being used in the organization.

3 Proposed Architecture

3.1 Requirements for the Integration of GT2 and GT3

The integration of both versions of Globus has a set of requirements derived from the intrinsic internal architectural organization of both platforms. First, intermediate elements with support for both GT2 and GT3 are necessary. Second, a RSL request must be built that describes the job execution request made by the user. Also, the code, input data, and output data must be transmitted. It would also be desirable that the client, the intermediate element, and the GT2 nodes in charge of the final execution could be located in different administrative domains, thus not having to belong to the same organization. Shielding the user of the details of GT2 (such as the RSL specification or the use of the GRAM protocol) is also desirable. Finally, the semantics of the execution stated by the GT3 client must be preserved along all the process, that is, the architecture must guarantee that neither the code nor the input or output data has been forged, as well as that the identity of the caller is not supplanted by any other entity.

3.2 The Elements of the Proposed Architecture

We have proposed a generic architecture with three different administrative domains, as shown in Figure 1. First, the domain of the GT3 client that wants to execute a job in a GT2 cluster. For the sake of simplicity, we suppose that this domain is composed exclusively of GT3 nodes, possibly connected to other GT3 nodes on different sites conforming a Virtual Organization. Second, an intermediate administrative domain exists to host the integration service, thus having to support both versions of Globus. This intermediate element can be seen as an enterprise (organization) dedicated to offer interconnection among Grids. The last administrative domain in the picture is the destination one, in which the execution of the job will be performed, composed of one or more GT2 nodes.

The main elements participating in the integration are the following:

- **GT2Gateway Service.** To avoid GT3 users having to explicitly build the GT2 RSL request, an OGSA service is introduced that builds this specification from the data given by the user (executable file, parameters, data files, etc.) This service is also responsible for signing the RSL and all the files on behalf of the user and for verifying the signed output generated by the GT2 nodes, in order to guarantee end-to-end integrity and authentication. The

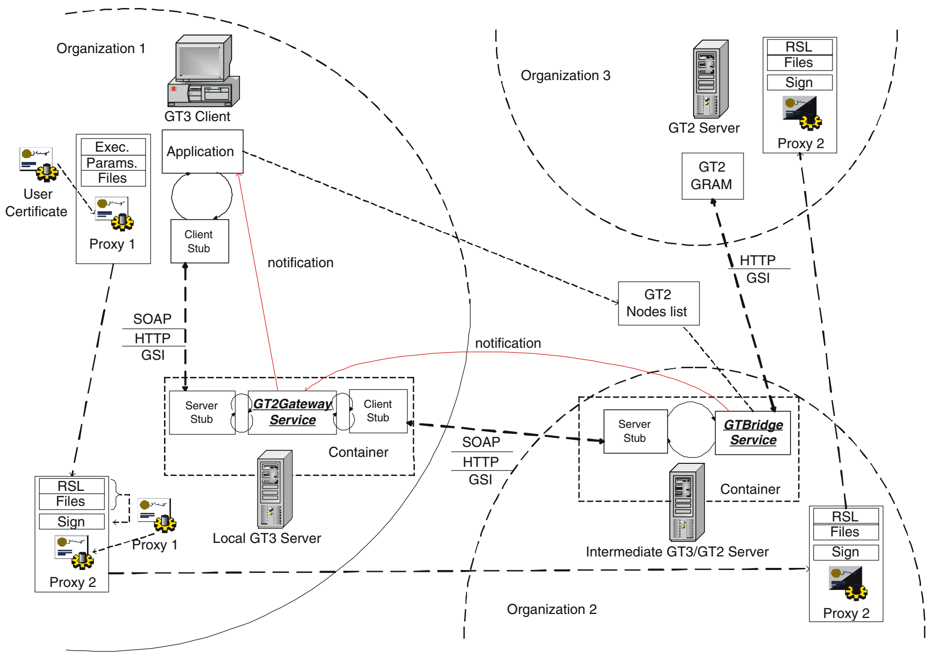

Fig. 1. Architecture of the proposed solution

GT2Gateway service must be running in the local domain to which the user belongs, because this service has to build and sign the whole description of the job. Besides, this service acts as a notification source for the client. We can find a similar approach to provide job integrity by means of job signing in UNICORE[10].

– **GTBridge Service.** This service offers two main functions: First, it is responsible for maintaining a list of GT2 nodes (or set of nodes) that are available to be used from GT3. This way, the user should first check this list (for example, using the *Index Service*) to see what nodes are available before interacting with the *GT2Gateway* service. And second, once the RSL is built, it receives from the *GT2Gateway* service the Job description, input files and executables, and the certificates needed to verify the signature, as we will see later. From this point on, the request has been converted into the GT2 format, and therefore it can be submitted to a GT2 GRAM server of other administrative domain. After executing the job, the results are collected by this service, and put in knowledge of the client.

