Dynamic Trust Federation in Grids

Mehran Ahsant[1], Mike Surridge[2], Thomas Leonard[2],
Ananth Krishna[2] and Olle Mulmo[1]

[1] Center for Parallel Computers, Royal Institute of Technology,
Stockholm, Sweden
{mehrana, mulmo}@pdc.kth.se
[2] IT-Innovation Center, University of Southampton,
Southampton, UK
{ms,tal,ak}@it-innovation.soton.ac.uk

Abstract. Grids are becoming economically viable and productive tools. Grids provide a way of utilizing a vast array of linked resources such as computing systems, databases and services online within Virtual Organizations (VO). However, today's Grid architectures are not capable of supporting dynamic, agile federation across multiple administrative domains and the main barrier, which hinders dynamic federation over short time scales is security. Federating security and trust is one of the most significant architectural issues in Grids. Existing relevant standards and specifications can be used to federate security services, but do not directly address the dynamic extension of business trust relationships into the digital domain. In this paper we describe an experiment in which we highlight those challenging architectural issues and we will further describe how the approach that combines dynamic trust federation and dynamic authorization mechanism can address dynamic security trust federation in Grids. The experiment made with the prototype described in this paper is used in the NextGRID project for the definition of requirements for next generation Grid architectures adapted to business application needs.

1 Introduction

A Grid is a form of distributed computing infrastructure that involves coordinating and sharing resources across Virtual Organizations that may be dynamic and geographically distributed [21]. The long-term future of the Grid will be to provide dynamic aggregations of resources, provided as services between businesses, which can be exploited by end-users and application developers to solve complex, multi-faceted problems across virtual organizations and business communities. To fulfill this vision, we need architectures and detailed mechanisms for bringing together arbitrary Grid-based resources, along with other resources such as conventional web-services, web-based information sources and people, in a highly dynamic yet manageable way. At present, this is not possible: it takes a lot of time and effort to implement such a collaboration using current technology. The NextGRID project [2] aims to define the architecture for next generation Grids, and addressing this need for highly dynamic federation is one of its main design goals.

Federating security and trust is one of the most significant architectural issues in Grids. Basic Grid security is based on well-developed mechanisms drawing from a wealth of off-the-shelf technology and standards, and work is now underway to address Grid scalability issues and support policy-based access control. However, trust (i.e. dependency) relationships may be expressed in different ways by each service, and the infrastructure may itself impose additional dependencies (e.g. through certificate proxy mechanisms).

In this paper, we focus on the architectural needs of Grid security to support dynamic federation of trust between Grid services running under different Grid (or non-Grid) infrastructure according to different binding models and policies. We examine relevant off-the-shelf components, standards and specifications including WS-Trust and WS-Federation to federate security services in a usage scenario in the Grid. We describe an experiment to test their use to federate trust between heterogeneous security mechanisms in a business relationship. We analyses this experiment to show that available standards cannot directly address the dynamic extension of business trust relationships into the digital domain. We show that it is possible by combining a trust federation mechanism *and* dynamic authorization to enable dynamic federation of resources based on a short-term, rapidly formed business relationship. We ultimately provide an experimental prototype to evaluate our approach by using a real example scenario based on rapid outsourcing of computation to a service provider in order to meet a deadline, based only on commonplace business-to-business trust mechanisms. In the following of this paper, in section 2 we describe the shortcomings of supporting dynamic Federation in Grids and we will mention why security and trust are the main barriers in this regard. In section 3, we give a Grid usage scenario that allows us to focus on dynamic aspects of Federation in Grids for our experiment. Section 4 introduces off-the-shelf components: GRIA and STS that we use as the starting point for our experiment. Based on these components, an experimental design will be provided in section 5. In section 6, we give an overview of WS-trust and WS-Federation as the current existing relevant specifications and in section 7, we analyse the architectural and standardisation challenges for addressing dynamic trust federation. Section 8 describes our approach to tackle architectural issues. Conclusion and future work are described in section 9.

2 Dynamic Trust Federation and Grids

Today's Grid architectures are not capable of supporting dynamic, agile federation across multiple administrative domains. Federation is possible if all parties use the same software, but to set it up is expensive and time consuming, and thus it is only occasionally cost-beneficial. It is reasonable to ask the question: why has the Grid so far failed to deliver the ability to federate resources in a cost-effective fashion dynamically? We believe there are two main reasons for this:

- Dynamic federation is a holistic property of Grids, but Grid architectures have been formulated in a fragmented way by specialized working groups (e.g. those of the Global Grid Forum [20]).

