On-demand Restricted Delegation

A Framework for Dynamic, Context-Aware, Least-Privilege Delegation in Grids

MEHRAN AHSANT

Doctoral Thesis
Stockholm, Sweden 2009
Akademisk avhandling som med tillstånd av Kungl Tekniska högskolan framlägges till offentlig granskning för avläggande av teknologie doktorsexamen i datalogi måndagen den 16 februari 2009 klockan 13:00 i Sal F3 Flodis, Lindstedtsvägen 26, Kungl Tekniska högskolan, Stockholm.

© Mehran Ahsant, February 2009

Tryck: Universitetsservice US AB
Abstract

In grids, delegation is a key facility that can be used to authenticate and authorize requests on behalf of disconnected users. In current grid systems, delegation is either performed dynamically, in an unrestricted manner, or by a secure but static method. Unfortunately, the former compromises security and the latter cannot satisfy the requirements of dynamic grid application execution. Therefore, development of a delegation framework that enables a restricted and flexible delegation mechanism becomes increasingly urgent as grids are adopted by new communities and grow in size. The main barriers in development of such a mechanism are the requirements for dynamic execution of grid applications, which make it difficult to anticipate required access rights for completing tasks in advance.

Another significant architectural requirement in grids is federated security and trust. A considerable barrier to achieving this is cross-organizational authentication and identification. Organizations participating in Virtual Organizations (VOs) may use different security infrastructures that implement different protocols for authentication and identification; thus, there exists a need to provide an architectural mechanism for lightweight, rapid and interoperable translation of security credentials from an original format to a format understandable by recipients.

This thesis contributes the development of a delegation framework that utilizes a mechanism for determining and acquiring only required rights and credentials for completing a task, when they are needed. This is what we call an on-demand delegation framework that realizes a bottom-up delegation model and provides a just-in-time acquisition of rights for restricted and dynamic delegation.

In this thesis, we further contribute the development of a credential mapping mechanism using off-the-shelf standards and technologies. This mechanism provides support for an on-the-fly exchange of different types of security credentials used by the security mechanisms of existing grids.

**Keywords:** Grid Security, Restricted and Context-Aware Delegation, Delegation Protocol, On-demand Delegation, Dynamic Trust Federation, Grid Interoperability, Credential Mapping
To my beloved wife, and

my parents whom I always adore ...
Acknowledgments

First and foremost, I would like to thank Jim Basney from the National Center for SuperComputing Applications, University of Illinois at Urban-Champaign, whose contributions were invaluable. I feel privileged to have worked with him and I am grateful for his support, encouragement and patience; his personal and professional insights; technical feedback; and comments on every part of my research and my publications. I also thank him for reading and commenting this thesis.

Thanks to Professor Lennart Johnsson, my advisor, for taking me on as a graduate student and for supporting my work; for his feedback on my research and for providing the financial support that was vital to my studies.

I am very grateful to Olle Mulmo, who during my first two years as a doctoral student, worked closely with me and introduced me to the area of grid computing and security. He opened many doors for me to the grid world and helped me to establish several helpful connections with experts in this field.

I would also like to thank Åke Edlund for his support during my doctoral studies at PDC and for letting me work on this thesis while I worked in the BalticGrid-II project.

Some parts of the research presented in this thesis were done in collaboration with other researchers from different projects. In particular, I would like to show my gratitude to Mike Surridge, Thomas Leonard, Ananth Krishna, E. Rowland Watkins and Joris Claessens, all of whom worked with me in the NextGrid project; Joni Hahkala, Martijn Steenbakkers and Akos Frohner, all of whom worked with me in the EGEE project. Thanks to Adam J. Lee who contributed to one of my publications. I would also like to thank Esteban Talavera González, Masashi Nakamura and Sina Khaknezhad, who helped with implementation and integration parts of the work presented in this thesis. Finally, thanks to Rachana Ananthakrishnan and Frank Siebenlist from the Argonne National Laboratory for their feedback and support in the PURSe project.

I would like to show my gratitude to Erwin Laure and Kristaps Džonsons, who read and commented on drafts of this thesis. In particular, Kristaps, who took a lot of time and helped with the proof-reading of this thesis.

I also acknowledge the help and support in the last stages of my doctoral studies by Professor Johan Hästad, Olof Ronborg and the director of graduate studies, Professor Jens Lagergren.
I am also very grateful to my friends Ali Ghodsi and Vahid Mosavat for their supports and consultations, especially Ali for reading and commenting on this thesis.

I would also like to take this opportunity and express my profound gratitude to my parents and all of my family for their continuous support and encouragement. In particular, Soheila, who has never withheld her support from me since I have been in Sweden.

Finally, my deepest gratitude and special thank to my gorgeous wife, Azadeh Jamshidi, for her love and support, for always being with me during difficult times, for always showing endless patience during bad times, and for her continuous encouragement.

I also acknowledge the projects that provided financial support for my research presented in this thesis: the EGEE project, funded by the European Union (contract IST-2003-508833); the NextGrid project co-founded by the European Commission (contract IST-2006-034567); the BalticGrid-II project funded by the European Union (contract 223807); and the SweGrid project founded by the Swedish Research Council (contract 343-2003-953).
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contents</td>
<td>ix</td>
</tr>
<tr>
<td>List of Figures</td>
<td>xi</td>
</tr>
<tr>
<td><strong>1 Introduction</strong></td>
<td>1</td>
</tr>
<tr>
<td>1.1 Problem statement</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Approach</td>
<td>3</td>
</tr>
<tr>
<td>1.3 Thesis organization</td>
<td>4</td>
</tr>
<tr>
<td><strong>I Background and Results</strong></td>
<td>5</td>
</tr>
<tr>
<td><strong>2 Foundations</strong></td>
<td>7</td>
</tr>
<tr>
<td>2.1 Grids</td>
<td>7</td>
</tr>
<tr>
<td>2.2 Grid security</td>
<td>7</td>
</tr>
<tr>
<td>2.3 Delegation</td>
<td>8</td>
</tr>
<tr>
<td>2.4 Ontology</td>
<td>9</td>
</tr>
<tr>
<td>2.5 Workflows</td>
<td>10</td>
</tr>
<tr>
<td>2.6 Summary</td>
<td>11</td>
</tr>
<tr>
<td><strong>3 Background</strong></td>
<td>13</td>
</tr>
<tr>
<td>3.1 UNICORE delegation</td>
<td>13</td>
</tr>
<tr>
<td>3.2 GSI delegation</td>
<td>14</td>
</tr>
<tr>
<td>3.3 gLite delegation</td>
<td>15</td>
</tr>
<tr>
<td>3.4 Discussion on current delegation approaches</td>
<td>16</td>
</tr>
<tr>
<td>3.4.1 Security vs. flexibility</td>
<td>16</td>
</tr>
<tr>
<td>3.4.2 Dependency to communication protocol</td>
<td>17</td>
</tr>
<tr>
<td>3.4.3 Interoperability</td>
<td>17</td>
</tr>
<tr>
<td>3.4.4 Observing and auditing</td>
<td>18</td>
</tr>
<tr>
<td>3.5 Summary</td>
<td>18</td>
</tr>
<tr>
<td><strong>4 Thesis contribution</strong></td>
<td>19</td>
</tr>
<tr>
<td>4.1 On-demand delegation framework</td>
<td>19</td>
</tr>
</tbody>
</table>
5 Results

5.1 Summary of thesis contribution papers ........................................ 31

Paper I: Toward An On-demand Restricted Delegation Mechanism
for Grids .................................................................................. 31

Paper II: Context-Aware, Least-Privilege Grid Delegation ............. 31

Paper III: Workflows in Dynamic and Restricted Delegation ...... 32

Paper IV: Grid Delegation Protocol .............................................. 32

Paper V: Security Credential Mapping in Grids ............................ 32

Paper VI: Dynamic Trust Federation in Grids ............................... 33

5.2 Summary of other papers ......................................................... 33

Paper VII: Streamlining Grid Operations: Definition and Deploy-
ment of a Portal-based User Registration Service ..................... 33

5.3 Software .................................................................................. 33

5.4 Related Work ........................................................................... 35

5.5 Conclusion ............................................................................... 40

5.6 Future Work ............................................................................ 41

Bibliography ................................................................................. 43

Appendix I: Delegation ontology in OWL/XML notation ............ 55

II Papers ....................................................................................... 63

Paper I: Toward An On-demand Restricted Delegation Mechanism
for Grids .................................................................................. 65

Paper II: Context-Aware, Least-Privilege Grid Delegation ............. 69

Paper III: Workflows in Dynamic and Restricted Delegation ...... 73

Paper IV: Grid Delegation Protocol .............................................. 77

Paper V: Security Credential Mapping in Grids ............................ 81

Paper VI: Dynamic Trust Federation in Grids ............................... 85
List of Figures

Paper VII : Streamlining Grid Operations: Definition and Deployment of a Portal-based User Registration Service

List of Figures

4.1 An example of on-demand delegation .................................. 20
4.2 Delegation ontology ......................................................... 23
4.3 All asserted classes of delegation ontology ............................ 24
4.4 Architectural components of on-demand delegation framework .... 25
4.5 A general use-case of exchanging security tokens using CMS .... 28
Chapter 1

Introduction

The capacity and performance of Internet communication channels are increasing immensely; therefore, it is becoming feasible to solve large-scale computational problems on a single virtual computer consisting of computers connected over the Internet. “Grid Computing”, where virtual computers are composed into “grids”, is a paramount approach to make this possible [47, 44].

In grids, delegation is a key facility allowing effective use of a wide range of dynamic grid applications [8]. When a grid user makes use of a remote resource to e.g. execute a job, that resource may in turn need access to third-party resources (e.g. data repositories) on behalf of the user in order to complete the task. Such access may possibly span across multiple security domains. In such scenario, delegation can be used to delegate (parts of) the user’s rights to the remote resource such that it in turn can access the necessary third parties [46]. In the rest of this thesis, some terminologies in regard to “delegation” are used frequently for which a basic and general definition can be given as the followings:

- **Delegation** is the act of transferring rights and privileges to another party (the Delegatee).

- **Delegatee** is the delegation target. It is the entity that the Delegation Credential is delegated to.

- **Delegator** is the entity that delegates the abilities and/or rights to the Delegatee.

- **Delegation Credential** is the desired result of delegation. It is a message conveying the abilities and rights from the Delegator to the Delegatee. Depending on the security infrastructure and delegation model used, the actual syntax and contents of Delegation Credentials might be different, however they are typically integrity protected and digitally signed.

The “least privileges” principle, which states that delegation should enable a delegator to grant only those rights required for performing a delegated task, is becom-
CHAPTER 1. INTRODUCTION

One of the most important security aspects regarding delegation. Unfortunately, there are security risks associated with performing delegation where delegated rights are not limited solely to the task intended to be performed within a limited lifetime and under restricted conditions. In a general sense, if unrestricted delegation credentials are used, an increase of sites included in the trust domain corresponds to greater probability for security holes’ occurrence. This increases potential damage brought by possible attacks and stolen credentials. In extensive environments such as grids, the probability of a host losing its trustworthiness by some irregular actions (e.g. be compromised) is likely to be (much) higher than in a highly confined and uniformly controlled environment. Therefore, performing a fine-grained delegation in a restricted and secure fashion is one of the most significant concerns of grid security, a concern not yet completely resolved.

Dynamic trust establishment and interoperability across multiple and heterogeneous organizational boundaries introduces nontrivial security architectural requirements. Grids are heterogeneous environments and therefore establishing dynamic trust relationships and using them to facilitate resource sharing becomes a challenging issue.

1.1 Problem statement

In the context of grid system delegation, there is a conflict between flexibility and security: applications must choose between limited or full delegation. On one hand, delegating a restricted set of rights reduces exposure to attack, but also limits the flexibility/dynamism of the applications; on the other hand, delegating all rights provides maximum flexibility, but increases exposure. Delegating fewer rights than are required for completing a task may fail the task execution, while delegating more rights than are needed may threaten abuse by malicious parties. Supporting restricted delegation has become a challenging issue, as dynamic execution of grid applications makes it very difficult to predict the set of rights required by a delegatee to complete the task on behalf of its delegator.

Furthermore, implemented delegation mechanisms are mostly tied to the underlying communication protocols and supported solely by particular security infrastructure. These hinder grid system interoperability and may force grid participants to use particular security infrastructures or especially a particular authorization mechanism for their back-end.

Interoperability in grids is also a significant concern, and one barrier to achieve it is the heterogeneity of security protocols used for authentication and identification. Organizations participating in grids may use diverse underlying security mechanisms or different grid security infrastructures, such as Public Key Infrastructure (PKI), Kerberos, or SAML. A user may hold a security credential, one from his local domain, which can not be used in relying domains (a domain that receives an assertion from an issuing party). For example, a recipient...
domain may be expected to receive a Kerberos ticket when it does not support Kerberos functionality and can only understand PKI-based certificates; similarly, it may be the case that the issuing and relying domain use the same authentication mechanism, however, they are missing a trust establishment. One solution is requiring users to collect a set of security credentials in different formats and/or as issued by different trust anchors. However, this obviously is not a practical, cost-beneficial and possible solution for users and organizations.

Cross-organizational authentication is clearly a challenging issue, and a credential mapping mechanism is required to address this issue. Providing such a mechanism enables grid organizations to collaborate with verifiable and understandable security credentials regardless their locally-established security mechanisms. The lack of such a mechanism may either cause the authentication to fail at any point of interaction, such as when accessing resources, or may limit the collaborations only between those organizations with fully compatible security mechanisms.

This thesis solves these problems by defining, developing and evaluating a framework that provides support for both a restricted delegation and a credential mapping mechanism. The framework developed in this thesis enables performing fine-grained and restricted delegation in a flexible, standard and interoperable manner. It also provides support for exchanging credentials for different kinds of security tokens used by current grid security infrastructures.

1.2 Approach

In this thesis we introduce a novel solution for addressing the least privileges principle of delegation in dynamic, distributed environments, which no existing solution adequately addresses. We define and develop an on-demand delegation framework, which utilizes a callback mechanism for acquiring corresponding credentials required for completing tasks on-demand. This approach addresses the shortcomings of current existing delegation mechanisms by providing restricted delegation in a secure, flexible, and interoperable fashion as needed for a wide range of dynamic grid applications.

This thesis further contributes a mechanism for a lightweight, rapid and interoperable translation of security credentials. We develop a Credential Mapping Service (CMS), which leverages a simple and standard messaging protocol for exchanging different kinds of security credentials. This greatly helps to address the issue of cross-organizational authentication and enables grid entities to collaborate with verifiable and understandable security credentials regardless of their local installed security mechanisms. The implementation of this service uses off-the-shelf standards and technologies and supports the exchange of different kinds of security

---

1 The term *we* is used throughout this thesis to denote the work lead and performed by the author while collaborating with other researchers. Where there are joint contributions, the parts done by the author are explicitly stated.
tokens used by existing grid security infrastructures, such as PKI, Kerberos and SAML.

1.3 Thesis organization

This thesis is divided into two parts. The first part summarizes the background, approach and results of our work. The second part consists of thesis contribution papers and one additional paper. Chapter 2 describes underlying technologies, concepts and foundations in the context of grids. Chapter 3 describes and discusses current delegation mechanisms and their shortcomings, which this thesis addresses. Chapter 4 describes the contribution of this thesis in addressing the problem. Finally, Chapter 5 presents results, related work, future work and a short description of software components developed as part of this thesis.
Part I

Background and Results
Chapter 2

Foundations

This chapter describes the foundational concepts and theory of the work presented in this thesis. We describe grids as a new paradigm of distributed computing environments. We further describe why security is a challenging issue in the context of grids and discuss how delegation mechanisms enable dynamic execution of grid applications. We briefly explore the emerging “Semantic Web” and “Semantic Grid”, and we describe how they are used in grids to achieve a high degree of easy-to-use and seamless automation. We finally describe scientific workflows and discuss the potential of scientific and grid computing for accommodating workflows.

2.1 Grids

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. A Virtual Organization (VO) is a group of individuals and/or institutions/organizations with a common purpose or interest who agree on policies for the sharing of resources in order to facilitate achieving common goals. Each VO has its own policies for access, use, management and monitoring [88, 47, 44].

2.2 Grid security

Grids are in place to enable Virtual Organizations (VOs) for cross-organizational interactions. VO members need to interact with each other. This interaction may span diverse security realms and traverse different services and hosting environments. VOs, participating members of VOs and resource owners each have their own specific rules and policies that govern how to access particular grid resources. VOs are built over heterogeneous organizations with different underlying security mechanisms. Furthermore, due to the dynamic nature of VOs there is no assumption of pre-established trust between organizations and the VOs or members of VOs.
CHAPTER 2. FOUNDATIONS

New services may be deployed and instantiated dynamically over a VO’s lifetime. There is no traditional means of security administration to update a centralized policy databases or issue new credentials to access these new services. Therefore, in a general sense, cross-organizational interactions through scalable and dynamic VOs make security a very challenging issue [88, 46, 122]. Effective management of grids therefore requires mechanisms for dynamic and robust trust establishment among VOs’ participants. Delegation and credential mapping, among many others, are two significant facilities for expanding trust relationships among VOs, which are the concerns of this thesis.

2.3 Delegation

Delegation may happen anywhere in daily activities. The Cambridge dictionary\(^1\) describes the verb “delegate” as:

“to give a particular job, duty, right, etc. to someone else so that they do it for you”

This simple description gives the main objective of delegation as getting a task or activity done by other parties. In delegating, a person assigns the authority and responsibility to another person for performing specific activities or tasks; it may either be shifting a decision-making authority to a subordinate or re-assigning a task or activity to another person. In both cases, delegation encompasses the obligation to perform the task, authority to authorize accomplishment of the task and responsibility for the possible consequences of an accomplished task [116].

In the context of IT system, diverse models of delegation exist. As an example, batch systems for job execution read job scripts and create processes on the system on behalf of the submitter to execute the jobs. Similarly, mail client programs have permission to write mails to an inbox that is owned by an end-user.

In the context of grid security, delegation is an important facility for expanding trust relationships. It enables members of virtual organizations to endow to other entities (other members/computer programs) the ability to access and manage resources on their behalf [88, 122, 49]. In grids, resource request and use may cover extensive periods. Furthermore, requests may be generated dynamically during the execution of an application and need user approval before being submitted to a resource broker or resource provider. Resource providers usually require some form of authentication of users and authorization of requests, i.e. a user is allowed to use the requested resources as described by the request. However, the user may not be directly accessible either because of malfunction or simply by having disengaged after submission of a request for execution. A common solution to the problem of disconnection is to delegate the authorization to an agent (program) that acts on the user’s/application’s behalf and that is less likely to be disconnected at any time.