– **GT2 GRAM with digital signature support.** The functionality of the GT2 GRAM server must be augmented (in form of a plug-in) in such a way that allows us to interpret digitally signed job descriptions. For that purpose, the RSL must indicate that this particular job is protected using a digital signature, making it to be interpreted by the modified GRAM. It is worth noting that this extension can be added without changing the base Globus installation in the node. Once the job is executed, the results are also protected against integrity attacks by means of a digital signature.

Finally, we also have to assure end-to-end security. Although GSI guarantees a secure communication between the entities in homogeneous environments, the introduction of an intermediate entity (the integration service) makes it necessary to deeply analyze the security implications of the integration.

3.3 The Integrity Problem

The proposed solution could have the problem of having to trust an external entity (the mediator) to which the execution of jobs is delegated. This external entity could modify the jobs before sending them for execution or even impersonate the user sending out jobs for execution without the user's knowledge. To overcome this problem, the job description is signed with the user's private key. This way the destination GT2 node can check the authenticity of the data received. Therefore, management of proxy certificates and private keys involved in the process is of paramount importance.

The Globus Security Infrastructure is based on the use of restricted user proxy certificates. These certificates are issued by the user for a temporal user-controlled key pair, and can be used to delegate operations to other elements of the system in a secure and controlled way (those delegation chains are not bounded). As can be seen in Figure 1, the GT3 client first generates a proxy that will be used by the *GT2Gateway* service to sign the job description. After that, this service generates another proxy, that will be sent, jointly with the signed job, to the *GTBridge* service. This second proxy, together with its certificate chain, will be used by the *GTBridge* service to contact the GT2 node and execute the job on behalf of the user. As the job description is signed with the first proxy, which is not accessible by the intermediate service, the contents of the job cannot be modified. Moreover, the use of the second proxy will allow executing the job on behalf of the user, thus applying the security policies mapped to that specific user (known as "*grid mapping*" in Globus). Also note that the architecture of GT3 imposes that a proxy certificate must be created for each service invocation.

4 Implementation of the Proposed Architecture

Once we have analyzed the main elements of our architecture, and having in mind the requirements imposed by such approach, we provide some details related to the implementation and operation of those elements.

4.1 Operation Steps

Figure 2 shows the main steps involved in the execution of GT2 jobs that have been submitted from a GT3 client.

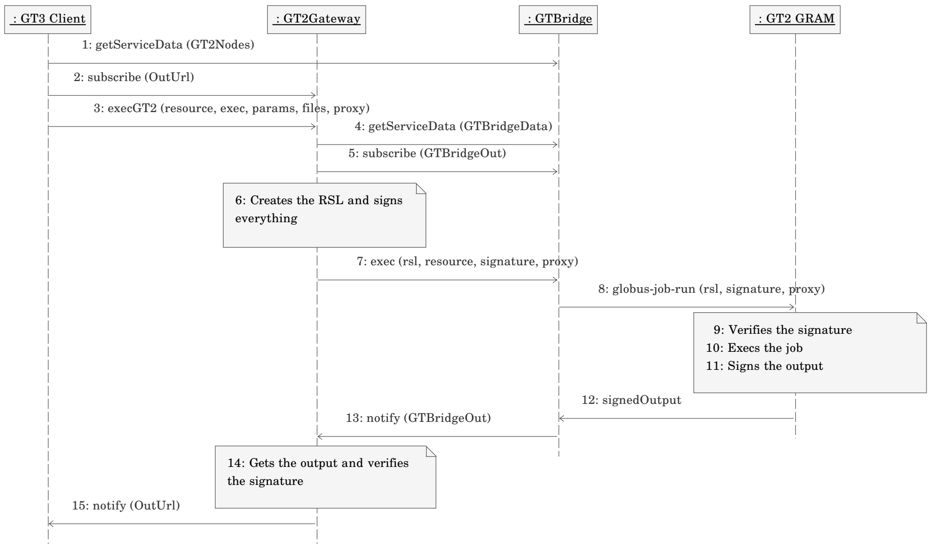
: GT3 Client
: GT2Gateway
: GTBridge
: GT2 GRAM
1: getServiceData (GT2Nodes)
2: subscribe (OutUrl)
3: execGT2 (resource, exec, params, files, proxy)
4: getServiceData (GTBridgeData)
5: subscribe (GTBridgeOut)
6: Creates the RSL and signs everything
7: exec (rsl, resource, signature, proxy)
8: globus-job-run (rsl, signature, proxy)
9: Verifies the signature
10: Execs the job
11: Signs the output
12: signedOutput
13: notify (GTBridgeOut)
14: Gets the output and verifies the signature
15: notify (OutUrl)