- Previous Grid visions such as those from the Globus team [21], although en compassing dynamic federation are too high level or too specific to scientific collaboration scenarios, with insufficient attention to business trust.

It is not possible today for different organizations running different Grid infrastructure to support even static federations. For example the GRIP project showed that some level of interoperability is possible between Globus and UNICORE, but that there were fundamental incompatibilities in the security architecture and resource descriptions [22] used by each system. Moreover, it is hard for domains to interact and federate resources even if they run the same Grid infrastructure. The complex negotiations needed to establish certification across multiple sites, establish access rights, open firewalls and then maintain software compatibility are well known and documented [23,24].

Establishing trust relationships, and using them to facilitate resource sharing is one of the most challenging issues in Grids. Dynamic trust establishment and interoperability across multiple and heterogeneous organizational boundaries introduce nontrivial security architectural requirements. The main challenge is to ensure that:

- Trust formation across organizational boundaries is subject to due diligence, usually carried out by humans in their business frame of reference; and
- Trust exploitation (enabling resource sharing on commercial or non-commercial terms) is then automated, so the benefits of a decision to trust can be realized very rapidly.

Current Grids do not support automation, so the number of human decisions needed is large, and federation takes a long time. Current Grids also do not support convenient trust scoping mechanisms, so a decision to trust an actor may involve placing complete trust in them, so the due diligence process is often arduous and time-consuming.

The OGSA v1 document [1] describes a range of security components to support access control and identity mapping for VOs. However, all are based on the existence of services established by the VO to support the necessary interactions (e.g. credential translation and centralized access control policy administration and implementation). These mechanisms assume that a VO is well-established, already fully-trusted by all participants, and has its own (trusted) resources to support the required services. We cannot make pre-assumptions about VO lifecycle or trust relationships between a VO and participating domains. Instead, we must support dynamic evolution of both VO and the trust relationships they are built upon, in a much more flexible way than before, in minutes rather than months, and with minimal (ideally zero) overheads and shared infrastructure [8].

3 A Grid Usage Scenario

To provide a focus for our work a scenario was chosen that has the benefit that the required application technology is already available from the GRIA project [6]. This allowed us to focus on the dynamic trust federation and access control issues for our experiment.

KINO is a leading producer of high-quality video content based in Athens. In the course of their business, KINO has a need to perform high-definition 3D digital video rendering calculations, taking "virtual" 3D scenes and characters and generating high-quality video sequences from them. However, this facility is only needed for a small subset of their work, so KINO cannot justify buying a large computational cluster to run such computationally intensive calculations. We assume that an animator is working on a high-definition video rendering job for a customer. On the day before the deadline, he realizes that there is not time to complete the rendering computations needed using the in-house systems available to him. However, he learns of the existence of some GRIA services for rendering high-definition video, operated by GRIA service providers, and capable of providing the level of computational power required on a commercial basis. (We do not concern ourselves here with how the animator finds out about these services, but focus on the trust federation challenges of using them). The animator tells his supervisor, and they agree that they should outsource the rendering jobs to meet their deadline. To do this, the supervisor must set up an account with one or more service providers, so the animator can submit rendering jobs to them. To meet the deadline, everything must be set up and the jobs submitted by the end of the day, so the animator can collect the output and assemble the final video in the morning. The problem is that the GRIA services require that account holders and users be authenticated via X.509 certificates. However, KINO operates a Kerberos (e.g. Active Directory) domain, and does not have a relationship with a third party certification authority. To get certificates from a trusted third party such as Verisign will take far too long – the only solution is to establish a dynamic VO between itself and at least one service provider.

4 Background

4.1 GRIA

GRIA [6] is a Web Service grid middleware created by the University of Southampton and NTUA in the GRIA project, based on components developed by them in GRIA, in the EC GEMSS [16] and UK e-Science Comb-e-Chem [17] projects. The GRIA middleware was tested using two industrial applications, one of which was KINO's high-definition video rendering application. GRIA uses secure "off the shelf" web services technology and it is designed for business users by supporting B2B functions and easy-to-use APIs. It can easily support legacy applications.