\(^1\)http://dictionary.cambridge.org/
Although delegation plays an important role in expanding trust relationships, the level of established trust might be different in diverse contexts. In non-grid applications such as mail programs mentioned before, the system administrator authorizes the server processes to perform actions on behalf of users. This can be accomplished by running a process with more privileges than a normal user. This strategy is applicable within a single administrative domain and is possibly an effective and efficient approach to leveraging various extensions, for example, complex collections of roles, access control lists, and time-expiring access tickets. In fact, end-users of such systems trust local system administrators and the process with more privileges to act on their behalf. Similarly, although grid delegation is used to expand trust relationships, jobs accessing resources on behalf of end-users are running on remote sites where the end-users cannot keep control over jobs’ behavior and detect malfunctions.

2.4 Ontology

In computer science, ontology is a conceptual schema to hierarchically describing and classifying entities and their relationships and rules within a certain domain. There are two kinds of ontologies, domain ontology and foundation or upper ontology. The former corresponds to a specific domain and represents the particular meanings of terms as they apply only to that domain; the latter is a model of the common objects that are generally applicable across a wide range of domain ontologies and describes general entities. In practice, a foundational ontology may be a glossary of terms used in a certain descriptive language such as a programming or modeling language. This enables creating a more specialized schema to make the data useful in making real world decisions. In this sense, ontology was described as “an explicit and formal specification of a conceptualization” by Gruber [108].

A well-known and widely used upper computational ontology is Dublin Core [58], which describes digital objects using meta-data elements such as title, creator and subject. As an example, all computer programs have a foundation ontology consisting of a processor instruction set, programming language, files in accessible file systems and so.

Recently, several ontological standards have emerged in order to describe World Wide Web content. The Semantic Web\textsuperscript{2} is an extension of the current Web, applying explicit representation of knowledge to Web content in a manner understandable by machines. It allows more sophisticated use of the World Wide Web in order to solve problems rather than simply to retrieve information, which is its primary model of Web use today. Well-defined ontologies are required in order to formalise the relationships among web content; without them, the degree of association error rises. Semantic WEB and ontologies have strong potential in representing and reasoning over policies in the World Wide Web, distributed systems [106, 65, 107, 112, 83] and, recently, in grids.

\textsuperscript{2}http://www.w3.org/2001/sw/
The Semantic Grid\(^3\) is an emerging technology to add Semantic Web capabilities to the grid computing applications. Scientific processes are usually very complicated and encompass many computational steps. Each computational step may require resources from different organizations that might be represented by different models and terminologies. Resources are usually domain specific and problem dependent, and therefore an effective realization of grid computation to achieve the best effect of resources require an explicit exposure and representation of resources in a common knowledge model. Many publications have demonstrated that realization of a Semantic Grid is the key to delivering the real potential of grids and e-science [90, 36, 111].

2.5 Workflows

Workflows traditionally have been used by business organizations for modeling and controlling the execution of business processes to achieve a business objective [69]. In workflow literature, each business process can be defined and modeled as follows:

1. a collection of tasks or activities which it encompasses;
2. the applications that should perform each task; and
3. the data required for performing tasks.

A workflow separates any given organizational process into a set of well-defined tasks, related and inter-dependent, as logical steps or descriptions of pieces of work that contribute toward the accomplishment of the whole process. In this context, a Workflow Management System (WFMS) is the unit to coordinate the execution of tasks that constitute a workflow. Generally speaking, tasks in a workflow are inter-related such that initiation of one task is dependent on successful completion of another set of tasks. The order in which tasks are executed are controlled by the WFMS [14].

Recently, workflows have emerged as a paradigm for representing and managing complex distributed scientific computations and accelerating the progresses of scientific activities. For scientific applications, a workflow paradigm can be beneficial from many aspects, such as building a dynamic application for orchestrating distributed systems, utilizing resources, reducing execution costs and even promoting inter- and intra-organizational collaboration [48, 23]. Similar to business workflows, scientific workflows are also concerned with the automation of processes, and enable the composition and execution of complex scientific tasks. In scientific workflows, each step specifies a process or computation to be executed (for instance, a software program or a Web service); the workflow links the steps according to the data flow and corresponding dependencies. Processes contain tasks that are structured based on their control and data dependencies.

\(^3\)http://www.semanticgrid.org
2.6 Summary

In this chapter, we described some of the foundational concepts on top of which this thesis has been built. Earlier, we described grids and elaborated on why security is such a challenging issue. We described the role of delegation in addressing security issues in terms of authentication and authorization, then explained how delegation can be used in expanding trust relationships in grids and why it is essential for using a wide range of dynamic grid applications. We described ontologies and how they can be used in describing hierarchical resources used by complicated scientific processes. We also described workflows as a new paradigm for representing and managing complex scientific computations. In the following chapter, we first describe how delegation is implemented and used by current existing grids. Then we discuss the shortcoming of existing approaches.
Chapter 3

Background

In this chapter, we describe the background of our work. We start by giving three examples of exiting delegation mechanisms implemented by the UNICORE\(^1\), Globus\(^2\) and gLite\(^3\) grid infrastructures. Our choice of these grid middleware systems is because they are widely used and well-established, and fully implement core grid functionality. Furthermore, from the security perspective, each system approaches delegation in a different way. It should also be noted that in this chapter we only describe the original model of delegation developed by these grid infrastructures: there have been subsequent improvements and enhancement explained as related works in section 5.4.

In the remainder of this chapter, we first describe each approach individually. We further consider the pros and cons of each approach and how the shortcomings of these mechanisms can be addressed.

3.1 UNICORE delegation

UNICORE (Uniform Interface to Computing Resources) provides a ready-to-run grid system and makes distributed computing and data resources available in a seamless and secure way. UNICORE supports a task-based grid security model. In this model, when a job is created using the Job Preparation Agent (JPA), the user must specify in advance the actions to be performed, the resources needed and the system upon which the job is to run. From this job description, an Abstract Job Object (AJO) is created that instantiates the class representing UNICORE’s abstract job model. AJOs can describe diverse tasks, such as computation, job management, file transfer and even some complex functions such as loops and conditionals. The AJO is signed with the end-user’s certificate and then is sent to a

\(^1\)http://www.unicore.org/
\(^2\)http://www.globus.org/
\(^3\)http://glite.web.cern.ch/glite/
Gateway for execution. Multi-site job execution and delegation in UNICORE is enabled by means of Sub-AJOs that are created from a parent AJO [57].

In the UNICORE security model, an “endorser” is an end-user who holds the ownership of an AJO by signing the AJO. The “consigner” is an agent that transfers the AJO to the server. It is either the end-user or another UNICORE server. “Endorsing” is the act of signing a task description by an end-user on the client site; “consigning” is the act of transferring an AJO from an end-user to a server or from a server to another server. An end-user is allowed to do both “endorsing” and “consigning” of AJOs, whereas a server can only “consign” AJOs. This means that server processes cannot create sub-AJOs dynamically. Servers do not have the ability to create grid processes dynamically, since all components of an AJO have to be pre-signed with the users’ certificate. In UNICORE, all the AJOs received by sites must have been created by end-users and cannot be modified by intermediary servers. Once an AJO is received, the server checks that the endorser has properly signed the AJO. Following this, the server performs local site authorization, ensuring that the endorser is allowed to execute jobs on the site and is mapped to a local account.

This approach requires a minimal trust relationship between the sites (since the end-user has explicitly authorized the file transfer request). There is, however, no support for dynamic grid functions (e.g., reading a database from a source whose location is only discovered during execution) since the entire job tree must be signed by the end-user at the client when the job is created [57, 49].

3.2 GSI delegation

The Globus Security Infrastructure (GSI) [46] is based on the Grid Security Architecture introduced in [79]. The GSI is used to manage mutual authentication between the local and remote machine, and authorization on the remote systems. The GSI infrastructure performs delegation by means of “proxy certificates” [56]. A proxy certificate is defined as an X.509 certificate [110] issued by an end-user to a process or another entity acting on the end-user’s behalf. Remote sites with support for GSI can interpret a proxy certificate as an authority of owner to perform a task on behalf of an end-user. In the GSI delegation approach, the owner of proxy certificate will be granted all rights that the end-user possesses. Proxy certificate holders further can issue other proxy certificates to other entities [121].

GSI provides delegation capability as an extension of the standard TLS protocol [40]. A proxy certificate still includes the identity of its end-user: this allows establishing SSL connections with remote sites. GSI binds a temporary private key to each proxy certificate for performing authentication. In order to support Single Sign-On [46] the private key associated with the proxy certificate is typically stored unencrypted, protected through only normal file-system mechanisms. The private key cannot be encrypted, as it must be used by a delegatee (processes that act on behalf of the end-user) without the need for entering a password by the
end-user [121].

In fact, what a GSI proxy mechanism provides through a simple implementation of proxy certificates is a full impersonation of end-users by granting all rights to a subordinate for performing tasks. In the Globus approach, in order to establish the required trust relationships, the possession of a proxy certificate is sufficient to authorize work at remote sites. Any site accepting GSI delegations must trust that the originating site and all other sites in the delegation chain are properly managed and not compromised. The GSI approach imposes a transitive trust requirement on all sites participating in a particular grid (members of a particular VO). Thus the delegation protocol needs to be carefully designed because the entities (both end-users and processes) issuing proxy certificates must be careful to authenticate the identity of the process requesting the proxy so that delegated rights are given to the correct process.

### 3.3 gLite delegation

gLite is a grid middleware system providing a framework for building Internet-wide grid applications. gLite is the largest grid computing middleware to date and is developed as part of the EGEE Project [68].

gLite describes delegation as a standalone Web Services portType and provides ready-to-use library implementations of this portType. Service implementers do not need to deal with the details of the delegation mechanisms and can factor this functionality out of the internal application logic. Delegation can also be instantiated as a standalone service, front-ending a (trusted) credential repository from which the target service can extract delegated credentials in a controlled and secure manner.

The java-delegation security component of gLite [59], implements delegation by leveraging proxy certificates [110] and using a simple request-response interaction protocol between a Delegatee and a Delegator. The gLite delegation protocol was originally based on the G-HTTPS protocol described in [78], which is also used in building the delegation interface of GridSite, the Grid Security for the Web platforms for Grids.

In the gLite delegation components, the operations “get-proxy-request” and “put-proxy-cert” respectively implement the request and the response interactions of the delegation protocol, which are described in the followings:

1. GET-PROXY-REQUEST: A Delegator may prompt a Delegatee to perform delegation. Delegator uses the GET-PROXY-REQUEST method allowing a Delegatee to prompt a Delegator to generate an X.509 certificate request and key, and then return the certificate request to the Delegator. This request
includes a Delegation-ID, an alphanumeric string used to identify the resulting delegation session. The Delegation-ID is used to distinguish between several delegations from the same user, using several attributes and expiration times. The server then generates an X.509 certificate request and private key, and stores the private key for retrieval using the Delegation-ID.

2. PUT-PROXY-CERT: This method is used to return the proxy certificate to a Delegatee after receiving a successful response to a GET-PROXY-REQUEST by a Delegator. The request may include a Delegation-ID header with the same delegation ID header as sent to the server in the GET-PROXY-REQUEST. Before responding, the server stores the signed certificate and chain associated with the private key the server has generated, the client’s GSI identity and the Delegation ID if present.

gLite delegation provides dynamic delegation due to the fact that it uses proxy certificates as the basic means when performing delegation. This enables a grid user to delegate all (or some subset) of their privileges to another entity in a limited-time interval as described in Section 3.2. It is also dynamic in terms of involving new entities in a delegation process: in addition to persistent services and entities, delegation of privileges can be provided to services that are created dynamically or do not hold any form of identity credential. A common scenario is that a user submits a job to a computational resource and wants to delegate privileges to the job to allow that job access to other resources on the user’s behalf (e.g., to access data belonging to the user on other resources or start sub-jobs on other resources). However, as described for the GSI delegation mechanism, this protocol still fails in providing a restricted delegation that can fulfill the requirements of the least privileges principle.

3.4 Discussion on current delegation approaches

We subsequently identify and address four significant concerns of delegation as the follows:

Security vs. flexibility

One the most significant concerns of current delegation mechanisms is their shortcomings in flexibly and dynamically addressing restricted and secure delegation. In general, Globus approaches a very dynamic delegation mechanism by fully impersonating end-users and granting all rights to subordinates for performing a task. Unfortunately, an unencrypted private key associated to a proxy certificate yields a GSI proxy certificate that is valuable and relatively easy to acquire and consequently abuse.

The GSI and gLite solutions, which use proxy certificates, implement a very dynamic delegation mechanism because there is no need to know in advance execu-
tion details or resources required. However, it increases the danger of unauthorized acquisition or usage of proxy certificates. Issuing short-lived proxy certificates has been the basic and primary solution used by GSI for addressing this shortcoming. However, even a short-lived proxy certificate not restricted to a specific task can cause serious dangers to security systems. A complementary approach, limiting the usage of X.509 proxy certificates, is using the Proxy Certificate Information (PCI) extension [56]. This extension enables issuers to express their expected delegated rights and to limit the number of proxy certificates that can be further issued by a Proxy Certificate holder.

In the UNICORE security model, delegated rights are restricted solely to the task for which delegation has been launched. Although the description allows some flexibility (e.g., conditional execution of parts of the job and a generic request to run anywhere suitable), there is insufficient flexibility to adapt this mechanism to the changing circumstances as required by many grid applications [49, 87, 57].

Thus, there is a conflict between addressing the flexibility and security in the context of delegation. This conflict originates in the fact that execution of grid application is highly dynamic: anticipating required access rights in advance and adapting them to policy statements for accessing resources during task execution is not straightforward [9].

**Dependency to communication protocol**

In current delegation mechanisms, delegation credentials have been closely tied to the underlying authentication and communication protocols, e.g., as an optional step performed after a TLS handshake [40] as done in the first incarnations of GSI [46] or in UPL (UNICORE Protocol Layer) [57]. However, this may break the compatibility with some communication protocols such as TLS and subsequently any protocol on top of TLS, such as HTTPS [89]. Hence, a delegation framework must separate delegation from authentication at least for non-legacy software components [8]. A well-established delegation framework can make this possible by defining and implementing a stand-alone Web Services delegation portType. For this service, required supports can be embedded in the service container/application server (as a separate and standalone service), or in the application itself (by inclusion of the delegation’s portType in the service description and helper libraries that implement the operations exposed by that portType) [8].

**Interoperability**

Current security mechanisms used by existing grids are not completely interoperable; despite that interoperability is becoming one of the most significant concerns of grid security [87]. Delegation is implemented and used in different ways by diverse grid middleware systems [34, 121, 49, 59]. Organizations may also use different technologies and security infrastructures (e.g., Kerberos [80], PKI [29] or SAML [100, 120]). Thus, assuming that current in-place security mechanisms will
continue to be used, different delegation mechanisms implemented by different grid systems need to interoperate.

**Observing and auditing**

A robust delegation mechanism should enable keeping track of delegated tasks and accurate usage of delegated rights. By approaching current delegation mechanisms, delegators can easily lose their control over delegated rights during the task execution; further, misbehavior can not be detected immediately [49, 9]. Therefore, providing a means of enabling extensive and fine-grained monitoring and logging of delegation credentials is an essential need that is not addressed by existing mechanisms.

**3.5 Summary**

In this chapter, we reviewed the delegation mechanism developed and deployed by UNICORE, Globus and gLite, to date the biggest and the most widely-used grid systems. We described the shortcomings of their delegation mechanisms and discussed some of the most important issues in terms of security, flexibility, dependability and interoperability. Our discussion was aimed at making a strong context for the following chapter, in which we describe the contribution of this thesis in addressing the shortcomings of other approaches.
Chapter 4

Thesis contribution

This thesis contributes a novel delegation framework for grid environments that addresses all of the shortcomings described in Chapter 3. By leveraging the foundations described in Chapter 2, we have built a delegation framework that not only meets the requirements of the least privileges principle, but also enables performing delegation in a standard, secure and flexible manner.

4.1 On-demand delegation framework

On-demand delegation is a novel solution providing a secure, restricted and fine-grained delegation mechanism fitted for a wide range of dynamic grid applications. This solution provides a delegation framework in which a delegatee, in order to act on behalf of a delegator, needs to obtain required rights (in terms of additional delegated credentials) at run-time. On-demand delegation allows delegators to delegate privileges when the other party has proved the necessary need for those privileges. This implies delegating rights to delegatees iteratively as needed until the task has completed. Papers I and II give a detailed description of this approach.

The intuition of the on-demand delegation model is a novel concept of delegation that we call bottom-up delegation. It is well explained when compared to a traditional model of delegation in which a top-down model is approached for delegating rights from a superior to a subordinate in advance before a delegatee starts off a delegated task. In contrast, in bottom-up delegation a subordinate needs to ask a superior for acquisition of sufficient rights when it needs to perform an action on behalf of a delegator. Although top-down delegation models have been used for a quite long time by many security systems, for some use-cases, like those described earlier, they are insufficient and there is a need for a bottom-up model that enables delegating rights just-in-time and in response to a valid request [9].

Figure 4.1 illustrates a simple example of performing on-demand delegation in a single organization. As depicted in this figure, the Delegator initiates delegation by providing a minimum set of rights to the Delegatee (Step 1). This minimum set
only allows the Delegatee to prove that it needs to act on behalf of the Delegator. The Delegatee has no rights to access resources and therefore needs to query the delegation system model to determine what additional credentials are required to complete the task. By this query, the required privileges for executing the task are determined and disclosed to the Delegatee during the job’s execution (Steps 2 and 3). The Delegatee contacts one or more Delegation Services to determine what credentials are needed and then obtains them (Step 4). In this simplest case, there is one Delegation Service associated with the Delegator and one associated with the Resource. The Delegation Service checks requests against a set of policies established for each specific delegatee that specify the circumstances for the issuance of additional credentials (Steps 5). Hidden from this picture is the process of establishing contexts. The context establishment is the process to collect all information required to assert a delegation request. These information are converted into a single internal format understandable by the Policy Engine. The Policy Engine uses a combination of delegation policies and context information to approve any delegation request: if the policies allow issuance of the requested credentials, the Delegation Service generates the credential and sends it to the Delegatee (Step 6). The Delegatee now has all required delegated credentials to access resources on behalf of the Delgegator (Step 7). A concrete example of a real grid usage has been given in Paper I Section V and Paper II section III.

On-demand delegation requires a delegatee to contact a superior in order to ask for additional rights. This gets more parties involved in the delegation process, which raises some requirements for this model as follows.

- First, there is a need for setting up a process or a service (i.e. a delegation service) that can automatically delegate a superior’s rights to a delegatee
4.1. ON-DEMAND DELEGATION FRAMEWORK

Without getting the superior involved.

- A delegation service is a machine and not a human being; it is, therefore, crucial to set up a policy engine that enables the service to decide if the subordinate should have the requested rights. Such established policies are essential for enabling a superior to maintain complete control over delegation.

- It is important to resolve how, in such framework, a delegation request can be approved and how a subordinate can be trusted to the truth about which rights is required and why.

- It is important to figure out how to determine required access rights and credentials when accessing resources at run-time, and how to request and obtain them from a delegation service.