Fig. 2. System interaction

1. A client obtains the information about the available GT2 nodes from the service data elements managed by *GTBridge*. Optionally, the client can delegate to *GT2Gateway* the selection of a specific set of nodes.
2. The client subscribes to the *OutUrl* notification of the *GT2Gateway* service to be notified about how to get the results when the job has been executed.
3. After the selection of nodes, the client sends to the *GT2Gateway* the name of the program, any input files or parameters, an identifier of the selected GT2 node and a proxy certificate generated from the user certificate.
4. *GT2Gateway* gets the URL where *GTBridge* will leave the signed job from the *GTBridgeData* service data to add it to the RSL which it is building.
5. Then, this service subscribes to the *GTBridgeOut* notification of the *GT-Bridge* related to the availability of results derived from the job execution.
6. *GT2Gateway* builds the RSL document that describes the job, including a random identifier which will be used by the GT2 node as a reference for the client session. On the other hand, *GT2Gateway* creates a second proxy from the proxy certificate submitted by the user. Finally, the RSL job description, and its related input files, are digitally signed by *GT2Gateway* making use of the private key associated to the first proxy certificate.
7. *GT2Gateway* uses the *GTBridge* service to request the remote execution in the GT2 node, providing the RSL description, its related files, the digital signature, and the new proxy certificate.
8. *GTBridge* uses the interface provided by the Java CoG Kit to submit the job to the target GT2 node.
9. The target GT2 node checks whether it is processing a signed job (which is specified in the RSL description in order to differentiate unsigned jobs

submitted by other GT2 nodes from GT3-originated jobs). It also verifies that the RSL description is properly signed using the first user proxy. Furthermore, the proxy used for authentication purposes must also be the last element of the trusted chain of certificates presented by the user.

10. Using the grid mapping policy, the user is mapped into a local user, and the job is executed according to that user's constraints.
11. Once the job has finished its execution, the GT2 node digitally signs the standard output and any other output files derived from the job execution with the host private key.
12. Next, the GRAM server sends the signed execution output to the *GTBridge*.
13. *GTBridge* specifies the URL where the signed output is available in the appropriate service data, triggering the notification to *GT2Gateway* Service.
14. Finally, *GT2Gateway* makes use of those service data elements in order to obtain the different outputs.
15. The output will be considered valid after verifying that the related digital signature is valid and that the signer certificate belongs to the GT2 Node. In that case, the GT3 client will be notified about the availability of results.

As we can see from these steps, clients are not aware of any operation related to RSL descriptions or digital signatures. The main goal is to achieve an integration as seamless as possible, which might be replaced by a different implementation in a transparent manner.

4.2 Some Details About the GT3 Services

Our OGSA services are specified using GWSDL, including definitions of some of the different service data elements used for notification purposes. Those services implement some OGSA standard interfaces, such as `NotificationSource` or `Factory`, in order to deal with each request in an independent and persistent manner. Figure 3 shows part of the GWSDL of *GT2Gateway*.

Digital signatures follow the *PKCS#7*[14] standard, since this type of document contains information about the signature, the data being protected, and the certificates composing the verification chain.

4.3 Some Details About the Plug-in for the GT2 GRAM Server

The GT2 GRAM server should be extended for three reasons. First, it must be able to understand the new RSL attributes for digital signature support. Second, it has to verify the different user proxies and digital signatures. Finally, it must deal with the different results derived from job executions, also signing the different outputs in order to protect them from an external modification or forgery. Everything has been implemented as a plug-in for the *Job manager*.

To do this, when the GRAM receives a new job, it checks if it is a signed job. In this case, first of all, the GRAM gets the location (URL) of the signed file from the received RSL document. Then, it makes use of the Globus GASS API

```
<types>
    <xsd:element name="runGT2OutputMessage">
       <xsd:complexType/>
    </xsd:element>
    <xsd:element name="runGT2InputMessage">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="exec" type="xsd:string"/>
                <xsd:element name="params" type="tns:stringArray"/>
                <xsd:element name="resource" type="xsd:string"/>
                <xsd:element name="files" type="tns:stringArray"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</types>

<gwsdl:portType name="GT2GPortType"
       extends="ogsi:GridService␣ogsi:NotificationSource">
    <operation name="runGT2">
        <input message="tns:runGT2InputMessage"/>
        <output message="tns:runGT2OutputMessage"/>
        <fault name="Fault" message="ogsi:FaultMessage"/>
    </operation>
    <sd:serviceData name="GT2Nodes"/>
    <sd:serviceData name="OutUrl"/>
</gwsdl:portType>
```

Fig. 3. `GT2Gateway` service definition.

to fetch this file. Once this is done, the digital signature, the certification chain and the RSL can be verified. Finally, when the job has been executed, the files generated in the execution are signed using the host private key, and included in a new PKCS#7 document. This document is transferred to the *GTBridge* Service using again the Globus GASS API.