Unlike more "traditional" Grids, GRIA was designed from the outset to support commercial service provision between businesses [7], by supporting conventional B2B procurement processes. The security infrastructure of GRIA is designed to support and enforce these processes, so that nobody can use GRIA services without first agreeing to pay the service provider. The procedure for using GRIA services is summarized in Figure 1:

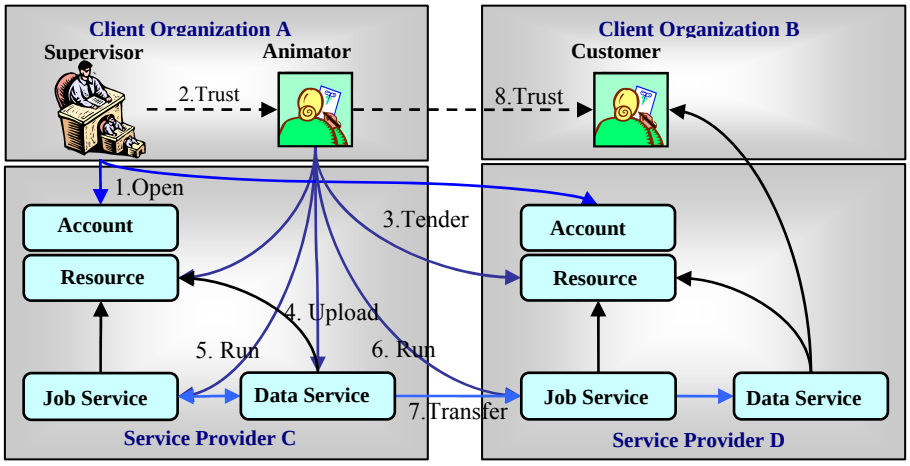

Fig. 1. GRIA usage procedure

Each GRIA service provider has an account service and a resource allocation service, as well as services to store and transfer data files and execute jobs to process these data files. The procedure for using GRIA services is as follows:

1. Account Establishment: First, the supervisor must open an account with the service provider, providing evidence of creditworthiness (e.g. a credit card number) to their account service. If the details are accepted, the service provider will assign an account with a credit limit, to which the supervisor can control access.
2. Resource Allocation: The animator must then allocate resources using the service provider's resource allocation service. This is only possible if the animator has access to an account (the one controlled by their supervisor), to which the resource allocation will be billed.
3. Data Transfer: To transfer data, the animator has to set up a data store using the data service. The animator can only do this if he/she has a resource allocation from which to assign the necessary resources (maximum storage and data transfer volume).
4. Data Processing: To process data, the animator has to set up a job using the job service. This also requires a resource allocation from which to assign the necessary resources (processing time and power). Once the job has been set up, the animator can specify which data stores the job should use for input and output, and subsequently start the job.
5. Data Retrieval: Once the job has finished, the animator can retrieve results from the specified output data store(s), or enable access so their customer can do so.

As indicated in figure 1, it is not necessary for the same person to carry out all these steps. Each service provides methods that allow the primary user to enable access to a (fixed) subset of methods to a specified colleague or collaborator. Thus, the supervisor in Figure 1 can enable their animator to initiate resource allocations charged to the account, and the animator can in turn enable their customer to have

read access to the computational output. This feature is implemented using dynamically updatable access control lists, linked to management operations of the GRIA services through which the corresponding resources are accessed. The GRIA middleware was a convenient starting point for these experiments because (a) it already has a dynamic authorization mechanism, and (b) applications needed for KINO's scenario are already available as GRIA services from the original GRIA project.

4.2 Security Token Service

A Security Token Service (STS) is a Web Service that issues security tokens as defined by the WS-Trust specification. This service can be used when a security token is not in a format or syntax understandable by the recipient. The STS can exchange the token for another that is comprehensible in recipient domain. For example, if the user holds a Kerberos ticket asserting their identity, but the target service needs an X.509 certificate, the Kerberos ticket can be presented to an STS, which will issue the holder with an equivalent X.509 certificate asserting the same identity.

The STS developed for this experiment was specifically focused on Kerberos-PKI interoperability, converting identity tokens only, but is architecturally open and able to handle attributes other than identity and other token formats such as SAML. Our STS implementation is based on a Kerberised Certification Authority (KCA), which issues short-lived user certificates based on the user's Kerberos identity. The KCA has its own certificate signing key, and a long-lived, self-signed CA certificate, which is not widely known. A relying party must trust the KCA's own certificate in order to verify user certificates issued by it. Thus, the KCA does not directly address the problem of establishing trust between domains, but it does provide a good starting point for experiments involving identity mapping and trust federation between domains including a translation between different authentication mechanisms.