- Finally, this approach needs to be reliable for delegators such that a delegation service never delegates more rights than required. It should also have the potential to optimize delegation, reducing the number of callbacks when performance is a concern and too many callbacks are required for completion of a task.

Building blocks

In the section, we describe the building blocks of an on-demand delegation framework and explain how they are used in addressing the above requirements. A detailed description can be found in Papers II, IV and V.

- **Delegation service**: A delegation service is a web service for delegating credentials to potential requesters (delegatees). It is run by a user or hosted on behalf of many users. A delegation services can delegate credentials based on a given delegation specification attached to the request. Along with the request, a delegation service requires the context information in which additional delegated rights are requested. It must also query the policy engine to check if delegation policies can be fulfilled in a particular context, permitting the delegation of additional rights. A delegation service is mainly the architectural component to address the first requirement mentioned above. Papers I and II describe in detail how a delegation service is used in an on-demand delegation framework.

- **Context manager**: A context manager is a service that is either notified by other services with new context information or that makes a query to directly obtain the context information of particular services, resources or job instances. This context information, along with the delegation request, is then sent to (or directly requested by) a delegation service in order to create an appropriate context in which a delegation request may be verified. Context information may be created to verify a delegation request that is made to
obtain additional rights for the sub-jobs created by a parent job. A context manager would typically be hosted within the service from which it obtains the information.

Contexts are established by monitoring, collecting and processing the status of jobs, resources and services. They evolve through the task life-cycle and are considered as any characterizing information about protected resources and surrounding environments. Contexts are basically associated with: i) resources to be controlled; ii) delegates who make requests to access protected resources; and iii) delegators on whose behalf access to resources are made. Paper II describes in detail how “contexts” are created, established and leveraged in an on-demand delegation framework. A context manager is mainly used to address the second and the third requirements mentioned above.

- **Policy engine:** A policy engine is required to set up delegation policies. It is used by delegation services to check a delegation request against a set of established delegation policies specified by local administrators and/or delegators. A policy engine is mainly used to address the second and the third requirements described above. Papers I and II explain in detail how a policy engine is used in an on-demand delegation framework.

- **Credential exchanging protocol:** An on-demand delegation framework requires a protocol for requesting and receiving delegation credentials as needed. The callback mechanism leveraged by this framework requires a flexible and efficient protocol enabling the dynamic exchange of delegation credentials between delegatees and delegation services over heterogeneous security domains. The Grid Delegation Protocol (GrDP) provides such communication protocol for exchanging delegation of rights and abilities in terms of delegation credentials. The design of GrDP is based on the WS-Trust [126] specification which is not bound to a particular security mechanism and it is therefore applicable for exchanging different kinds of security tokens of common use supported by grid environments. Papers I and IV describe in detail how this protocol is defined, developed and deployed for an on-demand delegation framework.

- **Delegation ontology:** Delegation ontology is used to describe how “delegation” allows access to resources and how delegated credentials can be provided upon receiving a delegation request [9, 71]. Ontologies are important to on-demand delegation in the following general aspects:

  - to be used in system description, which enables both fine- and coarse-grained authorization mechanism to protect resources; and
  - to be used in dynamic establishment of delegation policies, which protect the exposure of delegated rights and privileges.

Delegation ontology describes that each “Delegation” enables a “Delegator” to endow a “Capability” to a “Subject” under restricted conditions. “Creden-
4.1. ON-DEMAND DELEGATION FRAMEWORK

"Potential" is the means by which "Delegation" is authorized. Each "Capability" contains one or more "Verbs", which can be accomplished on one or more "Objects". Each "Capability" may have some dependencies on other capabilities that implies a hierarchical delegation taxonomy in system description and consequently the need for additional delegation credentials.

Figure 4.2 is a simple illustration of delegation ontology. This picture only illustrates the most important and relevant classes defined for delegation ontology without depicting the arrangement of these classes in a taxonomic hierarchy. It also shows defined properties and allowed values for these properties. Hidden from this picture are the values for these properties that have been filled in for instances and a knowledge base created by defining instances of these classes. On the service provider side, instances of the delegation ontology can also be used to describe the system in terms of "Objects", "Verbs" and the "Capability" that makes an appropriate relation between objects and verbs. It may also be used to specify individual instances of a "Delegator", who delegates access rights to the instances of class "Subject". Individual instances of class "Credential" can also be used to describe how the authority of the owner for accessing resources can be approved. On the Delegator side, individual instances of delegation can be used for establishing delegation policies and further specifying how credentials should be issued and appropriate constraints be applied on them. Figure 4.3 is a detailed and hierarchical
Delegation ontology is mainly used to address the forth requirement mentioned above. Paper I describes in detail how delegation ontology is defined and leveraged in an on-demand delegation framework. Appendix I depicts the delegation ontology deployed for this framework in OWL/XML notation.

- **Workflows**: A workflow gives a high-level and structural specification of processes used by Work Flow Management Systems (WFMS) to define the way that tasks should be ordered, scheduled and executed. The benefits of using workflow management systems in an on-demand delegation model are their support for defining, structuring, executing and controlling processes (job definitions), which encompass tasks. The task dependency, process structure and execution path requirements of a WFMS are what a bottom-up delegation model leverages to ensure that delegation is performed in a “reliable” and “optimized” way.

To achieve a reliable delegation we introduce the notation of Delegation Safety Invariant (DSI) to assuring the following: i) privileges can be delegated either at the time of launching tasks or during the execution (just-in-time), ii) no privileges can be delegated from a superior to a subordinate unless they are restricted to an intended delegated task (bound-to-task), and iii) no privileges can be delegated for accessing a resource unless they are less/equally powerful as the privileges held by the superior for accessing the same resource or executing the same task (limited-to-boundaries).

To achieve an optimized delegation we introduce the Risk of Delegation (RoD) parameter that can be set by security administrators or individual delegators to specify the level of security risks and threats when delegating rights. It gives the ability to adjust risk factors based on the current threat landscape and enforces the security model to operate accordingly. A higher value for this parameter indicates a higher level of security risks and misbehavior when delegating rights; a lower value implies less security risks and threats associated with delegation. The RoD parameter controls how restricted privileges can be delegated to potential delegatees. It also determines how often during a workflow execution a delegatee is required to ask (via making call-backs) for more rights. For example, if delegation is performed in safe and restricted environments to highly trusted delegatees, the value of RoD can be set to a lower
4.1. **ON-DEMAND DELEGATION FRAMEWORK**

level which allows providing more rights via a fewer number of callbacks; and in case of stronger potential of security threats and risks, a lower value can be set to the RoD parameter to enforce using a delegation model which requires a larger number of callbacks and more restricted range of delegated rights. This enables performing delegation in different level of efficiency and restrictions. In Paper III, based on the concept of ROD parameter we present a formal definition and develop appropriate algorithms for three different models of bottom-up delegation.

Figure 4.4 decouples the on-demand delegation framework into its architectural components described earlier. It also explains how these components are related and should interact with each other. What Figure 4.4 illustrates are the components of an on-demand delegation model in a generic grid infrastructure. The generic architectural components (white boxes) illustrated in this picture can be provided by many of currently-used grid systems; the components represented by gray boxes are what on-demand delegation provides to a grid infrastructure. In Paper II, we give a concrete design of this generic architecture for a real grid system and we describe how the components with generic names are replaced by the real components in Globus which is one of the most widely used grid infrastructures.

![Figure 4.4: Architectural components of on-demand delegation framework](image)

**On-demand delegation strengths**

The strengths of this approach, particularly those that address the shortcomings described earlier in Chapter 3 and those that fulfill the requirements explained above, are as follows:

- **Fine-grained delegation**
  
  This framework provides support for generating delegation credentials with a very limited and well-defined range of capabilities or policies. A delegator is
able to implicitly or explicitly entitle a delegatee solely a set of restricted and limited rights for the task to be performed.

- **Dynamic determination and provisioning of required rights**

On-demand delegation mechanisms enable dynamic determination of required rights and utilizes a callback mechanism for requesting and acquiring additional rights at run-time. Thus, there is no need to know about required rights in advance.

- **Context-aware delegation**

On-demand delegation framework is capable of establishing “contexts” in which the need for additional delegated rights can be verified. Supporting contexts addresses the requirements of active security systems: in active security systems, any assigned permission might be activated or deactivated when its associated context is evolving. Operations are then permitted if the associated permission is currently activated. In such security systems, access control models can distinguish between permission assignment and activation by considering different levels of “context” when processing an access operation on an object [27, 52, 55, 103].

- **Uniformly performing delegation**

On-demand delegation framework provides a protocol for requesting and exchanging delegation credentials when they are needed by delegatees to complete tasks. The GrDP protocol implements a flexible security token exchange protocol enabling requester and delegation services to interact dynamically over heterogeneous security domains¹.

- **Exploitation of delegation Ontology**

The resource description and associated delegation specification are utilized for dynamic determination of access rights and delegation requirements when accessing resources. In order to support an automatic process for establishing delegation policies and dynamic disclosure of required access rights, this framework provides a formal and abstract description of the concept of delegation that can be instantiated for any administrative domain.

- **Integration with workflows**

¹In 2005 and in order to harmonize delegation between different grid middlewares and projects we established a "Task Force" group and invited partners from different projects and organizations who have been experiencing delegation in their products such as: Globus, EGEE and GridSite. The preliminary goal set to gain a consensus on a single set of syntax and semantics for delegation based on the WS-Trust specification and creating a strawman document for discussion in a wider community/standard body. In order to solicit comments from the grid security community the idea was presented at the thirteenth Global Grid Forum (GGF13) in the WS-Delegation AdHoc BoF session.
4.2. SECURITY CREDENTIAL MAPPING

Using workflows in an on-demand delegation framework ensures to meet the requirements of DSI factor and increases the level of reliability of this approach in a sense that delegated privileges are not more than required for executing a process (job) and delegation is always bound to intended job(s). Using workflows we also presented three different models of bottom-up delegation: Per-step delegation, Multi-step delegation and One-Step delegation. These models provide delegation in different level of efficiency and restriction based on the current threat landscape specified by the RoD parameter described earlier.

- Observability and monitoring

In on-demand delegation frameworks, a delegator must be able to establish relevant delegation policies at the time of initiating tasks. These policies govern if the request for additional rights and privileges can be granted to a delegatee (specific task/program) at run-time. This enables a delegator to keep full control over delegated rights and privileges during the task execution and to take appropriate action in response to any potential misbehavior or unexpected situation.

4.2 Security credential mapping

We identified cross-organizational authentication and identification as a significant and challenging issue in grids: members participating in different VOs need to interact with each other. The requests for accessing grid services may cross organizational boundaries with different underlying security models. One part of this thesis contributes a credential mapping mechanism that enables grid organizations to collaborate with verifiable and understandable security credentials regardless of their locally-established security mechanisms.

This thesis addresses this problem with a mechanism for on-the-fly exchange of different types of security credential profiled by OASIS\(^2\), including Kerberos [80] and X.509 proxy certificates [110]. Our work presented in this thesis is the only full-mesh and standard credential mapping mechanism that has been developed as a lightweight, easy-to-integrate and open-source service for grids. In this work, we only deal with authentication token mapping that is in general about the syntax aspects of security credentials. Other aspects of “cross-organizational” interoperability, such as authorization and attribute mapping, are covered in collaboration with other researchers as it is described in Paper VI.

Previously, we mentioned the development of a standalone delegation service for issuing security tokens as defined by the WS-Trust specification. Using such a standard and flexible interface as proposed by the WS-Trust specification, and adding some more functionality, we have built a service capable of exchanging security tokens for different formats or syntax. In this case, the service will function

\(^2\)http://www.oasis-open.org/specs/
as a Credential Mapping Service (CMS) in our framework and can be used to address the cross-organizational authentication challenge described earlier. As a usage example, if a user holds a Kerberos ticket asserting his identity and needs to be authenticated by a target service that needs X.509 certificates for authentication, the Kerberos ticket can be presented to CMS, which then issues the holder with an X.509 certificate asserting the same identity in a different format. CMS developed in our framework provides a full translation between security credentials common in grids in addition to those profiled by the OASIS standard body\(^3\). This includes UsernameTokens [77], X.509 Certificates [56], SAML Assertions [100], Kerberos tickets [80] and X.509 proxy certificates [110].

Figure 4.5 depicts a design diagram of a general mapping scenario using the CMS service. In this framework, we assume that a prospective user, who is willing to request CMS for exchanging a security token, holds a valid credential obtained from his local identity provider. In follows, we describe the steps of this mapping as depicted in Figure 4.5:

1. The User authenticates with his local identity provider in a domain in which he is already registered. As the result of this authentication, the User obtains an authentication token from the local identity provider (IdP). This is basically a native authenticator token generated by the local authentication mechanism.

2. The requester (the user or an entity on behalf of the user) generates a request message. Depending on the exchanging scenario requested, the requester attaches all the “Claims” required by CMS.

3. The requester signs, encrypts and sends the message to CMS. How a request is signed and encrypted may differ for each particular mapping scenario and the way that the trust relationship is established between the local IdP and the CMS. We elaborate this when we describe each particular scenario separately.

\(^3\)www.oasis-open.org/
4. CMS receives the message, verifies the signature and decrypts it. If the verifi-
cation is successful, it checks the request against existing policies and checks if
all required “Claims” are attached to the request. It then uses an appropriate
mapping mechanism (a plug-in) to generate the requested credentials.

5. If CMS is already configured to use a third-party attribute provider, it re-
quests for a set of attributes to be incorporated in a newly-generated security
credential.

6. The generated credential then will be packed, encrypted, signed and sent back
to the requester.

7. The requester receives the response message, verifies the CMS’s signature,
decrypts the message, extracts the generated credential from the message
and stores that in a configured credential storage.

In this section, we briefly described a generic model of credential mapping archi-
tecture. Papers V and VI give a detailed explanation and design diagram for each
particular mapping scenarios. They further describe some real grid usage scenarios
in which credential mapping has been used.

4.3 Summary

In this chapter, we briefly described the contribution of this thesis. We introduced
an on-demand delegation framework that provides a novel approach for performing
delegation in grid environments. We described the building blocks, components,
technologies and protocols for achieving such a framework and explained how on-
demand delegation addresses the shortcomings of existing delegation mechanisms.
An implementation of this framework for a real grid usage scenario based on the
Globus environment is fully described in Paper II. We further described our contri-
bution in providing a credential mapping mechanism developed for addressing the
cross-organizational grid authentication.

Part II of this thesis includes a number of papers that give a more detailed expla-
nation of our approach. In regard to building an on-demand delegation framework
more information can be found from Papers I, II, III and IV. More elaboration on
credential mapping mechanism can be found from Papers V, VI and VII.
Chapter 5

Results

In this chapter, we present a summary of the papers contributed in this thesis. We also give a short description of the software components developed and implemented for building the framework described earlier. In brief, Papers I, II, III, and IV give a detailed description of the on-demand delegation framework and Papers V, VI and VII give a detailed description of the credential mapping mechanism and its usage in some real grid scenarios. In this chapter, we further summarize related works and give a conclusion of this thesis and highlight some works for future studies.

5.1 Summary of thesis contribution papers

Paper I: Toward An On-demand Restricted Delegation Mechanism for Grids

This conference paper\(^1\) identifies the shortcomings of current delegation mechanisms in grids and highlights the existing conflict in addressing restricted delegation in current grid security systems. It further proposes to approach delegation on-demand by leveraging a call-back mechanism, in order to achieve a secure and flexible restricted delegation in grids. The paper presents the concept of delegation ontology and describes the potentials of on-demand delegation framework in using a delegation ontology.

Paper II: Context-Aware, Least-Privilege Grid Delegation

This conference paper\(^2\) recognizes the need of supporting “contexts” for establishing delegation policies in an on-demand delegation framework. In this paper, we

\(^1\)This paper was published in Proceedings of the 7th IEEE/ACM International Conference on Grid Computing, Barcelona, September, 2006.

\(^2\)This paper was published in Proceedings of the 8th IEEE/ACM International Conference on Grid Computing, Austin, Texas, September, 2007.
introduce the concept of “active delegation” in our framework for enabling a just-in-time acquisition of delegated rights in an “associated context”. We describe the architectural components of the on-demand delegation framework and explain the implementation and integration of this framework in a real grid usage scenario based on the Globus grid infrastructure.

**Paper III : Workflows in Dynamic and Restricted Delegation**

This conference paper\(^3\) describes the integration of workflows with an on-demand delegation framework. In this paper, we describe the development of three different models of bottom-up delegation as: One-step, Multi-step and Per-step delegation that enable delegating rights in different levels of efficiency and restriction based on the current threat landscape indicated by the Risk of Delegation (RoD) parameter. The notation of Delegation Safety Invariant (DSI) is also presented in this paper. This paper uses some formal notations of a standard RBAC authorization model and a graph-based workflow model to define, analyze and evaluate these delegation models.

**Paper IV : Grid Delegation Protocol**

This paper\(^4\) highlights the lack of a standard delegation interface and further proposes a standard and interoperable protocol for performing delegation in grids. The GrDP protocol introduced by this paper, approaches delegation in grids through a standard and interoperable way based on the WS-Trust specification. This paper describes how the GrDP protocol can operate across diverse security realms and organizations for performing delegation in an interoperable and independent manner.

**Paper V : Security Credential Mapping in Grids**

This conference paper\(^5\) describes the development of a credential mapping mechanism for grids. The paper highlights the significant challenge of cross-organizational identification and authentication in grids as one of the main barriers to dynamic trust federation over short time scales. The paper describes the architecture and development of a credential mapping mechanism, based on off-the-shelf standard specifications and technologies. The paper describes the design diagram and the implementation of this framework for different credential mapping scenarios.

---

\(^3\)This paper will be published in Proceedings of the 4th International Conference on Availability, Reliability and Security (ARES/CISIS 2009), Fukuoka, Japan, March 2009.


\(^5\)This paper will be published in Proceedings of the 4th International Conference on Availability, Reliability and Security (ARES/CISIS 2009), Fukuoka, Japan, March 2009.
5.2. SUMMARY OF OTHER PAPERS

Paper VI : Dynamic Trust Federation in Grids

This conference paper\(^6\) investigates on trust federation and mapping mechanisms for managing aggregated security and trust relationships in dynamic virtual organizations. The work presented in this paper describes a joint collaboration with other researchers to design and develop a complete solution for achieving a dynamic trust federation in grids. Our contribution to the work presented in this paper has been carrying out a state-of-the-art analysis of trust federation and credential mapping mechanisms and their applicability to the architecture proposed for the next generation of grids. In this work we integrated our credential mapping mechanism with a dynamic authorization framework, contributed by the co-authors, for enabling dynamic federation of resources based on a short-term, rapidly formed business relationship.