4.4 State of the Implementation and Tests

We have tested the architecture allowing users to execute a file compressor using the GT2/GT3 integration service. A GUI to submit jobs to the GT2 node has also been developed, so that the user can specify the GT2 node to send the job, the needed files, and the command to be executed in the target node. Once the execution ends, the user is notified and can get the generated files using the same tool. This test bench is a starting point to prove the feasibility of the proposal, as we intend to extend it to support the new WSRF specification.

5 Conclusions and Future Directions

In this paper we outline the need for a real integration of GT3 and GT2 nodes in order to achieve a progressive shift of current developments to the OGSA framework in a seamlessly manner. We propose an architecture based on intermediate services which make use of digital signature mechanisms and proxy certificates to provide integrity and authentication security services.

We are currently working on the extension of our architecture by adding new operations related to the life-cycle of jobs, such as monitoring, migration, etc.

Furthermore we are currently implementing an automated mechanism for smart selection of GT2 nodes guided by some parameters such as computational load, network bandwidth or economic costs.

Although the new Globus Toolkit version (GT4), based on the new WSRF[7] specification, is in an advanced state of development, our work can be easily adapted to that version, as both GT3 and GT4 are conceptually equivalent, except for some changes of syntax and terminology.

References

1. CoG Kits home page. http://www.cogkit.org.
2. Globus toolkit home page. http://www.globus.org.
3. B. Atkinson, G. Della-Libera, S. Hada, M. Hondo, P. Hallam-Baker, C. Kaler, J. Klein, B. LaMacchia, P. Leach, J. Manferdelli, H. Maruyama, A. Nadalin, N. Nagaratnam, H. Prafullchandra, J. Shewchuk, and D. Simon. *Web Services Security (WS-Security). Version 1.0*, 2002.
4. Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte, and Dave Winer. *Simple Object Access Protocol (SOAP) 1.1*, 2000.
5. Marian Buback, Jesus Marco, Holger Marten, Norbert Meyer, Marian Noga, Peter A.M. Sloot, and Michal Turala. CROSSGRID - Development of grid environment for interactive applications, 2002.
6. R. Butler, V. Welch, D. Engert, I. Foster, S. Tuecke, J. Volmer, and C. Kesselman. A national-scale authentication infrastructure. *IEEE Computer*, pages 60–66, 2000.
7. K. Czajkowski, D. Ferguson, I. Foster, J. Frey, S. Graham, T. Maguire, D. Snelling, and S. Tuecke. From Open Grid Services Infrastructure to WS-Resource Framework: Refactoring & evolution, 2004.
8. F. Donno, V. Ciaschini, D. Rebatto, L. Vaccarossa, and M. Verlatto. The World-Grid transatlantic testbed: a successful example of Grid interoperability across EU and U.S. domains. In *Proceedings of the Conference for Computing in High Energy and Nuclear Physics*, 2003.
9. F. Donno, L. Gaido, A. Ghiselli, F. Prelz, and M. Sgaravatto. DataGrid prototype 1. EU-DataGrid collaboration. In *Proceedings of TERENA Networking Conference*, 2002.
10. D. Erwing, H. Ch. Hoppe, S. Wesner, M. Romberg, P. Weber, E. Krenzien, P. Lindner, A. Streit, H. Richter, H. Stuben, V. Huber, S. Haubold, and E. Gabriel. *UNICORE Plus Final Report*, 2003.
11. I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke. *The Physiology of the Grid. An Open Grid Services Architecture for Distributed Systems integration*, 2002. Draft.
12. I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the Grid. enabling scalable virtual organizations. *Supercomputer Applications*, 2001.
13. J. Gawor, S. Meder, F. Siebenlist, and V. Welch. *GT3 Grid Security Infrastructure Overview*, 2003. Draft.

14. RSA Laboratories. *PKCS#7: Cryptographic Message Syntax, Version 1.5*, 1993. An RSA Laboratories Technical Note.
15. M. Rambadt and P. Wieder. UNICORE - Globus: Interoperability of Grid infrastructures. In *Proceedings of Cray User Group*, 2002.
16. S. Tuecke, D. Engert, I. Foster, V. Welch, U. Chicago, M. Thompson, L. Pearlman, and C. Kesselman. *Internet X.509 Public Key Infrastructure Proxy Certificate Profile*, 2003. Internet Draft.