5 Experimental Design

In the KINO application scenario described earlier, we assume that the KINO end users are authenticated via Kerberos, while the GRIA service provider requires X.509 authentication. Other, more complex scenarios involving peer-to-peer interactions between Kerberos domains are also possible, but these are not explored in this paper. We used an STS that can be used to convert identity credentials between Kerberos and X.509 representations, and GRIA dynamic authorization to support dynamic extension of trust between the two users and the service provider through dynamic policy updates reflecting the new trust relationship. The two KINO users will use these capabilities to perform the following tasks:

1. The supervisor will open an account with a GRIA service provider, using a credit card to establish KINO's creditworthiness. To do this, the supervisor must present an X.509 certificate, which they get from the STS.

2. The supervisor will enable access to the account for the animator, allowing him to charge work to the account. To do this, the supervisor must specify the identity of the animator granted access to the account.
3. The animator will then allocate resources and submit their rendering jobs. To do this, the animator must present an X.509 certificate, which they get from the STS.
4. The following day the animator will retrieve the rendered video and compose it with other sequences to create the finished commercial.
5. Later the supervisor will receive a statement of jobs and charges to their credit card, giving the details of the user(s) who ran these jobs.

For simplicity, we consider only a single service provider even though it is obviously possible to use the same approach with multiple service providers, at least in a B2B service grid like GRIA. We assume that the credit card used by the supervisor is acceptable to the service provider (up to some credit limit), and that the supervisor is willing to trust the animator to decide how much rendering computation is needed (within that limit) and to submit the jobs. Thus, the three parties (supervisor, animator and service provider) are willing to trust each other sufficiently for the above scenario to be implemented. Our goals are therefore to conduct experiments to answer the following questions:

1. How can the service provider translate business trust (in the creditworthiness of the KINO supervisor) into a trusted digital authentication mechanism based on KINO's "self-signed" certification mechanism?
2. How can the supervisor dynamically authorize the animator to use this trust relationship and access the service provider's rendering service?
3. How can the service provider be sure the animator is the person who the supervisor intends should have access to his account?
4. When the supervisor gets their statement, how can they recognize that the correct person has been running jobs on their account?

Finally, in answering these questions, we also want to establish how these things can be achieved using current and proposed standards, and where (if at all) those standards cannot meet our needs.

6 Review of Standards and Specifications

6.1 WS-Trust

WS-Trust [11] defines a protocol by which web services in different trust domains can exchange security tokens for use in the WS-Security header of SOAP messages. Clients use the WS-Trust protocols to obtain security tokens from Security Token Services. WS-Trust is highly relevant to the question of how to obtain an X.509 certificate for accessing a web service based on a Kerberos-authenticated identity – indeed this is a scenario commonly used to illustrate how WS-Trust works.

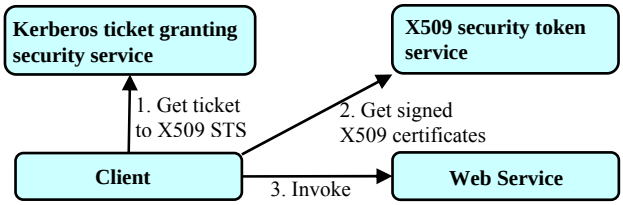

Fig. 2. WS-Trust example using Kerberos and X509

In this example, a client presents a Kerberos ticket granting ticket (obtained when the user logged in to the Kerberos domain) to a ticket granting service, and gets back a Kerberos ticket for an X.509 security token signing service, from which it can obtain a signed X.509 certificate for presentation (e.g. in the WS-Security header) to the target service. WS-Trust defines how the tokens are exchanged (steps 1 and 2 above). However, WS-Trust does not actually provide any mechanisms to manage trust between domains, and only describes token exchange between entities that already trust each other.

6.2 WS-Federation

WS-Federation [11] describes how to use WS-Trust, WS-Security and WS-Policy together to provide federation between security domains. It gives a number of scenarios, starting with a simple example involving two domains, as shown in figure 3.

In this scenario, a client from domain A authenticates itself to its own organisation's Identity Provider (a type of security token service). To use the service in domain B, it needs a different token that will be trusted by that organization. WS-Federation describes the pattern of WS-Trust exchanges needed for this and many other scenarios. However, WS-Federation does not define any standard way to establish this trust relationship dynamically.