5.2 Summary of other papers

Paper VII : Streamlining Grid Operations: Definition and Deployment of a Portal-based User Registration Service

This journal paper\(^7\) describes PURSe, a Portal-based User Registration System (PURSe) and credential management tool for grid users. The tool is aimed at providing end-users with an easy-to-use, semi-automatic, yet secure way of obtaining and managing credentials necessary to access grid resources. The tool is implemented as a set of highly customizable components suitable for portal integration. Our contribution to the work presented in this paper has been the improvement of its security design and development of corresponding software components that enable PURSe to support external Certification Authorities at back-end. We have also extended the functionality of PURSe by integrating this portal with our Credential Mapping Service (CMS)\(^8\).

5.3 Software

For both the on-demand delegation framework and the credential mapping mechanism contributed in this thesis, we have provided a detailed architectural design and a prototype implementation of appropriate software components to demonstrate

\(^6\)This paper was published in Proceedings of The 4th International Conference on Trust Management, Italy, May, 2005.

\(^7\)The first edition of this paper was published in Proceeding of the Workshop on Grid Applications: from Early Adopters to Mainstream Users In conjunction with GGF14, (June 27, Chicago, USA). The second edition of this paper was published in Journal of Grid Computing, Volume 4, Number 2 / June, 2006.

\(^8\)The development of software components which integrate PURSe with CMS was accomplished in Google Summer of Code 2008 program (GSoC 2008) and the result are available from the Google Code web site.
the feasibility of our approach. The followings are the most important software components developed for these frameworks:

- **Delegation Service** is a kind of WS-Trust security token service [76, 126] that can be used by any hosting domain in on-demand delegation framework for providing delegation credentials to a potential requester. Upon receiving a request a delegation service checks the validity of request again established delegation policies and issue delegation credentials based on a given set of delegation specifications. Depending on received delegation request and type of requested delegated credential the delegation service may use different mechanisms at back-end to obtain requested credentials.

- **Grid Delegation Protocol (GrDP)** is a delegation protocol to describe delegation as a standalone Web Services portType based on the WS-Trust specification. A set of ready-to-use library implementations of this portType were also developed to enable service implementers to support delegation without a need for understanding implementation details. This helps to factor out delegation functionality from the applications. Furthermore, GrDP instantiates delegation as a standalone service, frontending a (trusted) credential repository, from which the target service can extract the delegated credential in a controlled and secure manner.

- **Ontology-based software components** are a set of Java libraries developed for our on-demand delegation framework by using the Protégé-OWL API [26, 117]. These software components are used for both: populating and instantiating delegation ontology and generating, parsing and evaluating ontological queries dynamically.

- **Credential Mapping Service (CMS)**, is a Web Service that issues security tokens as defined by the WS-Trust specification. This service can be used to exchange a security token which is not in a format or syntax understandable by recipient domains. For example, if a user holds a Kerberos ticket asserting their identity, but the target service needs an X.509 certificate, the Kerberos ticket can be presented to CMS, which will issue the holder with an equivalent X.509 certificate asserting the same identity. The CMS implementation shares the same libraries with delegation service as both of these services are using the same interface.

- **PURSe** provides an end-user portal for credential management in grids and it is currently being used as an end-user portal for SweGrid\(^9\) and the Earth System Grids (ESG)\(^{10}\). We have contributed in development of PURSe by providing a set of software components that improve the security and functionality of this portal. Our contributed software components enables PURSe

---

\(^9\)[http://www.swegrid.se/]

\(^{10}\)[http://www.earthsystemgrid.org]
to obtain certificates through a secure communication with an external Certification Authority. They also make PURSs to interacting with a credential mapping service (CMS) for generating different types of security tokens [45, 10].

- **gLite-java-delegation** is a set of Java libraries that provide support for performing delegation within the context of gLite grid middleware as described in Section 3.3. The gLite-java-delegation component consists of three sub-components. The gLite-delegation-interface, defines a delegation interface based on the G-HTTPS request-response interaction protocol as described in Section 3.3. The gLite-delegation-java provides a collection of Java libraries to perform delegation. These APIs not only implement the gLite delegation interface, but also provides developer with a set of standalone Java libraries to perform delegation in a flexible and arbitrary way. Finally, the gLite-delegation-service implements a stand-alone delegation service, which can be used by clients (Delegatees) to obtain delegation privileges. More information can be achieved through the technical documentations of relevant APIs [6].

### 5.4 Related Work

In this section we describe related works in the area of grid delegation and other relevant areas including active security models, and credential mapping mechanisms.

There are significant prior works on delegation in grids and other distributed computing environments. In this section, we describe some of these works that we found most relevant to our own.

Delegation Issuing Service (DIS) [34] is a service that issues X.509 Attribute Certificates (AC) on behalf of an Attribute Authority (typically a manager), integrated into the PERMIS authorization infrastructure. DIS only issue Attribute Certificates and lacks a standard protocol for interactions with an Attribute Authority who requests to issue ACs. It still needs a human to decide what is the least set of privileges to delegate to another entity and there is no support for an automated logic that can determine what are the least privileges to delegate.

The Community Authorization Service (CAS) [84] is a third-party, trusted by resource owners and used by end-users to obtain rights to access resources. It issues credentials, which limit the rights of the holder to only those agreed on between the VO and the resource providers. CAS has been primarily developed to set PCI extensions to limit the delegated rights to the intersection of rights between VOs and resource owners. The Virtual Organization Membership Service (VOMS) [12] has also been developed to solve this problem. It grants authorization data to users at the VO level by providing support for group membership, roles and capabilities. Although CAS and VOMS allow users to obtain and delegate specific rights via tags and roles during a session, they do not address the need for dynamic, on-demand

---

delegation, considering that it is often difficult to determine the rights needed by a grid job in advance.

“Explicit Trust Delegation” (EDT) described in [49] introduces “trusted agents” to provide support of dynamic delegation in the UNICORE security model (We fully described delegation in UNICORE in Section 3.1). These trusted agents (e.g. portals) are the entities that are allowed to create and sign Sub-AJOs on behalf of end-users. Scalability, the lack of providing auditing and robust control over delegated rights are some other issues and concerns of this approach.

Currently the GSI approach, described in Section 3.2, uses identity credentials with gridmap-files, a kind of ACL (access control list), in its authorization system. However, emerging works like Akenti [105], PERMIS [35] and, PRIMA [74] have been aimed at providing support for policy-based authorization system in GSI. Akenti and PERMIS provide a distributed policy-based authorization system. Akenti designed for grid environments using attribute certificates and delegated authorization. PERMIS also uses X.509 attribute certificates to hold roles/attributes and currently is fully integrated into the Globus GT4 authorization framework. Privilege Management and Authorization system (PRIMA) embeds authorization credentials in GSI proxy certificate to enable transport of authorizations in GSI. PRIMA can also embed XACML [101] privilege statements in GSI proxy credentials to be compared with XACML resource policies in PRIMA access control engine.

Rein [63] is an open and extensible approach for representing policies and provides a unified way of decision making by reasoning over policies and delegation networks. It uses Ontologies for specifying and reasoning about access policies in heterogeneous policy domains with different policy languages. Ontologies are used for describing and modeling different information in this framework such as policies, requests and delegation of authority and trust. However, in regard to delegation, this framework can only be used for simple scenarios and mainly for asserting delegation chains and delegation constraints cryptographically. From this perspective, Rein is analogous to other policy frameworks developed to support delegation of authority and trust to address delegation and can not be used as a complete solution for addressing the issue of principle of least privileges in grids.

Rein basically uses high level Rei concepts [61, 64], which is a policy framework that we used for reasoning over delegation policies in our framework. It has strong potential to meet the requirements of active delegation framework described in Paper II. The Rei policy framework permits specifying, analyzing and reasoning over declarative policies defined as norms of behavior [62, 61]. Rei policies can restrict domain actions that an entity can/must perform on resources in the environment, allowing policies to be deployed as contextually constrained deontic concepts, i.e., permission, prohibition, obligation and dispensation. In the Rei policy specification\(^\text{12}\), context conditions can be specified by defining one or more constraints. A constraint, which may be simple or Boolean, i.e., the boolean combination of a pair of simple constraints, defines a set of actors or a set of actions that fulfill a certain

\(^{12}\text{http://www.cs.umbc.edu/~lkagal1/rei/}
property. Rei’s support for contexts makes it well-suited for our purposes.

In order to use Rei, as described in Paper II, we have extended SpeechAct delegation by incorporating a new property called *pre-condition*, which is defined as the conditions that need to be true before the delegation speech act can be performed. These pre-conditions are in fact constructing the “context” in which delegation can be activated and a delegation speech act can be triggered in response to a request. We need to recognize the distinction between the *pre-condition* property and the *condition* property defined originally in the Rei specification. The *pre-condition* determines when rights may be delegated, while the *condition* determines when rights may be used. Grid tasks can take longer than the validity interval of a context for which delegation is activated, which implies that constraints that must be satisfied before delegating rights should not necessarily remain true to make the delegated rights usable.

“Multiple Authorization” is a concept suggested in [116] for restricted delegation to prevent abusing of delegated tasks in a management system based on mobile agents. It proposes to share the responsibility for protected operations and to supervise actions of subordinates instead of transferring rights or even chains between agents before delegating the task. In this approach, a protected operation cannot be executed unless additional authorizations by other partners are provided. This approach binds authorization to particular and protected operations under special circumstances. This enables a fine-grained access control on a delegated task. This approach introduces two kinds of agents; work agents and authorization agents. Authorization agents are owned by the entities that their approvals are required for executing the task. When the work agent needs to execute a delegated task, it has to request for the authorizations that are needed to complete the task. Thus it sends a message to all agents and asks them to send their corresponding authorization agents. When all the authorization agents are collected, they will grant the execution of task by the work agent on behalf of user.

The “Workflow-based Authorization Service” (WAS) [66] proposes an authorization architecture for supporting restricted delegation and rights management. The WAS architecture uses a task workflow, created from the task source code, to obtain the sequence of required rights for executing the task. This can provide a useful way of determining the required rights in advance for deterministic jobs. However, in practice we still need on-demand delegation, because even if we can predict the job’s behavior, the environment is dynamic, and we need the flexibility to use different services on grids opportunistically.

Trust negotiation is an approach for establishing trust in open distributed systems [25, 130]. In trust negotiation, two strangers carry out a bilateral and iterative exchange of attribute credentials to gradually establish trust in one another. On-demand delegation mechanism sets out to solve somewhat the different problem of determining when to grant more privileges to a sub-job running on behalf of a user. This approach complements trust negotiation in that our work could be used in a system employing trust negotiation to determine when a user’s sensitive attribute certificates should be accessible to his sub-jobs. The Delegation Service
in on-demand delegation framework also bears some resemblance to the Traust Service [70], which acts as a stand-alone authorization service that uses trust negotiation to broker access tokens for resources in its security domain.

The notation of context and active security system are observed in many works. In some of them diverse delegation models are also supported. However, as far as we are aware, the delegation concept in those works has not been investigated from the perspective of our own work presented in this thesis. Most of the works in this context are based on the RBAC96 family [93], which supports role activation within sessions to provide an active model of authorization management and access control models, such as the TeaM-based Access Control (TMAC) model, which is introduced in [102] and extended in [51] to a family of context aware access control models. TMAC basically recognizes the importance of context information for just-in-time activation of permissions. Similar to the TMAC approach, the Task-Based Access Control (TBAC) model [104] is also used for management of authorizations that encapsulate a group of permissions, in a way that they are turned-on only in a just-in-time fashion and synchronized with the processing of authorizations in progressing tasks. The OASIS RBAC model [127] is also an extension to the role-based access control architecture. OASIS RBAC does not use role delegation but instead defines the notion of appointment. One could consider applying our work to these systems by employing an active security model to determine when a team or session needs to be created or a new member needs to be joined into an established context (team).

None of above approaches addresses all the requirements of delegation described earlier. For the works around active security systems, although they provide support for contexts, they lack the ability to determine the “need” for establishing contexts “dynamically”, which is an essential requirement for highly dynamic environments like grids. This is where we introduce the Context Manager as an architectural component. We also recognize a very special requirement of grid delegation when we distinguish the constraints applied at authorization time from the constraints applied at delegation time. The former implies that a delegation must be valid as long as the conditions are fulfilled, while the latter specifies that a delegation can be activated only when a set of pre-conditions are met. This is where we extend the Rei policy language to incorporate “pre-conditions” in the delegation SpeechAct as described earlier. What distinguishes our work from others is the adaptation of the active delegation model to grids and providing a supporting framework which can be integrated into current grid systems and leveraged by existing security mechanisms. Our framework further provides more support for observing and auditing the delegation process adapted to the dynamic requirements of grid applications.

In the context of integration of workflows when performing delegation, the work presented in [119, 118] provides an authorization model to handle delegation in an RBAC workflow environment. Their model extends the RBAC model to support a user-to-user delegation, user to a group of users delegation and revocations. Their workflow delegation model addresses dynamic constraint handling by defining spe-
5.4. RELATED WORK

cific delegations. Multiple delegation and partial delegation through the chain of
delegation is also supported by this model.

The work presented in [11] incorporates delegation in a workflow management
system. There is a delegation module, which is invoked whenever a delegation
decision for task assignment must be made. There is a delegation table that only
displays static constraints and relationships of delegations; then when a workflow
assigns a task, this assignment is screened by the delegation module to determine
whether delegation is required. The delegation module in turn inquires delegation
tables and reply “yes” or “no” to the assignment request. It will notify the workflow
engine to generate a new task assignment for a new delegatee. This model uses a
static delegation table along with a static time constraints model; the authorization
and the workflow are not accommodated and therefore if a task is completed sooner
than anticipated, the delegated authorization being valid for longer than required.

All these approaches perform delegation in a full-ahead-plan tied to a particular
user or group. They are mainly intended to business workflows which are more
static, predictable and controlled environments as described in Paper III.

In the context of security credential mapping our work was originally inspired
from the KX.509 protocol [41], which provides a mechanism for obtaining X.509
identity certificates based on a Kerberos domain login identity. Two main compo-
nents in this protocol are i) the KCA (Kerberized Certification Authority) [67], a
Kerberized service, running inside a Kerberos domain, and provides the function-
ality of an X.509 certification authority, and ii) the KX.509 that is a standalone
client program to acquire a short-lived X.509 certificate from the KCA for an au-
thenticated Kerberos user. As for the differences, CMS uses a standard interface
for exchanging security tokens where the KX.509 does not. Our approach is more
general and provides support for more use cases in the sense that it can let both
the user or any delegated entity request for exchanging a security token (not only
a Web browser as approached by KX.509).

MyProxy [81] is a software for managing PKI security credentials. It combines
an on-line credential repository with an online certification authority to allow users
to securely obtain credentials when and where needed. MyProxy supports storage
of multiple credentials (X.509 proxy certificates) per user and retrieval as needed
using username and password.

SACRED (Securely Available Credentials) [54] is an IETF framework and pro-
tocol for credential portability to allow secure and authenticated access to security
credentials. This project provides SACRED client and server implementations.

CredEx [114], is a Web Service which provides a secure storage of credentials
and enables exchanging of different credential types using the WS-Trust token ex-
change protocol. It can be seen as a credential mapping mechanism for exchanging
username-password token to X.509 certificates.

The works described above basically develop an on-line credential repository
to ease and secure the process of credential management. Although such on-line
credential repositories are also required for many use-cases, however, we should
emphasize that the work presented in this thesis was intentionally designed for just
CHAPTER 5. RESULTS

not storing credentials on a remote and on-line repository; it is rather designed to perform an on-the-fly credential mapping to make a non-vulnerable, lightweight, easy operated and quick to configure service. Furthermore, these solutions do not provide a full mesh credential exchange as provided by the CMS solution presented in this thesis.

5.5 Conclusion

In this thesis we demonstrated the importance of delegation in a wide range of grid applications and use-cases. We highlighted the shortcomings of existing mechanisms in providing a dynamic, restricted, standard and interoperable delegation system for grid applications. We discussed that current approaches are unable to provide secure and restricted delegation for dynamic grid applications. We demonstrated that current delegation mechanisms are mostly coupled to the underlying security mechanisms and are not interoperable between different grid security systems; furthermore, we demonstrated that there is a lack of standard interfaces and protocols for performing delegation in heterogeneous grid environments.

We solved these problems with a novel approach providing restricted delegation for dynamic execution of grid applications. We analyzed and developed the building blocks of a delegation framework, which allows on-demand determination, requesting and provisioning of delegated credentials. This framework has the benefits of observing, screening and auditing of delegated rights during the execution of tasks. We described how this framework, which leverages a call-back mechanism and a flexible and interoperable delegation protocol (GrDP), restricts the scope of delegated rights solely to those required for completing an intended task. In order to address some security concerns and preventing violation of delegation policies, we extended our framework to be capable of establishing “contexts” in which the need for additional delegated rights could be verified.

We discussed that on-demand delegation, although having the benefits of real-time control and auditing at the delegation service, might penalize delegation service performance for obtaining rights. In that regard, we developed a strategy to optimize on-demand delegation by delegating more rights to jobs either at the time of launching delegation or during each callback; thus, a delegatee does not have to callback to a delegation service so frequently. Determining these set of rights was a challenging issue and we addressed this challenge by integrating on-demand delegation framework with workflows.

We introduced the concept of Risk of Delegation (RoD) parameter in on-demand delegation framework. We further implemented appropriated algorithms for three different models of bottom-up delegation: Per-step delegation, Multi-step delegation and One-Step delegation. These models were designed to perform delegation in different levels of efficiency and restriction based on the current threat landscape indicated by the RoD parameter.

Finally, in order to address the cross-organizational authentication and iden-
5.6. Future Work

We proposed using the Rei policy framework for on-demand delegation frameworks. However, we believe that any other policy language incorporating the concept of contexts and providing support for delegation can also be used to describe active delegation policies. Therefore, future work could be investigating how other policy languages may be used in such framework. One alternative is XACML [101], a standard and general-purpose policy system designed to support the needs of modern authorization systems. The current policy schema of XACML version 2.0 does not support delegation; however, version 3.0 of this policy language will incorporate delegation concepts in the XACML policy schema, making XACML a strong and standard alternative for deployment in on-demand delegation frameworks.

We have not yet integrated our framework with a real grid workflow system. We only gave a formal definition and evaluation of that idea, and therefore another future task is to provide a concrete integration of the on-demand delegation with a real grid workflow system. Adopting the concept of bottom-up delegation to other grid middlewares and frameworks is another task and topic of future study.

The level of granularity of resource and system description can also affect the complexity of restricted delegation. Fine-grained system descriptions and access rights may result in unnecessarily complicated processes for restricting delegation and high overhead. This may sacrifice the usability of the whole system. Therefore, an investigation should be performed as to what is the best level of granularity to describe the system and grid resources without losing security and introducing unnecessary complexity.
Bibliography


Appendix I: Delegation ontology in OWL/XML notation

<?xml version="1.0"?>

<!DOCTYPE owl2xml:Ontology [ 
  <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
  <!ENTITY dc "http://purl.org/dc/elements/1.1/" >
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
  <!ENTITY owl2xml "http://www.w3.org/2006/12/owl2-xml#" >
  <!ENTITY daml "http://www.daml.org/2001/03/daml+oil#" >
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
  <!ENTITY owl2xml "http://www.w3.org/2006/12/owl2-xml#" >
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-syntax-ns#" >
  <!ENTITY http://www.w3.org/2001/XMLSchema# >
  <!ENTITY rdfs "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
  <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
  <!ENTITY owl2xml "http://www.w3.org/2006/12/owl2-xml#" >
  <!ENTITY owl "http://www.nada.kth.se/Delegation.owl#" >
  <!ENTITY p1 "http://www.daml.org/2003/01/periodictable/PeriodicTable#" >
]>

<owl2xml:Ontology xmlns="http://www.nada.kth.se/Delegation.owl#"
  xml:base="http://www.w3.org/2006/12/owl2-xml#" 
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:owl="http://www.nada.kth.se/Delegation.owl#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl2xml="http://www.w3.org/2006/12/owl2-xml#"
  xmlns:p1="http://www.daml.org/2003/01/periodictable/PeriodicTable#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:rdfs="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:assert="http://www.owl-ontologies.com/assert.owl#"
  xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
  xmlns:Delegation="http://www.nada.kth.se/Delegation.owl#"
owl2xml:URI="http://www.nada.kth.se/Delegation.owl">
  <owl2xml:Import
      >http://www.owl-ontologies.com/assert.owl</owl2xml:Import>
  <owl2xml:Import
      >http://www.owl-ontologies.com/assert.owl</owl2xml:Import>
  <owl2xml:SubClassOf>
    <owl2xml:Class owl2xml:URI="&;Capability"/>
    <owl2xml:Class owl2xml:URI="&owl;Thing"/>
  </owl2xml:SubClassOf>
  <owl2xml:SubClassOf>
    <owl2xml:Class owl2xml:URI="&;Capability"/>
    <owl2xml:ObjectSomeValuesFrom>
      <owl2xml:ObjectProperty owl2xml:URI="&;hasObject"/>
      <owl2xml:Class owl2xml:URI="&;Object"/>
    </owl2xml:ObjectSomeValuesFrom>
  </owl2xml:SubClassOf>
  <owl2xml:SubClassOf>
    <owl2xml:Class owl2xml:URI="&;Capability"/>
    <owl2xml:ObjectSomeValuesFrom>
      <owl2xml:ObjectProperty owl2xml:URI="&;hasVerb"/>
      <owl2xml:Class owl2xml:URI="&;Verb"/>
    </owl2xml:ObjectSomeValuesFrom>
  </owl2xml:SubClassOf>
  <owl2xml:SubClassOf>
    <owl2xml:Class owl2xml:URI="&;DN"/>
    <owl2xml:Class owl2xml:URI="&;Identifier"/>
  </owl2xml:SubClassOf>
  <owl2xml:SubClassOf>
    <owl2xml:Class owl2xml:URI="&;Name"/>
    <owl2xml:Class owl2xml:URI="&;Identifier"/>
  </owl2xml:SubClassOf>
  <owl2xml:SubClassOf>
    <owl2xml:Class owl2xml:URI="&;NotAfter"/>
    <owl2xml:Class owl2xml:URI="&;Constraints"/>
  </owl2xml:SubClassOf>
  <owl2xml:SubClassOf>
    <owl2xml:Class owl2xml:URI="&;NotBefore"/>
    <owl2xml:Class owl2xml:URI="&;Constraints"/>
  </owl2xml:SubClassOf>
  <owl2xml:SubClassOf>
    <owl2xml:Class owl2xml:URI="&;URI"/>
    <owl2xml:Class owl2xml:URI="&;Identifier"/>
  </owl2xml:SubClassOf>
</owl2xml:SubClassOf>
<owl2xml:Class owl2xml:URI="&;Capability"/>
</owl2xml:ObjectPropertyRange>
<owl2xml:ObjectPropertyDomain>
    <owl2xml:ObjectProperty owl2xml:URI="&;hasDependency"/>
    <owl2xml:Class owl2xml:URI="&;Capability"/>
</owl2xml:ObjectPropertyDomain>
<owl2xml:ObjectPropertyRange>
    <owl2xml:ObjectProperty owl2xml:URI="&;hasDependency"/>
    <owl2xml:Class owl2xml:URI="&;Capability"/>
</owl2xml:ObjectPropertyRange>
<owl2xml:ObjectPropertyDomain>
    <owl2xml:ObjectProperty owl2xml:URI="&;hasObject"/>
    <owl2xml:Class owl2xml:URI="&;Capability"/>
</owl2xml:ObjectPropertyDomain>
<owl2xml:ObjectPropertyRange>
    <owl2xml:ObjectProperty owl2xml:URI="&;hasObject"/>
    <owl2xml:Class owl2xml:URI="&;Object"/>
</owl2xml:ObjectPropertyRange>
<owl2xml:SubObjectPropertyOf>
    <owl2xml:ObjectProperty owl2xml:URI="&;hasValidityInterval"/>
    <owl2xml:ObjectProperty owl2xml:URI="&;isRestrictedBy"/>
</owl2xml:SubObjectPropertyOf>
<owl2xml:InverseObjectProperties>
    <owl2xml:ObjectProperty owl2xml:URI="&;hasVerb"/>
    <owl2xml:ObjectProperty owl2xml:URI="&;isVerbOf"/>
</owl2xml:InverseObjectProperties>
<owl2xml:ObjectPropertyDomain>
    <owl2xml:ObjectProperty owl2xml:URI="&;hasVerb"/>
    <owl2xml:Class owl2xml:URI="&;Capability"/>
</owl2xml:ObjectPropertyDomain>
<owl2xml:ObjectPropertyRange>
    <owl2xml:ObjectProperty owl2xml:URI="&;hasVerb"/>
    <owl2xml:Class owl2xml:URI="&;Verb"/>
</owl2xml:ObjectPropertyRange>
<owl2xml:FunctionalObjectProperty>
    <owl2xml:ObjectProperty owl2xml:URI="&;identifies"/>
</owl2xml:FunctionalObjectProperty>
<owl2xml:ObjectPropertyDomain>
    <owl2xml:ObjectProperty owl2xml:URI="&;identifies"/>
    <owl2xml:Class owl2xml:URI="&;Identifier"/>
</owl2xml:ObjectPropertyDomain>
<owl2xml:ObjectPropertyRange>
    <owl2xml:ObjectProperty owl2xml:URI="&;identifies"/>
</owl2xml:ObjectPropertyRange>
<owl2xml:ObjectUnionOf>
<owl2xml:Class owl2xml:URI="&Credential"/>
<owl2xml:Class owl2xml:URI="&Delegation"/>
<owl2xml:Class owl2xml:URI="&Delegator"/>
<owl2xml:Class owl2xml:URI="&Object"/>
<owl2xml:Class owl2xml:URI="&Subject"/>
</owl2xml:ObjectUnionOf>
</owl2xml:ObjectPropertyRange>
<owl2xml:ObjectPropertyDomain>
  <owl2xml:ObjectProperty owl2xml:URI="&isApplicableTo"/>
  <owl2xml:ObjectProperty owl2xml:URI="&Delegation"/>
</owl2xml:ObjectPropertyDomain>
<owl2xml:ObjectPropertyRange>
  <owl2xml:ObjectProperty owl2xml:URI="&isApplicableTo"/>
  <owl2xml:ObjectProperty owl2xml:URI="&Capability"/>
</owl2xml:ObjectPropertyRange>
<owl2xml:InverseObjectProperties>
  <owl2xml:ObjectProperty owl2xml:URI="&isAuthorizedBy"/>
  <owl2xml:ObjectProperty owl2xml:URI="&authorizes"/>
</owl2xml:InverseObjectProperties>
<owl2xml:ObjectPropertyDomain>
  <owl2xml:ObjectProperty owl2xml:URI="&isAuthorizedBy"/>
  <owl2xml:ObjectUnionOf>
    <owl2xml:Class owl2xml:URI="&Capability"/>
    <owl2xml:Class owl2xml:URI="&Delegation"/>
  </owl2xml:ObjectUnionOf>
</owl2xml:ObjectPropertyDomain>
<owl2xml:ObjectPropertyRange>
  <owl2xml:ObjectProperty owl2xml:URI="&isAuthorizedBy"/>
  <owl2xml:ObjectUnionOf>
    <owl2xml:Class owl2xml:URI="&Capability"/>
    <owl2xml:Class owl2xml:URI="&Delegation"/>
  </owl2xml:ObjectUnionOf>
</owl2xml:ObjectPropertyDomain>
<owl2xml:ObjectPropertyRange>
  <owl2xml:ObjectProperty owl2xml:URI="&isAuthorizedBy"/>
  <owl2xml:ObjectUnionOf>
    <owl2xml:Class owl2xml:URI="&Credential"/>
  </owl2xml:ObjectUnionOf>
</owl2xml:ObjectPropertyDomain>
<owl2xml:ObjectPropertyDomain>
  <owl2xml:ObjectProperty owl2xml:URI="&isDelegatableTo"/>
  <owl2xml:ObjectProperty owl2xml:URI="&Delegation"/>
</owl2xml:ObjectPropertyDomain>
<owl2xml:ObjectPropertyRange>
  <owl2xml:ObjectProperty owl2xml:URI="&isDelegatableTo"/>
  <owl2xml:ObjectProperty owl2xml:URI="&Subject"/>
</owl2xml:ObjectPropertyRange>
<owl2xml:ObjectPropertyDomain>
  <owl2xml:ObjectProperty owl2xml:URI="&isDelegatedBy"/>
  <owl2xml:ObjectProperty owl2xml:URI="&Delegation"/>
</owl2xml:ObjectPropertyDomain>
<owl2xml:ObjectPropertyDomain>
  <owl2xml:ObjectProperty owl2xml:URI="&isDelegatedBy"/>
<owl2xml:DataProperty owl2xml:URI="&;isValidNotAfter"/>
<owl2xml:Class owl2xml:URI="&;Delegation"/>
</owl2xml:DataPropertyDomain>
<owl2xml:DataPropertyRange>
  <owl2xml:DataProperty owl2xml:URI="&;isValidNotAfter"/>
  <owl2xml:Datatype owl2xml:URI="&xsd ; string"/>
</owl2xml:DataPropertyRange>
<owl2xml:DataPropertyDomain>
  <owl2xml:DataProperty owl2xml:URI="&;isValidNotBefore"/>
  <owl2xml:Datatype owl2xml:URI="&xsd ; string"/>
</owl2xml:DataPropertyDomain>
<owl2xml:DataPropertyRange>
  <owl2xml:DataProperty owl2xml:URI="&;isValidNotBefore"/>
  <owl2xml:Datatype owl2xml:URI="&xsd ; string"/>
</owl2xml:DataPropertyRange>
<owl2xml:DataPropertyAssertion>
  <owl2xml:DataProperty owl2xml:URI="&assert ;notEmpty"/>
  <owl2xml:Individual owl2xml:URI="&;Delegator"/>
  <owl2xml:Constant owl2xml:datatypeURI="&xsd ; string">
    SELECT ?Delegator ?Capability WHERE {
      ?Delegator:hasA ?Capability
    }
  </owl2xml:Constant>
</owl2xml:DataPropertyAssertion>
</owl2xml:Ontology>
Part II

Papers
Paper I: Toward An On-demand Restricted Delegation Mechanism for Grids

Toward an On-Demand Restricted Delegation Mechanism for Grids

Mehran Ahsant #1, Jim Basney ∗2, Olle Mulmo #3, Adam J. Lee %4, Lennart Johnsson #5

#1 Center for Parallel Computers, Royal Institute of Technology
Valhallavgen 79, 10044 Stockholm, Sweden
mehrana@pdc.kth.se
#2 National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign
201 N. Goodwin Ave., Urbana, IL 61801 USA
jbasney@ncsa.uiuc.edu
#3 Department of Computer Science, University of Illinois at Urbana-Champaign
1205 W. Clark St., Urbana, IL 61801 USA
mulmo@pdc.kth.se
%4 Department of Computer Science, University of Illinois at Urbana-Champaign
201 N. Goodwin Ave., Urbana, IL 61801 USA
adamlee@cs.uiuc.edu

Abstract—Grids are intended to enable cross-organizational interactions which makes Grid security a challenging and non-trivial issue. In Grids, delegation is a key facility that can be used to authenticate and authorize requests on behalf of disconnected users. In current Grid systems there is a trade-off between flexibility and security in the context of delegation. Applications must choose between limited or full delegation: on one hand, delegating a restricted set of rights reduces exposure to attack but also limits the flexibility/dynamism of the application; on the other hand, delegating all rights provides maximum flexibility but increases exposure. In this paper, we propose an on-demand restricted delegation mechanism, aimed at addressing the shortcomings of current delegation mechanisms by providing restricted delegation in a flexible fashion as needed for Grid applications. This mechanism provides an ontology-based solution for tackling one of the most challenging issues in security systems, which is the principle of least privileges. It utilizes a callback mechanism, which allows on-demand provisioning of delegated credentials in addition to observing, screening, and auditing delegated rights at runtime. This mechanism provides support for generating delegation credentials with a very limited and well-defined range of capabilities or policies, where a delegator is able to grant a delegatee a set of restricted and limited rights, implicitly or explicitly.

I. INTRODUCTION

Grid computing, since its emergence, has been widely regarded as a revolution in information technology. Grids provide mechanisms for the creation and management of large numbers of dynamic virtual organizations (VOs), spanning multiple heterogeneous organizations with different underlying security mechanisms [1][2].

In a Grid, users or applications may need access to resources that belong to different organizations. The jobs submitted by users of these systems may take long periods of time to execute, and resource providers usually require some form of authentication of users and authorization of requests, i.e., that a user is allowed to use the requested resources as described in the request. Requests may be generated dynamically during the execution of an application and need user approval before being submitted to a resource broker or resource provider. However, the user may not be directly accessible either because of malfunction simply having disengaged after submission of a request for execution of an application. A common solution to the problem of disconnection is to delegate the authorization to an agent (program) that acts on the user’s or application’s behalf and that is less likely to be disengaged at any time [1].

Delegation is a key facility in Grids that makes possible an effective use of a wide range of dynamic Grid applications. However, the least privilege principle, stating that we should grant only those rights required to perform a required action to minimize vulnerability, is becoming one of the most important security aspects in regards to delegation. There are potential security risks associated with performing delegation in a way that delegated rights are not limited only to the task intended to be performed within a limited lifetime and under restricted conditions. Therefore, a fine grained policy for restricted delegation is highly desirable, yet care must be taken to not introduce unmanageable complexity.

However, performing restricted delegation in a static manner cannot meet all the requirements of dynamic execution of Grid applications, because the required access rights for completing a task cannot easily be anticipated in advance. Delegating fewer rights than required for completing the task may cause task execution to fail while delegating more rights than needed may threaten abuse by malicious parties. Therefore, utilizing a mechanism that allows determining and acquiring only required rights and credentials for completing a task, when they are needed, would be a more reasonable and robust approach for restricted delegation.

We will discuss later in this paper how current delegation approaches in Grids trade off between security and flexibility. We therefore conclude that a desirable delegation mechanism for Grid applications needs to balance the security and the
flexibility of delegation while minimizing dependencies on underlying security mechanisms used by diverse Grid systems.

In this paper, we propose a novel solution for the least privilege delegation in dynamic, distributed environments, which no existing solution adequately addresses. We describe an on-demand delegation mechanism that tackles the most challenging issues of dynamic restricted delegation. This mechanism exploits ontologies and the potential power of ontological queries for determining required access rights at run-time and utilizes a callback mechanism for acquiring corresponding credentials required for completing the task at on-demand. It should also be noted that this paper deals only with the architectural issues of this mechanism and does not describe in details the implementation.

Our paper proceeds as follows. In section II, we give an overview of our on-demand delegation framework. In section III, we describe related work aimed at addressing restricted delegation. In section IV we describe how ontologies can be exploited in our proposed framework. In section V, we provide a Grid job simulation scenario as a motivating example of the potential power of this approach and finally sections VI and VII describe the security considerations and current implementation status. Our conclusion and proposed future work are both described in section VIII.

II. ON-DEMAND DELEGATION

On-demand delegation is a novel approach in which a delegatee obtains additional rights (in terms of additional delegated credentials) as required and requested for acting on behalf of a delegator. On-demand delegation allows delegators to delegate privileges when the other party has proved the necessary need for those privileges. This implies delegating rights to delegatees iteratively as needed until the task has completed.

![Diagram of delegation process](image)

**Fig. 1. An example of on-demand delegation**

Fig. 1 illustrates an example of performing on-demand delegation. Through this approach, a delegator initiates delegation by providing a least set of rights to a delegatee (job). This least set of rights only allows the delegate to prove that it needs to act on behalf of its delegator. The Delegatee has no rights to access resources and therefore needs to query the delegation system model to determine what additional credentials are required to complete the task. By this query, the required privileges for executing the task are determined and disclosed to the delegatee during the job’s execution. The Delegatee contacts one or more Delegation Services to determine what credentials are needed and to then obtain them. In this simplest case, there is a Delegation Service associated with the Delegator and one associated with the Resource. The Delegation Service checks requests against policies established for each specific delegatee that specify the circumstances for issuance of additional credentials.

On-demand delegation allows a delegatee to leverage very simple ontological queries for determining the required access rights dynamically at run-time and to utilize a call-back mechanism to request required credentials from Delegation Services. Delegation policies are established to govern if additional credentials can be granted to a delegatee upon request and enable the delegator to keep full control over delegated rights. Ontologies are the powerful means for establishing these policies dynamically at the time of performing each delegation.

In fact, what an on-demand delegation mechanism provides is an efficient approach for performing restricted delegation in a dynamic fashion, because in practice we expect a small number of credentials to be required which can be determined by querying appropriate delegation ontologies. Ontological queries provide a very simple and efficient way of acquiring required access rights according to system descriptions.

We address the important concern of scalability by distributing Delegation Services across the Grid associated with different resources. Delegation Services need only know about local policies, and delegatees should need to query only a few Delegation Services in practice. This also provides manageability by providing local policy control points (the Delegation Service), and the semantic web community has shown [3] that ontologies can be used effectively for specifying policies in practice. By using call-backs, our on-demand framework never leaves a job stranded without credentials, but policy dictates the least privilege credentials that are delegated. When a job needs credentials that policy won’t allow, we can put the job “on hold”, notify the user, and wait for user intervention. If the user approves, the job can then proceed, thereby providing a very effective approach compare to other delegation mechanisms.

The approach presented in this paper is in some ways similar to the more general notion of trust negotiation [4] [5]. In trust negotiation, two strangers carry out a bilateral and iterative exchange of attribute credentials to gradually establish trust in one another. We set out to solve the somewhat different problem of determining when to grant more privileges to a subjob running on behalf of a user. This approach complements trust negotiation in that our work could be used in a system employing trust negotiation to determine when a user’s sensitive attribute certificates should be accessible to his subjobs. The Delegation Service in our framework also bears some resemblance to the Traust service [6], which acts as...
a stand-alone authorization service that uses trust negotiation to broker access tokens for resources in its security domain. Our Delegation Services, however, grant privileges only to a certain user’s subjobs or provide policy-level information for resources in a particular security domain.