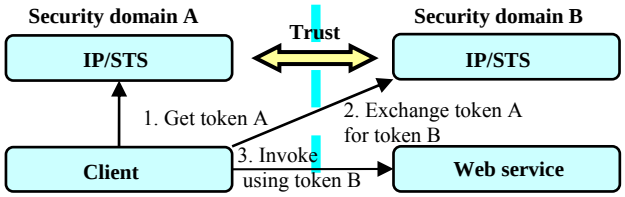

Fig. 3. Usage of WS-Federation between two security domains

According to the specification:

"The following topics are outside the scope of this document:

1. *Definition of message security or trust establishment/verification protocols..."*

Thus, trust relationships must already exist between the WS-Trust token services in a WS-Federation exchange, as indicated in Figure 3. Although these two specifications describe the message exchanges needed, they do not solve the problem of dynamic trust and security federation.

7 Architectural and standardization challenges

The standards and specifications described above cover many aspects of building a secure grid spanning multiple security domains over a public network. However, they leave four major questions unanswered from a Grid architecture and standards perspective.

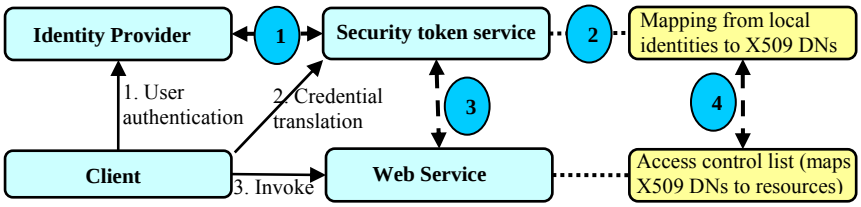

Fig. 4. Key architectural challenges

Our experiments were designed to answer these questions, as indicated in figure 4:

1. How can the security token service guarantee the identity or other attributes of users (the authentication problem)?
2. How does the security token service know what tokens to issue to a user with a given set of home domain attributes (the mapping problem)?
3. How can the web service validate tokens issued by the security token service (the trust problem)?
4. How does the web service know how to interpret the security token issued tokens (the policy problem)?

Some of these questions are unanswered because it is not clear how best to apply the available specifications, and some because the specifications explicitly avoid addressing the question.

8 Approach

In practice, the four questions highlighted above are clearly related. For example, the access control policy used by the target web service specifies the attributes (tokens) required by a user in order to gain access to the service. This policy in effect defines how the web service will interpret tokens presented to it. The mapping used by the security token service to issue tokens to authenticated users must therefore be consistent with the access policy of the target web service.

Thus the web service can only trust the security token service if the mapping used *IS* consistent with its access policy, *AND* it has a way to digitally verify that tokens

claimed to have been issued by the security token service are genuine, *AND* the security token service has accurate information about users when applying its mapping to decide what tokens it can issue.

For example, suppose the web service policy is such that a user identified as *goodguy@kino.gr* can access the service. This implies that security token service will only issue a certificate in this name to a KINO user if they are supposed to be able to access the service. The mapping might be done as the followings:

- supervisor → *goodguy@kino.gr*.
- animator → *goodguy@kino.gr*.
- cameraman → *badboy@kino.gr*.

This is OK if the intention is that the supervisor and animator can access the service but the cameraman cannot. If we now want the cameraman to have access, we can:

- change the internal identity authentication mechanism so the cameraman can authenticate themselves to the security token service as "animator"
- change the security token service mapping so that a user authenticated as "cameraman" can get a certificate in the name of *goodguy@kino.gr*.
- ask the web service provide to change their access policy so *badboy@kino.gr* can also have access to the service.

This is why we decided to combine dynamic access control and trust (attribute) federation and mapping mechanisms and investigate them together. In dynamic security these aspects must remain consistent, so treating them separately will neglect some possible scenarios, and may even be dangerous. Conversely, using them together gives us more options to solve the trust and security federation problems.

Clearly, relationships (1) and (3) in Figure 4 represent critical points in our investigation, since they are the points where one has to validate some action by a remote user. The obvious solution is to co-locate two of the services so that one of these relationships operates within a single trust domain. The normal approach is to co-locate the security token service and the target web service, suggested by Figure 3. This makes it easy to define the meaning of tokens (in terms of the access rights associated with them), and to digitally verify them at the target service. However, when users from a new domain wish to use the service, one must dynamically update the mapping used by the security token service (taking account of the attributes that might be presented by the new users), and create a new digital authentication path (1) between the new user domain and the security token service. Our approach was therefore to place the security token service in the same domain as the client, co-locating the security token service and the user authentication service. This makes it easy to establish the authentication relationship (1) from figure 5 and it means the mapping used by the security token service only needs to handle user attributes from one domain. (It also makes it easy to implement the security token service in a Kerberos domain). Then instead of updating the mapping in the security token service, we can use the dynamic authorization mechanism from GRIA to allow trusted users on the

client side to amend the access control policy (restricted to the resource they control) in terms of the X.509 security tokens issued by the STS.