It should also be noted that for a system with a high level of granularity of access rights, making multiple calls to a Delegation Service could hurt performance. Performance evaluation and optimization is an important area for our future work, discussed in brief in section VIII.

III. RELATED WORK

There is significant prior work on restricted delegation for Grids and other distributed computing environments. In this section, we describe the prior work that we find most relevant to our own.

UNICORE1, which is one of the widely used Grid infrastructures, addresses delegation and multi-site job execution by creating Sub-AJOs from a parent Abstract Job Object (AJO) [7]. In this approach, all components of an AJO are signed with the end-users’ certificate at creation time, granting a limited set of rights to the specific job. This implies a secure but static delegation mechanism. “Explicit Trust Delegation” proposed in [7] is aimed to address this shortcoming by introducing trusted agents that are allowed to create and sign AJOs on behalf of end-users. By this approach end-users need to trust other agents for endorsing Sub-AJOs on their behalf and if job runs on a site which its server is not trusted, execution fails. More ever end-users loose their control on Sub-AJOs during the execution of task. Scalability also is an issue in this approach. In order to ensure that an endorser is authorized to endorse a Sub-AJO on behalf of end-user many information should be coded explicitly in authorization database at each site.

The Globus Security Infrastructure (GSI) implements delegation by means of “proxy certificates”, which can provide full impersonation of end-users by granting all rights to a subordinate [8]. This provides a dynamic delegation mechanism, as there is no need to know the details of the execution in advance. However, this exposes all of the user’s rights to possible compromise. Issuing short-lived proxy certificates is one solution for limiting the danger caused by unauthorized acquisition or usage of a proxy identity. Another approach is to use the PCI extension to carry a policy statement that limits the delegated rights [8].

The Community Authorization Service (CAS) [9] is a third-party, trusted by resource owners and used by end-users to obtain rights to access resources. It issues credentials, which limit the rights of the holder to only those agreed on between the VO and the resource providers. CAS has been primarily developed to set PCI extensions to limit the delegated rights to the intersection of rights between VOs and resource owners. The Virtual Organization Membership Service (VOMS) [10] has also been developed to solve this problem. It grants authorization data to users at the VO level by providing support for group membership, roles and capabilities. Although CAS and VOMS allow users to obtain and delegate specific rights via tags and roles during a session, they do not address the need for dynamic, on-demand delegation, considering that it is often difficult to determine the rights needed by a Grid job in advance.

The “Workflow-based Authorization Service” (WAS) [11] proposes an authorization architecture for supporting restricted delegation and rights management. The WAS architecture uses a task workflow, created from the task source code, to obtain the sequence of required rights for executing the task. This can provide a useful way of determining the required rights in advance for deterministic jobs. However, in practice we still need on-demand delegation, because even if we can predict the job’s behavior, the environment is dynamic, and we need the flexibility to use different services on the Grid opportunistically.

“Multiple Authorizations” is a concept suggested in [12] for restricted delegation in a management system based on mobile agents. It proposes to share the responsibility for protected operations and to supervise actions of subordinates instead of transferring rights before delegating the task. In this approach, a protected operation cannot be executed unless the additional authorizations by other partners are provided. When a “work agent” needs to execute a delegated task, it has to collect approvals from all the corresponding “authorization agents” for the task. This approach utilizes a call back mechanism to ask for required grants on-demand, but the collection of required grants that authorizes task execution needs to be determined in advance.

Rein [13] is an open and extensible approach for representing policies and provides a unified way of decision making by reasoning over policies and delegation networks. It uses ontologies for specifying and reasoning about access policies in heterogeneous policy domains with different policy languages. Ontologies are used for describing and modeling different information in this framework such as policies, requests and delegation of authority and trust. However, in regard to delegation, this framework can mainly be used for cryptographically asserting delegations for policy decision making. From this perspective, Rein is analogous to other policy frameworks developed to support delegation of authority and trust to address delegation and can not be used as a complete solution for addressing the issue of delegation of least privileges in Grids.

None of above approaches address all the requirements of restricted delegation described earlier. In general, there is a compromising situation for addressing flexibility and security in the context of delegation. The most challenging issue of restricted delegation, namely dynamically anticipating access rights required for completing a task, is still unresolved. Some of these approaches have tried to address this challenge partially, though not through a dynamic and generic solution. All these approaches are strongly dependent on a specific

---

1http://www.unicore.org
and particular underlying authorization mechanism or Grid infrastructure. There is little support for observing and auditing of the delegation process that could be adapted to the dynamic requirements of Grid applications.

IV. DELEGATION ONTOLOGY

In computer science, an ontology is a conceptual schema that describes and classifying entities, the relationships between entities, and rules within a certain domain by means of a hierarchical data structure. It implies a more specialized schema for making the data useful for making real-world decisions. In this sense, Tom Gruber and R. Studer describe ontology as “an explicit and formal specification of a conceptualization”. The Semantic Web\(^2\) is a direct extension of the current Web to the explicit representation of knowledge by giving meaning and semantics, in a manner understandable by machines, to the content of documents on the Web. In Grids there are also many possible applications of knowledge-based problem-solving functionality which potentially can exploit ontologies. Therefore, the Semantic Grid\(^3\) is also emerging to add Semantic Web capabilities to the Grid computing applications. We propose that on-demand delegation is a paradigm in which ontologies can support the automatic process of determining, requesting and delegating credentials.

In on-demand delegation, ontologies can be populated for sharing a common understanding of a delegation concept among different Grid systems with different underlying security mechanisms. The delegation ontology can provide a formal description of the delegation concept to be instantiated specifically for each administrative domain. This provides strong support for reusing and analyzing the domain knowledge required to meet the requirements of dynamic restricted delegation. In a general sense, on-demand delegation exploits ontologies for:

- Describing systems and resources which eventually result in determining and providing required credentials for access to resources and;
- Establishing delegation policies to automate the process of decision making.

This implies that the delegation ontology can be populated to describe the service provider’s requirements for resource access as well as the Delegator’s policies for performing delegation. Later in this paper we show how ontologies can be used in a real Grid usage scenario to describe the delegation mechanism in a particular system for enabling fine-grained access to protect resources in a highly descriptive fashion.

Ontologies also have a strong potential for making more efficient, adaptive, and intelligent queries on any system description. With ontologies in place, one can start adding reasoning capabilities for automatic disclosure of privileges according to delegation policies. Furthermore, ontologies make defining and managing delegation polices easier\(^14\)|\(^15\). Ontology can be used for detecting conflicts and inconsistencies in the system description, which increase the risk of unauthorized access to resources. Even more, fulfillment of delegation policies could be assessed by analyzing resource usage and task definition information described using ontologies. Ontologies can also be used to determine the least privileges required to fulfill a request for delegation.

The delegation ontology depicted in Fig. 2, describes that each “Delegation” enables a “Delegator” to endow a “Capability” to a “Subject” under restricted conditions. “Credential” is also the means by which “Delegation” is authorized. It also depicts that each “Capability” contains one or more “Verbs” which can be accomplished on one or more “Objects”. Each “Capability” may have some dependencies on other capabilities, which implies a hierarchical delegation taxonomy in system description and consequently the need for further required delegation credentials.

Fig. 2 is a simple illustration of a delegation ontology populated for on-demand delegation. This picture only illustrates the most important and relevant classes defined for delegation ontology without depicting the arrangement of these classes in a taxonomic hierarchy. It also shows defined properties and allowed values for these properties. Hidden from this picture are the values for these properties which have been filled in for instances and a knowledge base which is also created by defining instances of these classes. On the service provider side, instances of the delegation ontology can also be used to describe the system in terms of “Objects”, “Verbs” and the “Capability” which makes an appropriate relation between the objects and verbs. It can also be used to specify individual instances of “Delegator” who can delegate access rights to the instances of class “Subject”. Individual instances of class “Credential” can also be used to describe how the authority of the owner for accessing resources can be approved. On the Delegator side, individual instances of delegation can be used for establishing delegation policies and further specifying how credentials should be issued and appropriate constraints be applied on them.

\(^2\)http://www.w3.org/2001/sw/
\(^3\)http://www.semanticgrid.org/
V. A Grid usage scenario

In this section, we demonstrate the potential benefits of on-demand delegation by thoroughly discussing the example depicted in Fig. 3. This picture describes a Grid scenario for job submission that utilizes on-demand delegation to perform a simulation job on the Grid. We describe in detail how on-demand delegation can be used to provide the job acting on behalf of Alice with the required delegation credentials for completing the task during its execution lifecycle.

Alice needs to submit an earthquake simulation job to the PDCGrid cluster. A Delegation Service is running on Alice’s side to mediate between Alice and Alice’s job for providing additional delegation credentials. The delegation service is aware of the domain delegation ontology that represents relationships between resources, users and delegates (jobs). This simulation job may need to get its input from a database on the NCSAGrid cluster, then perform a complex computation task on the SunetGrid cluster, and finally store the results in a database located in the PDCGrid cluster.

Alice needs to generate an Independent Proxy Certificate (IPC), which is required for creating an independent identity for the submitted job with which to associate Delegation Service policies. These policies specify how further privileges could be linked later to this credential for enabling its owner to act on her behalf for completing a specific task. For example, such policies may specify where (on which site/cluster) this submitted job could be executed or for what operations on which services this job can act on behalf of Alice.

Alice creates an IPC and submits the job. A mutual authentication happens between Alice and the cluster’s job manager. Once authentication is complete, the job manager will verify that Alice is authorized to submit the job. Alice also registers the IPC with the Delegation Service as a valid job’s identity with permission to make use of her privileges to access resources.

Now the job has started on PDCGrid job scheduler. During its execution the job needs to locate some input data for the simulation process. First it needs to access a replica location service (RLS). The job sends a query to the Delegation Service running on NCSAGrid cluster to determine what permissions are required to invoke the RLS on behalf of Alice. This query is depicted in Fig. 4 and the result is depicted in Table I. This states that access to the RLS on behalf of Alice is authorized through a proxy certificate.

The Delegation Service receives the request. It verifies the request signature, parses the request and compares it against Alice’s issuance policies, which she specified when she submitted the job. Ultimately, if the policy allows issuance of the requested credentials, the Delegation Service generates the credential and sends it to the job.

Now Alice’s job can authenticate to the RLS and use the service to locate its input data. The RLS determines that input data can be obtained from the ShakeTable on the NCSAGrid cluster. However, to obtain the input data it may need to determine the required credentials and request them from the Delegation Service again.

The job sends a query to the Delegation Service on the NCSAGrid cluster to determine what credentials are needed.
to authorize the capability “Read_DB” on behalf of delegator Alice. This query is depicted in Fig.5. The Delegation Service queries the ontology and sends the result back as it is depicted in Table II.

WHERE {
?delegator:isIdentifiedBy ?iden.
?delegator:hasA ?capability.
?capability:hasDependency ?dependency.
?dependency:isAuthorizedBy ?credential.
?dependency:hasObject ?object.
?object:isIdentifiedBy ?objiden.
?objiden:EPR ?EPR.
FILTER regex(str(?capability), "Read_DB")
FILTER regex(str(?iden), "Alice_DN")
FILTER regex(str(?object), "DB")
FILTER regex(str(?EPR), "gsiftp://ncsa.teragrid.org/shake/1352")
}

Fig. 5. Query on READ_DB

The results show the required access rights and associated credentials to read data from the ShakeTable on behalf of Alice. It also illustrates how the attribute “Dependency” enables an automatic reasoning for determining all required access rights only by one query. The job requests the credentials again from Alice’s Delegation Service as described earlier. Once Alice’s job has obtained the required delegation credentials from Alice’s Delegation Service, it can get its input data and continue on to completing its task.

This process continues to obtain all the required credentials iteratively until the job execution has completed. Table IV shows the results of appropriate queries made to Delegation Services to determine what credentials are required to authorize Alice’s job to perform the computation task and store output data in a proper location.

Ontologies are also utilized by Alice’s Delegation Service to decide when a credential should be issued. Fig. 6 shows a sample query to answer the question, “What are the parameters and constraints of delegation which authorize Alice’s job, identified by JOB_1_DN, for job submission?”.

SELECT ?Subject ?Identifier ?capability ?validNotBefore ?validNotAfter ?isApplicableTo ?isDelegatableTo
WHERE {
?delegaion:isDelegatedTo ?Subject.
?idn:isIdentifiedBy ?Identifier.
?na:isValidNotAfter ?validNotAfter.
?ap:isApplicableTo ?isApplicableTo.
?de:isDelegatableTo ?isDelegatableTo.
FILTER regex(str(?capability), "Submission_JOB")
FILTER regex(str(?Identifier), "Job_1_DN")
FILTER regex(str(?validNotBefore), "2006-04-10T12:00:00")
FILTER regex(str(?validNotAfter), "2006-04-11T12:00:00")
FILTER regex(str(?isApplicableTo), "NoCapability")
FILTER regex(str(?isDelegatableTo), "NoCapability")
}

Fig. 6. Query on delegation description

The results of the query in Table V specify the constraints that should be considered when a delegation credential is issued for Alice’s job identified by “JOB_1_DN”.

VI. SECURITY CONSIDERATIONS

We must recognize that any form of delegation entails some risks that we cannot eliminate completely. If the job runs at a compromised site, it may be hijacked and its credentials may be misused. However, we believe that on-demand restricted delegation provides mechanisms to help us manage these risks.
In particular, the callback mechanism enables us the possibility of detecting misbehaviors and raising appropriate alarms to indicate that a job might be hijacked and compromised. The Delegation Service can log what credentials the job obtained, so after the compromise is detected, we can determine how the compromise spread and what credentials need to be revoked. Even more, we can consider:

- An exception mechanism for handling unexpected situations by blocking the job, notifying the user, and waiting for the user’s decision on how to proceed, or by granting the rights so the job can proceed, but notifying the user, and letting the user terminate the job if it has overstepped its bounds.
- A learning mechanism that can be used to modify the policies according to the credentials that the job required when it last ran and even build up the policies by leveraging an interactive mechanism which involves the owner’s approval for provided credentials over and over until all needed credentials are acquired.

These mechanisms assist the user in building up restricted policies to limit the vulnerability of delegated credentials and allow the user to monitor jobs to detect when changes in policy are required or when jobs are misbehaving.

VII. IMPLEMENTATION AND VALIDATION

Currently, we demonstrate the feasibility of our approach through a prototype implementation, which we have successfully tested with a simple Grid application. In our test scenario we have implemented a client program for performing a third party data transfer in Grids, which is run with no pre-generated proxy certificates. It further obtains credentials to use with GridFTP [16] services on demand and solely for the particular files specified by its command line arguments. For this we have implemented a Delegation Service and a communication protocol for requesting and exchanging credentials as described in [17]. In our implementation we have used the Ontology Web Language (OWL) [18] for creating delegation ontology which includes descriptions of classes, properties and the instances of delegation ontology described earlier. The SPARQL query language [19] is also used to make efficient, adaptive, and intelligent queries on system description and delegation ontology. We have also used the Protege OWL API [20] to develop a set of software components for generating, parsing and evaluating queries and also populating and instantiating ontologies.

VIII. CONCLUSION AND FUTURE WORK

In this paper we claimed that the lack of a flexible least privilege delegation mechanism necessitates the design of an on-demand delegation framework. We believe that provisioning restricted delegation in a flexible way is the most significant and challenging issue for delegation in Grids.

We proposed an on-demand delegation framework that utilizes a callback mechanism for provisioning of restricted credentials containing only the rights that are actually needed. We described how ontologies can be utilized for determining the access rights required by a delegate to complete a task and further how a Delegation Service can use ontologies for performing delegation. Approaching on-demand delegation has benefits of real-time control and auditing at the Delegation Service. However, it may hurt performance if it requires multiple callbacks to the Delegation Service for obtaining rights. In this regard, one strategy to optimize on-demand delegation would be delegating more rights to the job either at the time of launching or during each callback, so that a delegatee does not have to callback to a Delegation Service so often. Determining this set of rights in an optimal way is a challenging issue that can be considered part of the future work of this research.

Rein [13], the policy framework described earlier, supports an ontology-based mechanism for describing delegation and therefore it has certainly strong potential to be used in our proposed on-demand delegation. Then one future work would be investigating on how to use Rein policy framework in this framework. The level of granularity of resource and system description and access rights can also affect the complexity of restricted delegation and even the security of the whole system. Therefore, one additional target point of future work would be the optimization of the delegation description to find an optimized level of granularity that yields a less complex and more usable system without significantly compromising the security.

ACKNOWLEDGMENT

Adam J. Lee was supported by a Motorola Center for Communications graduate fellowship and by the NSF under grants IIS-0331707, CNS-0325951, and CNS-0524695.

REFERENCES


Paper II: Context-Aware, Least-Privilege Grid Delegation

Dynamic, Context-Aware, Least-Privilege Grid Delegation

Mehran Ahsant #1, Jim Basney ∗2, Lennart Johnsson #3

#Center for Parallel Computers, Royal Institute of Technology
Valhallavagn 79, 10044 Stockholm, Sweden
1mehrana@pdc.kth.se
3johnsson@pdc.kth.se

∗National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign
1205 W. Clark St., Urbana, IL 61801 USA
2jbasney@ncsa.uiuc.edu

Abstract—Performing delegation in large scale, dynamic and distributed environments with large numbers of shared resources is more challenging than inside local administrative domains. In dynamic environments like Grids, on one hand, delegating a restricted set of rights reduces exposure to attack but also limits the flexibility and dynamism of the application; on the other hand, delegating all rights provides maximum flexibility but increases exposure. This issue has not yet been adequately addressed by current Grid security mechanisms and is becoming a very challenging and crucial issue for future Grid development. Therefore, providing an effective delegation mechanism which meets the requirements of the least privilege principle is becoming an essential need. Furthermore, we are witnessing a phenomenal increase in the automation of organizational tasks and decision making, as well as the computerization of information related services, requiring automated delegation mechanisms. In order to meet these requirements we introduce an Active Delegation Framework which extends our previous work on on-demand delegation, making it context-aware. The framework provides a just-in-time, restricted and dynamic delegation mechanism for Grids. In this paper we describe the development of this framework and its implementation and integration with the Globus Toolkit.

I. INTRODUCTION

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. However, today’s Grid architectures are not capable of supporting dynamic, agile federation across multiple administrative domains and the main barrier, which hinders dynamic and secure federation over short time-scales, is security.

Delegation is a powerful means to support expanding and propagating trust relationships in Grids. Jobs running on remote sites use delegated rights for accessing resources on behalf of their owners. Supporting this capability in a reliable, secure, and functional manner requires that we address the challenges of restricted and dynamic delegation.

In this context we need to meet the requirements of the least privilege principle, stating that we should grant only those rights required to perform a required action to minimize exposure to misuse. Unrestricted delegation, even in a highly confined and uniformly controlled environment, creates security risks. In widely-distributed environments such as Grids, there is greater possibility of losing trustworthiness by some irregular actions (for example, system compromise by an outside attacker).