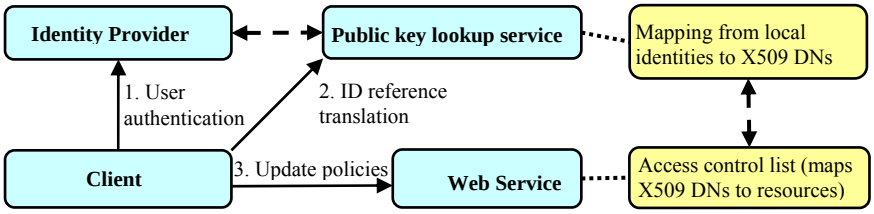

Fig. 5. Dynamic authorization in GRIA

There are still some practical problems to be solved, one of which is shown in Figure 5: to tell the web service (step 3) that a new user is to be authorized, we need to know what (mapped) token that user would have got from the security token service. We therefore need a second service for translating user attributes based on the same mapping. Since the STS is a kinds of CA that issues X.509 identity certificates, the translation service must provide a way to look up the X.509 certificate that would be issued to a specified user. Note that this second service does not sign a public key presented by the requester, as the requester would then be able to claim the attributes specified in the returned token. The token simply allows the requesters to refer to another user's identity in a way that can be recognized later by the target service.

8.1 Dynamic authorization

Dynamic authorization is only needed at the service provider, following the pattern of Figure 5. In our experiment, we relied on the existing GRIA process-based access control (PBAC) dynamic authorization system, but we did consider how this might be used in combination with more generic trust mapping facilities in future. One interesting point is that the dynamic authorization functions are provided by methods of the target service (e.g. *enableAccess*, *disableAccess* on the account service, *enableRead*, *enableWrite* on the data service, etc). This makes sense, because:

- The access policy should refer to capabilities of the service, so dynamic update options available must also be related to capabilities of the service; and
- Access to the dynamic update options should also be regulated by the same dynamic policy, so the full trust lifecycle can be supported in a single architectural mechanism.

It would be possible to provide a generic "dynamic authorization" WSDL port type, using a standard method for requesting more access policy amendments. However, any user who was allowed to access this method would be able to request any policy amendment (not just enabling *badboy@kino.gr* to access their account). One would then need a further, more complex authorization policy regulating what kinds of dynamic policy amendments could be requested by each user. This "meta-policy"

would be quite difficult to generate, since the "target" would be a constraint on some other (potentially arbitrary) policy update request. A more sensible arrangement is to retain the approach used in GRIA, as shown in Figure 6.

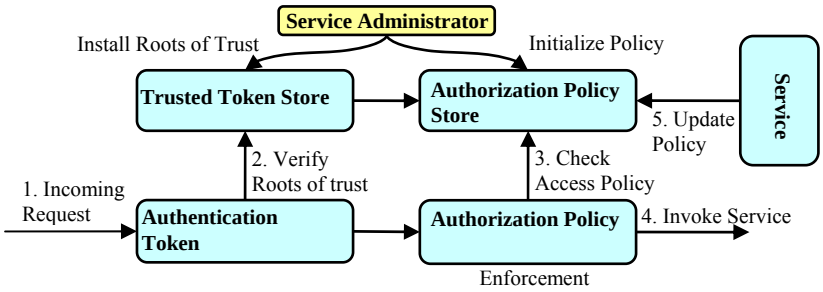

Fig. 6. Dynamic authorization infrastructure

In this approach, the interfaces to the dynamic policy store (3) and (5) should only be accessible to the service provider, so it is not essential that they be standardised, though it would be advantageous for service implementation portability. There are several candidate specifications for checking policy (3) including XACML [26], and IETF Generic AAA [14], but these do not explicitly address dynamic policy updating mechanisms. Obviously, when performing operations like *enableAccess*, the client must specify the scope of the policy change. In GRIA, this is done by sending a context identifier specifying what is to be made accessible (e.g. an account code), as well as a reference token relating to the colleague or collaborator who should be added to the access control list for that context. This approach can also be applied to WSRF resources, and use SAML tokens or X.509 attribute certificates to indicating more general role-based policy updates, of course.

8.2 Implementation and Technical validation

A proof-of-concept implementation of this prototype has been provided for both: the client (Kerberos) side and server (X.509) side. The components that client side contains are: a GRIA client application and middleware, able to send authenticated requests to use GRIA services, A STS, that can supply signed X.509 identity tokens in response to GRIA client and a Public Key Certificate service that can supply X.509 certificates (but not the private keys) for any user in the Kerberos domain.

Having located all the trust (attribute) mapping technology on the client side (inside the client Kerberos domain), the only components we need on the server side is a set of GRIA services for managing accounts and resource allocations, and for transferring and processing data. To validate the modified GRIA implementation, we ran tests between a prospective client in a Kerberos domain (at KTH) and a GRIA service provider established at IT Innovation. A GRIA client application for rendering was released to KTH, and used to run rendering calculations at IT Innovation.

The system worked exactly as expected. A user at KTH was unable to access the GRIA services initially, but they were able to apply for an account. When the service

administrator at IT Innovation approved the account, the service became capable of authenticating credentials issued by the STS inside the KTH domain. The user at KTH was then able to use the account and delegate to colleagues authenticated in the same way, so they could allocate resources and run jobs. The main lessons learned in conducting these tests were as follows:

Previously untrusted users can open accounts, and become trusted if the service provider's checks show that the business risks are acceptable. However, the service provider will then accept connections from other users that have X.509 credentials from the same source as the new user. For example, if a school teacher opened an account using the school's corporate credit card, their students would then be able to make connections to the GRIA service as well. Only the original account holder would be added to the authorization policy of the service, so requests from the students would be rejected unless explicitly authorized by the teacher. However, in principle it would be better to impose some authorization checks at the transport layer as well as the service layer to reduce risks of attack by "malicious colleagues".

Trusted users cannot delegate access rights to users from a currently untrusted Kerberos domain. It is clear that this could be supported by allowing a trusted user to specify a new token source as well as the attributes of their intended delegate. The problem is that it would then be a remote (though trusted) user, rather than a service provider, who approved a decision to trust a new token source. The new user's rights would be tightly specified, but again there could be a risk of "malicious colleague" attack, so service providers may prefer not to delegate such decisions to customers.

Adding the client's STS certificate to the service provider's trust store once business trust is established provides for an efficient implementation. One could use a call-back token authentication mechanism (as in Shibboleth [25]), but that adds an overhead to each subsequent call to the service by the newly trusted user. Note that in a conventional X.509 configuration, a call-back would be needed to check the Certificate Revocation List for the remote user. However, the STS issues short-lived tokens, so the risks associated with infrequent updates of the CRL are much lower than in a conventional PKI. The remote server has to be certified by a "well known" CA that the client already trusts, or else the client cannot risk passing any sensitive information to it. In our test scenario, the supervisor passes a credit card number (or other evidence of creditworthiness) to the GRIA service, so he must be able to authenticate it even if the service cannot at that stage authenticates him except through the validity of the card number. Thus, it is necessary to hold a set of "well known" CA certificates in a trust store, while simultaneously updating the client's key pair and associated certificate. It is not normally appropriate to attempt to use simultaneous bi-directional trust propagation – at least not using the mechanisms tried here.

9 Conclusion and future work

Dynamic resource federation is an obvious requirement of next generation Grid architecture, to address the need for short-term virtualization of business relationships to address transient opportunities and deliver short-term goals. Our studies have been based on a practical (if small scale) scenario from KINO, which is driven by a tran-

sient, short-term business need. The main barrier to dynamic federation over short time scales in such scenarios is security. We have examined relevant standards and specifications including WS-Security, WS-Trust, WS-Federation, etc. These can be used to federate security services, but do not directly address the dynamic extension of business trust relationships into the digital domain.

Our analysis of specifications shows that dynamic trust federation and dynamic authorization (access control) are intimately coupled aspects of dynamic security federation on the Grid. The mechanisms used to federate trust (i.e. authenticate attributes and tokens) are quite different from those needed to enforce access control policies. However, both aspects must be consistent, and in a dynamic federation scenario, this means they need to be changed through some concerted procedure. On the other hand, the fact that dynamic federation can be achieved through a combination of the two mechanisms offers a wider range of options for implementing federation mechanisms. Our analysis suggests that trust (e.g. identity) mapping should normally be performed by the domain in which the identity (or other) attributes are assigned to users, while the consequences are defined in the target domain by using dynamic authorisation mechanisms to update the policy for the target service. This is not the pattern traditionally seen in WS-Federation, but uses the same specifications.