However, as we have described in our earlier work [1], performing delegation in a restricted but static fashion may cause task execution to fail due to insufficient delegated rights, because the required rights are difficult to predict in advance for dynamic Grid environments, where the resources used by the task may be chosen at run-time based on availability, cost, performance, and other concerns. Agile and automated delegation requires embedding delegation functionality in services. Delegation functionality can be provided by the container that hosts the service [2]. This kind of delegation service is becoming an essential security component in many distributed security infrastructures [3][4].

All the above mentioned requirements led us to recognize the need for a new model of delegation to be used by current Grid systems—a framework which supports a context-aware, restricted, dynamic and automated delegation for Grids. In this paper we describe the development of this framework which approaches a just-in-time delegation to enable delegating rights and privileges to potential Delegatees as required for completing tasks.

Our paper proceeds as follows. In Section II we describe the Active Delegation Framework as our approach to meeting the requirements described earlier. This section also describes the architecture of this framework for a generic Grid infrastructure. Section III describes the architectural components of the active delegation framework in a real Grid system. In this section through a real Grid scenario we illustrate the practicability of our approach and describe how the components of the framework are implemented and can be integrated into a real
Grid infrastructure. In section IV, we discuss related works. Section V gives a discussion on our approach from different aspects such as scalability, security and performance. Finally, in section VI we conclude and describe future works.

II. ACTIVE DELEGATION FRAMEWORK

We introduce an active delegation framework for Grids, in which a restricted, dynamic and automated delegation can be supported. This provides a context-based and on-demand delegation paradigm which enables a just-in-time acquisition of delegated rights in an associated context. Throughout this approach, when a delegated task is being completed, the Delegator requests required rights from the Delegator and if the request is approved, in an associated context, the rights will be provided to the Delegatee. This implies delegating rights iteratively as needed until the task has completed. This also addresses the requirements of the least privilege principle, as no right through the delegation is given to a Delegatee unless it is needed and the “need” can be approved.

We elaborate upon our approach by describing a simple scenario. When the Delegatee initiates execution of a delegated task it may only obtain a minimal set of rights which allows it to prove that it needs to act on behalf of its Delegator. The Delegatee has no additional rights to access resources and therefore needs to obtain them later when they are required in order to proceed to task completion. Thus, when required privileges are determined, the Delegatee contacts the Delegation Service and requests the required rights. The Delegation Service receives the request and consults a policy engine to check the request against the delegation policies established by the Delegator. The policies can be fulfilled only if the context in which additional delegated rights are requested has been established. Thus, if the request fulfills the circumstances, the Delegation Service generates appropriate credentials with embedded rights and sends them back to the Delegatee. The Delegatee can therefore continue its execution.

Central to this approach is both: a mechanism which enables “on-demand” requesting and provisioning of delegated credentials and a way of establishing “contexts” in which the need for additional delegated rights can be verified. By utilizing a call-back mechanism the Delegatee would be able to make a request for the required rights “on-demand” and also allows the Delegator to provide them just-in-time. Monitoring, collecting and processing the status of jobs, resources and service can also be used for establishing “contexts”. Contexts can evolve through the task life-cycle and can be considered as any characterizing information about protected resources and surrounding environments in Grids. Contexts are basically associated to: a) resources to be controlled, b) Delegatees who make requests to access protected resources and c) Delegators whom on their behalf access to resources is being made.

The approach of the active delegation model is designed to address the requirements of active security systems. The idea of active security models was preliminary presented in [5] as a security model which distinguishes from a passive security model by introducing contexts in which permissions can be activated and primarily serves the function of maintaining permission assignment. In active security systems any assigned permission might be activated or deactivated when its associated context is evolving, and operations are then permitted if the associated permission is currently activated. In order to address the needs for providing active security models, some access control models have been introduced to enable distinguishing between permission assignment and activation by considering different levels of context when processing an access operation on an object [6][7][8].

The work presented in this paper is the continuation of our previous work in the context of Grid delegation mechanisms presented in [1]: a so-called “on-demand” delegation mechanism which is an efficient, scalable and manageable approach to provide a restricted and flexible delegation framework for Grids. The main contribution of this work is adding sufficient support to the on-demand delegation framework to make it capable of asserting the “need” for requested delegation rights. To achieve this, the main effort has been moving toward an active delegation model described earlier by introducing “contexts” in the on-demand delegation framework. Furthermore this paper also gives a full description of architectural components and implementation details of our developed active delegation model.

A. Architectural components

In this section, we decouple the framework described earlier into its architectural components. We also explain how these components are related and should interact with each other. What Figure 1 illustrates are the components of the active delegation model (gray boxes) in a generic Grid infrastructure. Those generic architectural components (white boxes) illustrated in this picture can be provided by many currently used Grid systems. Later in this paper, when we explain the development of the active delegation framework, we describe how the components with generic names are replaced by the real components in a widely used Grid infrastructure.

![Fig. 1. Architectural components](image-url)

The components of the active delegation framework are the following:

1) Delegation Service: The Delegation Service is a web service for delegating credentials to potential requesters (Delegatees) that can be run by a user or hosted on behalf of many
users. Delegation Services can delegate credentials based on a given delegation specification attached to the request. Along with the request, the Delegation Service requires the context information in which additional delegated rights are requested. The Delegation Service must also query the policy engine to check if delegation policies can be fulfilled in a particular context and delegating additional rights is therefore permitted.

2) Context Manager: The Context Manager is a service which is either notified by other services with new context information or makes a query to directly obtain the context information of particular services, resources or job instances. This context information along with the delegation request are then sent to the Delegation Service to make the appropriate context in which a delegation request needs to be verified. The Context Manager might collect a variety of information, such as a decision made by the scheduler on where to run a job or the closest replica of a database which contains the input data as selected by a Replica Location Service. Furthermore, context information might be created to verify a delegation request which is made to obtain additional rights for the sub-jobs created from a parent job. The Context Manager would typically be hosted with the services it obtains information from.

3) Policy Engine: A policy framework is required to set up delegation policies. As part of this framework there is a Policy Engine which can be used by the Delegation Service to check a delegation request against the set of established delegation policies. Delegation policies are established either by the local administrator or the Delegator herself, based on templates provided for common use cases. Therefore, in addition to the “context” in which delegating rights is permitted, the Delegator and local administrator can also impose their own policies before delegating rights to the Delegatee.

4) Credential Exchanging Protocol: The on-demand delegation framework requires a protocol for requesting and receiving delegation credentials as needed. The callback mechanism leveraged by this framework requires a flexible and efficient protocol that enables exchanging different security credential formats between the Delegatee and the Delegation Service dynamically over heterogeneous security domains.

5) Collection of Information: Depending on what services are called or which resources are being used, the Context Manager may be required to leverage different ways of communication and information collection. The Context Manager further needs to convert collected information into a single internal format for processing.

III. A GRID USE-CASE

In this section, we first describe the development of the active delegation model by thoroughly illustrating its design and implementation into a Grid system. We describe also a Grid scenario in which the potential benefits of active delegation can be illustrated. Although the design of this framework can support more advanced scenarios, in this section for the sake of simplicity and to avoid illustrating complex policy schema we only focus on a simple Grid job submission.

A. Design and Implementation

Figure 2 depicts the design of the active delegation framework into a Globus\(^1\) GT4 environment. It illustrates the instances of the more generic architectural components and communication protocols described in section II-A. This figure also depicts the flow of information and the sequence of events which we will further describe in section III-B.

![Integrating active delegation model into GT4](http://www.cs.umbc.edu/ lkagal1/rei/)

1) Rei Policy language: Rei is a policy framework that we use for reasoning over delegation policies [9][10]. It has strong potential to meet the requirements of the active delegation framework. The Rei policy framework permits specifying, analyzing and reasoning over declarative policies defined as norms of behavior [11][9]. Rei has been aimed at leveraging the potential of Semantic Web and ontologies in describing security requirements and specifying security policies. In our earlier work [1], we also showed a strong potential of ontologies and Semantic Web in development of an on-demand delegation framework. Rei policies restrict domain actions that an entity can/must perform on resources in the environment, allowing policies to be deployed as contextually constrained deontic concepts, i.e., permission, prohibition, obligation and dispensation.

In the Rei policy specification\(^2\), context conditions can be specified by defining one or more constraints. A constraint, which may be simple or Boolean, i.e., the boolean combination of a pair of simple constraints, defines a set of actors or a set of actions that fulfill a certain property. Rei’s support for contexts makes it well-suited for our purposes, though evaluation of other policy languages is planned for future work.

The class Action is one of the most important concepts in the Rei specifications as policies are described over possible actions in the domain. Rei includes a representation of actions that allows more contextual information to be captured. Action has two main subclasses: DomainAction and SpeechAct.

---

More important to our framework are the SpeechActs. The SpeechActs are primarily used for dynamic and remote policy management. There are six subclasses of SpeechAct: Delegate, Revoke, Request, Cancel, Command, and Promise. One property encompassed by SpeechAct is condition which states the constraints that the sender is adding to any SpeechAct. That is, the speech act is only valid when these constraints are true. An entity can “request” another entity for a permission, which if accepted causes a “delegation” to be performed. Delegation of a SpeechAct can also lead to a new permission.

In order to use Rei we have extended SpeechAct delegation by incorporating a new property called pre-condition, which is defined as the conditions that need to be true before the delegation speech act can be performed. These pre-conditions are in fact constructing the “context” in which delegation can be activated and a delegation speech act can be triggered in response to a request. We need to recognize the distinction between the pre-condition property and the condition property defined originally in the Rei specification. The pre-condition determines when rights may be delegated, while the condition determines when rights may be used. Grid tasks can take longer than the validity interval of a context for which delegation is activated, which implies that constraints that must be satisfied before delegating rights should not necessarily remain true to make the delegated rights usable.

As a motivating example, assume that a Delegation Service recognized the need for performing a delegation for enabling access to a database because the first targeted database is unavailable; after performing the delegation administrators may fix the problem and make the service available again. This makes a new context in which the previous delegation would no longer be valid, but the active task should be able to complete its session with the backup database. Using a “pre-condition” can enable this use case.


3) Delegation service: The Delegation Service used in this framework is a token service that delegates credentials according to the delegation specification. This token service supports a WS-Trust interface for requesting those security tokens profiled by OASIS, such as SAML and X.509 certificates. The GrDP and Delegation service together provide a flexible mechanism to support an on-demand delegation.

4) Gridway system: GridWay is a light-weight meta-scheduler that performs job execution management and resource brokering. GridWay is specifically designed to work on top of Globus services. It allows unattended, reliable, and efficient execution of jobs, array jobs, or complex jobs on heterogeneous, dynamic and loosely-coupled Grids. GridWay performs all the job scheduling and submission steps transparently to the end user and adapts job execution to changing Grid conditions by providing fault recovery mechanisms, dynamic scheduling, migration on-request and opportunistic migration.

5) Context manager: The Context Manager can collect appropriate information from the resources and jobs. The context manager may also be configured to be notified by other services when the status of resources or jobs has changed. It can use a standard interface (WS-Notification). If there is no notification support, the service can make a direct query to get the status of job or resources. Context information is then constructed in the format of “Constraints” used in the Rei policy format to be used by the policy engine for asserting a request. The context information can be obtained either from the GridWay service or the built-in GT4 services, like MDS (Monitoring & Discovery System) and WS-GRAM. The development of the Context Manager is currently tied to the GT4 services and can construct only simple context information. The context information when collected, enriched and converted to Rei constraints is signed by the Context Manager for the assertion of delegation policies.

The Figures 1 and 2 illustrate the components and the interactions of the active delegation model only for executing jobs on a single site. However, a job running on the Grid may need to visit different sites when completing tasks. Figure 3 roughly describes how the active delegation model is used in a scenario which involves more than one site to execute a job. As it is depicted in this figure, when a job migrates from Site1 to Site2 for completing a task execution, upon requiring additional rights, the execution service of Site2 needs to make a delegation request to the Delegation Service running on the job’s home site (Site1). The context required for verifying the delegation request, should also be requested by the Site1 Delegation Service from the Context Manager of Site2.

We have developed a proof-of-concept implementation over the Globus GT4 environment for the components described earlier. An implementation of the GrDP is used for requesting and exchanging credentials between jobs and delegation services. This implementation profiles the WS-Trust specification for SAML assertions, X.509 certificates and proxy certificates.
It also implements the GSI delegation mechanism for proxy certificates. The current implementation of the Delegation Service generates an independent proxy with an embedded SAML assertion which contains appropriate authorization statements for accessing resources. The current implementation of the Context Manager only supports simple scenarios and can be used to generate very simple contexts. It uses the “gwsps” and “gwhost” system monitoring commands provided by the Gridway system to gather information about jobs and resources.

B. Scenario

We now describe a scenario to illustrate the active delegation framework. Alice needs to render large images by using some rendering services provided by the Grid. A rendering service may also need to get its input data from a database. There is usually more than one site that can host the job and therefore Alice may use the Gridway meta-scheduler for the job submission. Depending on some criteria specified for the meta-scheduler, Gridway chooses one site out of all available sites to run Alice’s job. It may also exist more than one replica of input data that the job can use. For this, Alice’s job needs to call a Replica Location Service (RLS) provided by GT4 to locate the closest database service which contains a replica of the required input images.

As it is typically used, the GT4 security infrastructure requires Alice to generate a Proxy Certificate (PC) [13] for her job which gives the job full permission for acting on behalf of Alice (impersonating Alice’s identity). This does provide dynamism, but does not restrict delegation in any way. Alice could also restrict the PC by embedding policies which restrict the usage of the PC for only an intended task or specific resource access; however, Alice can not anticipate which sites are going to be chosen by the scheduler or which database replica will be selected by the RLS service. Alice therefore needs to delegate to her job the permissions for accessing all the sites and all the databases. The rest of this section describes how the active delegation framework can be leveraged to provide a restricted and still dynamic delegation from Alice to the job.

Alice uses a client program and the Gridway system to submit her job (Step 1a of Figure 2). The client program uses a policy template and updates the Rei policy repository with a Rei delegation policy as depicted in Figure 4 (Step 1b). This can enable delegating more requested permissions to the job by Alice. It may also include any local delegation policy, like the simple one depicted, which implies that a delegation would be valid as long as the job (SimJob) is active. However, delegation can be triggered or activated only if the pre-conditions are fulfilled. In this case it needs to be asserted that SimJob requires additional permissions. The client also asks the Context Manager to initiate a new context and associate that to the SimJob (Step 1c).

Gridway examines different Grid resources to select one to which the simulation job can be submitted. The Context Manager watches the decisions that the Gridway system makes regarding where to submit the job (Step 2). Now the site which will host the simulation job is determined, though a delegation from Alice is also required for the job to be submitted. The required rights can be requested from the Delegation Service running on behalf of Alice, if the request can be approved by the Delegation Service. Figure 5 shows such a request in Rei format. For this, the WS-GRAM makes a delegation request in GrDP format and sends that to the Delegation Service (Step 3). The Delegation Service can approve the request if the Context Manager can assert the need for the requested rights. For this, the Delegation Service makes a query to the Context Manager to obtain the context information for this specific job (Step 4). The Context Manager has already created an associated context, which asserts the “need” (scheduler’s decision) for the required rights. It converts the context information into a single internal format (Rei constraints) as depicted in Figure 6, embeds them in an X.509 certificate, signs it and sends that back to the Delegation Service. The Delegation Service receives the request, verifies the signature and sends the constraints along with a delegation query to the policy engine (Step 5). The policy engine performs reasoning over the delegation policies to decide whether the requested rights can be delegated to the SimJob (Step 6). If there is no conflict with the local delegation policies established by Alice or the administrator, the Delegation Service, according to the delegation request specification generates the requested credential, makes a GrDP response message and sends it back to the WS-GRAM to submit the job (Step 7).

In a multi-site scenario, if the job migrates to another site the delegation SpeechAct depicted in Figure 4 needs to be updated in a way to impose new delegation pre-conditions as depicted in Figure 7.

Fig. 4 The SpeechAct delegation associated to the SimJob implies that delegation is valid as long as the SimJob is active and delegation can also be triggered only if the pre-conditions are fulfilled—in our scenario, if the SimJob needs to be submitted for execution.

Fig. 5 The request in Rei format, which asks permission for job submission.
The notation of context and active security system are observed in many works. In some of them diverse delegation models are also supported. However, as far as we are aware, the delegation concept in those works has not been investigated from the perspective of our own work presented in this paper. Most of the works in this context are based on the RBAC96

Delegation Issuing Service (DIS) [4] is a service that issues X.509 Attribute Certificates (AC) on behalf of an Attribute Authority (typically a manager), integrated into the PERMIS authorization infrastructure. DIS only issue Attribute Certificates and lacks a standard protocol for interactions with an Attribute Authority who requests to issue ACs. It still needs a human to decide what is the least set of privileges to delegate to another entity and there is no support for an automated logic that can determine what are the least privileges to delegate.

The notion of context and active security system are observed in many works. In some of them diverse delegation models are also supported. However, as far as we are aware, the delegation concept in those works has not been investigated from the perspective of our own work presented in this paper. Most of the works in this context are based on the RBAC96
family [22], which supports role activation within sessions to provide an active model of authorization management and access control models, such as the TeaM-based Access Control (TMAC) model, which is introduced in [23] and extended in [24] to a family of context-aware access control models. TMAC basically recognizes the importance of context information for just-in-time activation of permissions. Similar to the TMAC approach, the Task-Based Access Control (TBAC) model [25] is also used for management of authorizations that encapsulate a group of permissions, in a way that they are turned-on only in a just-in-time fashion and synchronized with the processing of authorizations in progressing tasks. The OASIS RBAC model [26] is also an extension to the role-based access control architecture. OASIS RBAC does not use role delegation but instead defines the notion of appointment. One could consider applying our work to these systems by employing an active security model to determine when a team or session needs to be created or a new member needs to be joined into an established context (team).

None of above approaches addresses all the requirements of delegation described earlier. For the works around active security systems, although they provide support for contexts, they lack the ability to determine the “need” for establishing contexts “dynamically”, which is an essential requirement for highly dynamic environments like Grids. This is where we introduce the Context Manager as an architectural component. We also recognize a very special requirement of Grid delegation when we distinguish the constraints applied at authorization time from the constraints applied at delegation time. The former implies that a delegation must be valid as long as the conditions are fulfilled, while the latter specifies that a delegation can be activated only when a set of pre-conditions are met. This is where we extend the Rei policy language to incorporate “pre-conditions” in the delegation SpeechAct as described in section III-A.1. What distinguishes our work from others is the adaptation of the active delegation model to Grids and providing a supporting framework which can be integrated into current Grid systems and leveraged by existing security mechanisms. Our framework further provides more support for observing and auditing the delegation process adapted to the dynamic requirements of Grid applications.