We developed an experimental Grid prototype based on trust mapping technology used by KTH (STS) and a Business-to-Business Grid middleware (GRIA) that includes dynamic authorization support. The experimental prototype shows that by combining trust federation and dynamic authorization, one can enable dynamic federation of resources based on a short-term, rapidly formed business relationship.

The next step will be to formalize the architectural concepts used to achieve this as part of the NextGRID next generation Grid architecture. A more general reference implementation of these concepts is now being produced within the NextGRID project, and will be made available to the community and incorporated in a future release of the GRIA middleware, and possibly other NextGRID compatible Grid middleware in future.

10 References

1. The Open Grid Services Architecture. Released on http://www.gridforum.org as v1.0 in July 2004.
2. EC IST Project 511563: *The Next Generation Grid*.
3. The NextGRID Architecture Straw Man. Available from the NextGRID website at http://www.nextgrid.org.
4. Heimdal Kerberos 5 Derrick J Brashear, Ken Hornstein, Johan Ihren, et al, http://www.pdc.kth.se/heimdal/heimdal.html
5. Web Services Security X.509 Certificate Token Profile Chris Kaler, et al, 1st March 2004. http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0.pdf
6. EC Project IST-2001-33240 Grid Resources for Industrial Applications. See the GRIA website at http://www.gria.org for the current GRIA middleware version.

7. Surridge, M., Taylor, S. J. and Marvin, D. J. (2004) Grid Resources for Industrial Applications. In Proceedings of 2004 IEEE International Conference on Web Services, pages pp. 402-409, San Diego, USA.
8. Surridge, M., Taylor, S. J., De Roure, D and Zaluska, E. J. (2005), Experiences with GRIA - Industrial applications on a web services Grid. To appear in Proceedings of 1st IEEE Conference on e-Science and Grid Computing, Melbourne, Australia, Dec 2005.
9. RFC3820, see http://www.ietf.org/rfc/rfc3820.txt.
10. Web Services Security (WS-Security) v1.0, Chris Kaler, April 2002. http://www-106.ibm.com/developerworks/webservices/library/ws-secure/
11. Web Services Trust Language (WS-Trust) v1.1, Steve Anderson, Jeff Bohren, et al, May 2004.
12. Web Services Federation Language (WS-Federation), Chris Kaler and Anthony Nadalin, July 2003. http://www-106.ibm.com/developerworks/webservices/library/ws-fed/
13. Debunking SAML myths and misunderstandings, Frank Cohen, IBM developerWorks, 08 July 2003. http://www-106.ibm.com/developerworks/xml/library/x-samlmyth.html
14. The IETF has published generic AAA specifications as RFC2903 (architecture) and RFC2904 (framework). See http://www.ietf.org/rfc/rfc2903.txt and http://www.ietf.org/rfc/rfc2904.txt.
15. IETF draft at http://www.ietf.org/internet-drafts/draft-ietf-pkix-certstore-http-08.txt.
16. GEMSS project website at http://www.gemss.de.
17. Comb-e-Chem project website at http://www.comb-e-chem.org.
18. Shibboleth v 1.2.1 Scott Cantor, Steven Carmody, Marlena.Erdos, et al. h
19. D. De Roure et al. 2002. The semantic Grid: a future e-Science infrastructure. http://www.semanticgrid.org/documents/semgrid-journal/semgrid-journal.pdf
20. The Global Grid Forum website is at http://www.gridforum.org.
21. I. Foster, C. Kesselman, S. Tuecke., *The Anatomy of the Grid: Enabling Scalable Virtual Organizations*, http://www.globus.org/research/papers/anatomy.pdf.
22. J.Brooke, K.Garwood and C.Goble, Interoperability *of Grid Resource Descriptions: A Semantic Approach*, see http://www.semanticgrid.org/GGF/ggf9/john/.
23. See the work of the TERENA Task Force on *Authentication, Authorisation Coordination for Europe*, via http://www.terena.nl/tech/task-forces/tf-aace/
24. V Welch, *Globus Toolkit Firewall Requirements*, See http://www.globus.org/security/v2.0/firewalls.html.
25. Shibboleth v 1.2.1 Scott Cantor, Steven Carmody, Marlena.Erdos, et al. http://shibboleth.internet2.edu/shibboleth-documents.html
26. eXtensible Access Control Markup Language (XACML) Version 2.0 draft 04, Tim Moses, Dec 2004. See http://www.oasis-open.org/committees/xacml.