One part of our approach, which provides an on-demand provisioning of delegated credentials, is in some ways similar to the more general notion of trust negotiation [27]. In trust negotiation, two strangers carry out a bilateral and iterative exchange of attribute credentials to gradually establish trust in one another. Our approach complements trust negotiation in that our work could be used in a system employing trust negotiation to determine when sensitive attribute certificates of a user should be accessible to his jobs.

V. DISCUSSION

What a just-in-time and active delegation mechanism provides is an efficient approach for performing restricted delegation in a dynamic and automated fashion, because in practice we expect a small number of credentials to be required. We address the important concern of scalability by distributing Delegation Services and Context Managers across the Grid, associated with different resources. Delegates should need to query only a few Delegation Services in practice. Delegation Services need only know about local policies. This provides manageability by providing local policy control points (the Delegation Service). Establishing contexts by the Context Managers can help Delegation Services to adequately perform decision making even in highly changing environments with dynamic execution paths. By using call-backs, our delegation framework never leaves a job stranded without credentials, but policy dictates the least privilege credentials that are delegated. When a job needs credentials that policy won’t allow, we can put the job “on hold”, notify the user, and wait for user intervention. If the user approves, the job can then proceed, thereby providing a very effective approach compared to other delegation mechanisms.

The active delegation mechanism is not actually aimed at replacing any existing delegation mechanisms mentioned in section IV. The concept and the framework are rather modeled and designed to be used to improve and complement the delegation mechanism used by existing security systems, such as GSI. The mechanism provided by this work does not either mandate a Grid system to use any particular authorization system: it can be leveraged by any existing authorization system used by Grids to provide a dynamic and restricted delegation mechanism. The framework is also not limited only to use a particular scheduler or particular information service for collecting information and establishing contexts. It is rather open to use any other source of information (Grid information services) like the native Grid services provided by GT4 environments or other Grid middlewares.

Trust is an important issue in this framework. The Delegation Service has to establish a trust relationship with the Context Manager. That is, the Delegation Service needs to ensure the authenticity of the Context Manager and the integrity of context information. GSI provides mature support for authentication and integrity checking through digital signatures. The context information collected and represented by the Context Manager should also be trusted. This information is used to assert a delegation request and therefore crucial for trustworthy policy evaluation. Trust establishment is transitive in this case, as trusting the Context Manager implies trusting the information services like the Gridway system and other source of information.

This approach has benefits of real-time control and auditing at the Delegation Service. However, it may hurt performance if it requires multiple callbacks to the Delegation Service for obtaining rights. In this regard, one strategy to optimize delegation would be delegating more rights to the job either at the time of launching or during each callback, so that a Delegatee does not have to callback to a Delegation Service so often. Determining this set of rights in an optimal way is a challenging issue that can be considered part of the future work of this research.
VI. CONCLUSION AND FUTURE WORK

In this paper we described new requirements imposed by Grid systems for providing an agile, automated and restricted delegation mechanism. We introduced the notion of an active delegation (just-in-time) model to address the issue. We described an active delegation model as a context-based approach which supports a call-back mechanism for provisioning on-demand and restricted delegation credentials. We described how by leveraging this approach permissions could be given to the Delegates on request, just-in-time and only for an intended task. We described the architectural components of our proposed framework and how to deploy this framework in a GT4 based Grid environment. We made use of Rei, a context-based policy framework to express and deploy delegation policies. We also used GrDP, a credential exchanging protocol with support for the WS-Trust interface for requesting and receiving delegation credentials. We described a proof-of-concept implementation of our proposed delegation framework with a use case scenario for job submission in a GT4 based Grid system.

The active delegation framework has the potential to be integrated into other Grid systems and middlewares. Therefore as future work we plan to investigate how to use the active delegation model in some other Grid systems like UNICORE, gLite5 and NAREGI6. We also described Rei and its potential to express just-in-time delegation in the active delegation framework. However we believe that any other policy language which incorporates the concept of contexts and provides support for delegation can also be used to describe active delegation policies as well. Therefore, one future work would be investigating how to use other policy languages in this framework. One alternative would be XACML, a standard and general purpose policy system designed to support the needs of modern authorization system.

REFERENCES

Paper III : Workflows in Dynamic and Restricted Delegation

Abstract—Delegation is a key facility in dynamic, distributed and collaborative environments like Grids and enables an effective use of a wide range of dynamic applications. Traditional delegation frameworks approach a top-down model of delegation for delegating rights from a superior to a subordinate in advance before a delegate starts off a delegated task. However, a top-down model of delegation cannot meet all the requirements of dynamic execution of distributed applications, as in such environments, required access rights for completing a task cannot easily be anticipated in advance. Delegating fewer rights than required for completing a task may cause the task execution to fail while delegating more rights than needed may threaten abuse by malicious parties. It is therefore reasonable and more robust to utilize a mechanism that allows determining and acquiring only required rights and credentials for completing a task, when they are needed. This is what we call an on-demand delegation framework, which realizes a bottom-up delegation model and provides a just-in-time acquisition of rights for a restricted and dynamic delegation. In this paper we elaborate the concept of bottom-up delegation and describe how an on-demand delegation framework can leverage workflows to meet the requirements of the least privileges principle. We also discuss the vital need for dynamic and adaptive scientific workflows to support an on-demand delegation framework. We present three different models of bottom-up delegation, which cover a wide range of usage scenarios in Grids and dynamic collaborative environments. Using a standard RBAC authorization model and a graph-based workflow model (DAG), we define and analyze a formal model of our proposed bottom-up delegation approach.

I. INTRODUCTION

In large distributed computing environment like Grids, users or applications may need access to resources that belong to different organizations. Jobs submitted by users of these systems may take long periods of time to execute, and resource providers usually require some form of authentication of users and authorization of requests to grant access to use the requested resources as described in the request. Requests may be generated dynamically during the execution of an application and need user approval before being submitted to a resource broker or resource provider. However, the user may not be directly connected after submission of a request for execution of an application.

A common solution to the problem of disconnection is to delegate the authorization to an agent (program) that acts on the users’ or applications’ behalf and that is less likely to be disengaged at any time [1]. Delegation is a key facility in such environments that makes possible an effective use of a wide range of dynamic Grid applications. However, the least privilege principle, stating that we should grant only those rights required to perform a required action to minimize exposure to attack, is becoming one of the most important security aspects in regards to delegation.

In existing Grids, however, delegation is performed either in a static and restricted manner or a fully dynamic and unrestricted fashion [2]. Performing delegation in a static manner cannot meet all the requirements of dynamic execution of distributed and collaborative applications, mainly because required access rights for completing a task cannot easily be anticipated in advance; delegating rights in a dynamic but unrestricted manner is undesirable because it threatens abuse by malicious parties.

We believe that performing delegation on-demand would be a significant approach to achieve a restricted delegation in dynamic and distributed collaborative environments, in which the set of rights to be delegated cannot be determined in advance and dynamically changing circumstances during the execution may require a new set of rights and privileges to be delegated [2].

On-demand delegation provides a restricted, dynamic and automated delegation through a just-in-time acquisition of delegated rights. Using this approach, during the execution a delegate requests for required rights from a delegator and if the request is approved for an associated context, the rights will be provided. This implies delegating rights iteratively as needed until the task has completed and addresses the requirements of the least privilege principle, as no right through the delegation is given unless it is needed and the need is approved [3].

On-demand delegation requires a delegate to contact a superior to ask for additional rights and this gets more parties involved in the delegation process which raises new concerns. First off, in this model there is a need for setting up a process or a service that can automatically delegate a superior’s rights to a delegate without the superior being involved. This obviously creates the need for a policy engine that enables the

1The work presented in this paper has been partly supported by BalticGrid-II, the project funded by the European Commission within the framework of the 7th Framework Programme (contract number 223807).
service to decide if the subordinate should have the requested rights. Such established policies enable a superior to maintain the ultimate control over delegation. Another issue to resolve is how in such framework a delegation request can be approved and how the subordinate can be trusted for telling the truth about which rights is required and why. Furthermore this approach may raise some concerns when too many callbacks are required for obtaining rights.

In order to address some of above issues, in our previous work [3], we extend our delegation model to incorporate contexts as sufficient support for asserting the need when additional rights are requested. The main contribution of this work is to address the remaining issues by introducing workflows in an on-demand delegation framework to achieve two main goals: i) optimizing delegation by delegating more rights to a job either at the time of launching or during each callback, so that a delegate does not have to make callbacks to a delegation service so often (determining this set of rights is a challenge which we address in this paper), and ii) ensuring a Delegation Safety Invariant (DSI) factor, as we describe in section II, which is about how to ensure that delegated rights are bound to a particular context and do not violate security policies.

Our paper proceeds as follows. In Section II, we present the concept of bottom-up delegation as a new model for performing delegation. In section III, we give an introduction on workflows in general and we describe how workflows can be used when providing restricted delegation in dynamic environments. In section IV, we give a formal definition for the authorization model and the workflow model used in our framework. In section V, we describe three different models of on-demand delegation using workflows. In section VI, we discuss some important aspects of leveraging on-demand delegation in scientific workflows. In section VII, we present some related works and finally in section VIII, we conclude our work and describe the future work.

II. BOTTOM-UP DELEGATION CONCEPT

The intuition of the on-demand delegation model is a novel concept of delegation that we call bottom-up delegation. It is well explained when compared to a traditional model of delegation in which a top-down model is approached for delegating rights from a superior to a subordinate in advance before a delegate starts off a delegated task. In contrast, in a bottom-up delegation a subordinate needs to ask a superior for acquisition of sufficient rights when it needs to perform an action on behalf of a delegator. Although top-down delegation models have been used for a quite long time by many security systems, for some use-cases, like those described earlier, they are insufficient and the need for a bottom-up model which enables delegating rights just-in-time and in response to a valid request is highly motivated [2].

Each of these models have their own pros and cons. A top-down delegation, due to its static nature and representation, can be designed and implemented in a more straightforward way than a bottom-up delegation, as superiors can do the whole process on their own without involving the delegates in the process. It is also more straightforward as delegates never need to take the rights up with delegators. However, the principle of least privileges, as one of the most important security concerns, requires a user (delegate) be given no more privileges than necessary to perform a job. In the context of delegation in order to meet the requirements of this principle we require i) identifying what the user’s job is, ii) determining the minimum set of privileges required to perform that job, and iii) restricting those privileges to a particular delegate and a specific domain or context. For dynamic environments, however, a top-down delegation model is insufficient and unable to meet all these requirements as we discuss in [2][3]. Therefore, a better strategy would be either combining a bottom-up delegation with a top-down model by delegating certain restricted rights initially to the delegate (top-down), and if it turns out that not enough have been given, the delegate then later asks for more rights (approaching the bottom-up model); or just completely implementing a bottom-up delegation and requiring delegates to ask for rights just-in-time. The delegation model presented in this paper covers both of these strategies.

III. WORKFLOWS

Workflows traditionally have been used by business organizations for modeling and controlling the execution of business processes to achieve a business objective [4]. In workflow literature each business process can be defined and modeled as: i) a collection of tasks or activities which it encompasses, ii) the applications that should perform each task, and iii) the data required for performing tasks.

In fact, a workflow separates any given organizational process into a set of well-defined tasks, with clear relations and dependencies, as logical steps or description of pieces of work that contribute toward the accomplishment of the whole process. In this context, a workflow management system (WFMS) is the unit to coordinate the execution of tasks that constitute the workflow. Generally speaking, tasks in a workflow are inter-related in such a way that initiation of one task is dependent on successful completion of a set of other tasks, and therefore, the order in which tasks are executed is very important and is controlled by a WFMS [5][4].

Recently, workflows have also emerged as a paradigm for representing and managing complex distributed scientific computations and accelerating the progress of scientific activities. For scientific applications a workflow paradigm can also be beneficial from many aspects such as building dynamic applications for orchestrating distributed systems, utilizing resources, reducing execution costs and even promoting inter- and intra-organizational collaborations [6][7]. Similar to business workflows, scientific workflows are also concerned with automation of processes and enabling the composition and execution of complex scientific tasks.

In scientific workflows, each step specifies a process or computation to be executed e.g. a software program or a Web service. The workflow links the steps according to the data flow and dependencies among them. Processes contain tasks,
which are structured based on their control and data dependencies. This gives a high-level and structural specification of processes which are used by the WFMS to define the way that tasks are ordered, scheduled and should be executed. Thus, the benefits of using workflow management systems in an on-demand delegation model are their support for defining, structuring, executing and controlling the processes (job definitions) which encompass tasks. We will show how these can help out determining that requests for additional privileges are actually made just-in-time, delegated privileges are not more than required for executing the process (job) and delegated rights are bound to a set of intended task(s).

For structuring and modeling workflows, various approaches have been used in a broad range from using a simple graph to using advanced Petri-net graphs [8][9]. One promising approach, however, has been using Directed Acyclic Graphs (DAG), a primarily mathematical abstract entity which possesses a very intuitive way of visualization that can be handled easily even by non-expert users. Many scientific workflow approaches are built on this special subclass of graphs. DAGs may restrict the kinds of workflows that can be modeled, however as they are very easy to implement they have been used in many workflow management systems for modeling scientific application [6][9]. In this paper we are not going to describe the details of these modeling as they are well covered in this literature by other authors [8][7].

IV. FORMAL DEFINITIONS

In this section, we give a formal definition for a bottom-up delegation model using a graph-based workflow model integrated with an RBAC authorization system.

A. RBAC authorization model

Our model is based on the standard Role Based Access Control model (RBAC) [10], which presents an authorization model that greatly simplifies the security management by including three basic sets of entities: users $U$, roles $R$ and permissions $P$. Central to this model is the concept of role relations, around which a role is a semantic construct for formulating policies. In this model, permissions are associated with roles. Individual users acquire permissions if they are assigned to appropriate roles, which allow those permissions. These assignments are performed based on the responsibilities and qualifications of users in their organization. A role is a means for naming many-to-many relationships among individual users and permissions. A user in this model can be a human being, where it can be extended to include intelligent autonomous agents, machines and even networks. A role might be a job function or a job title, and permission is an approval for executing an object method (access to one or more objects, or privileges to carry out a particular task). The following is a list of formal notations of original RBAC components:

$$UR \subseteq U \times R$$

$PR \subseteq P \times R$ is a many-to-many permission to role assignment relation.

In this model, a user can be authorized to play several roles, where a role may be played by several users. We assume that there is a mechanism that associates users with roles. We also assume that a user is explicitly assigned to a given role and that assignment gives him or her the right to play the role. Whenever a user tries to perform an operation on a particular object, the authorization system checks both: whether the role associated with the user is allowed to perform the operation, and whether the user is authorized to play the role. This checking is performed before the authorization system checks any other existing access policies established by the system administrators.

RBAC introduces role hierarchies (RH), which is a natural means of structuring roles to reflect an organization’s lines of authority and responsibility. It basically recognizes two types of role hierarchies: general role hierarchies and limited role hierarchies. In the hierarchical RBAC a general role hierarchy is depicted with the notation $\geq$ and is defined as follows:

Definition 1: $RH \subseteq R \times R$ is a partial order on $R$, called the inheritance relation and denoted as $\geq$ which implies that $r_1 \geq r_2$ only if all permissions of $r_2$ are also permissions of $r_1$ and all users of $r_1$ are also users of $r_2$. A formal notation of this relation is as follows:

If $r_1 \geq r_2 \Rightarrow \text{Perms}(r_2) \subseteq \text{Perms}(r_1) \land \text{Users}(r_1) \subseteq \text{Users}(r_2)$ where the Perms and Users functions are defined as:

$$\text{Perms}(r) = \{p \in P : rt \geq r, (p, rt) \in PR\}$$

$$\text{Users}(r) = \{u \in U : rt \geq r, (u, rt) \in UR\}$$

In addition to the basic definitions of hierarchical RBAC we define the following notations:

Definition 2: $\forall u \in U, r$ is an Absolute Minimum Role ($\text{AMin}$) iff: $r \in R \land u \in \text{Users}(r) \land \exists rt \in R : rt < r \land u \in \text{Users}(rt)$

Definition 3: $\forall u \in U, r$ is an Absolute Maximum Role ($\text{AMax}$) iff: $r \in R \land u \in \text{Users}(r) \land \exists rt \in R : rt > r \land u \in \text{Users}(rt)$

Definition 4: $\forall u \in U, p \in P, r$ is a Relative Minimum Role ($\text{RMin}$) iff: $r \in R \land p \in \text{Perms}(r) \land u \in \text{Users}(r) \land \exists rt \in R : rt < r \land p \in \text{Perms}(rt) \land u \in \text{Users}(rt)$

Definition 5: $\forall u \in U, p \in P, r$ is a Relative Maximum Role ($\text{RMax}$) iff: $r \in R \land p \in \text{Perms}(r) \land u \in \text{Users}(r) \land \exists rt \in R : rt > r \land p \in \text{Perms}(rt) \land u \in \text{Users}(rt)$

According to above, it is straightforward to prove that:

$$\forall u \in U, p \in P, R_{\text{AMax}}(u) \geq R_{\text{RMax}}(u, p)$$

Let us assume that this relation is not true and then $\exists u \in U, p \in P$ such that $R_{\text{AMax}}(u) < R_{\text{RMax}}(u, p)$. If $\text{AMax}(u) = R_{\text{AMAX}}$ and $\text{RMax}(u, p) = R_{\text{RMAX}}$, then according to the definition of RBAC general role hierarchy we can conclude that: if
\[ R_{AMAX} < R_{RRMAX} \Rightarrow Users(R_{AMAX}) \not\subseteq Users(R_{RRMAX}) \]
or \[ Perms(R_{RRMAX}) \not\subseteq Perms(R_{AMAX}). \]
The first assumption cannot be true as: \( \exists u \in U : u \in Users(R_{AMAX}) \land u \in Users(R_{RRMAX}). \) This implies that \( Users(R_{AMAX}) \not\subseteq Users(R_{RRMAX}). \) The second assumption cannot be true either; as the \( AMAX(u) \) is the highest role in the general role hierarchy assigned to user \( u \), so it includes all the permissions that \( u \) can hold from the role hierarchy. As \( u \) holds \( R_{RRMAX} \), it therefore holds the permission \( p \), then \( \exists p \in P : p \in Perms(R_{RRMAX}) \land p \in Perms(R_{AMAX}) \) which implies that \( Perms(R_{RRMAX}) \not\subseteq Perms(R_{AMAX}). \)

Given a similar proof it is simple to prove the relation \( \forall u \in U, p \in P, R_{RMin(u,p)} \geq R_{AMax(u)} \).

**B. Workflow authorization model**

The workflow model used in this work is based on a workflow graph model presented in [8]. Any process model based on this graph modeling conforms to the following basic properties:

- it uses only core modeling structures namely “sequence”, “choice”, “merge”, “fork”, “synchronization”, “begin/end”, “nesting” and “iteration”.
- it does not have any cycles, and the iteration structure is supported only through blocked iteration.
- it has exactly a single begin and a single end task.

Each object (node and transition) of the workflow graph has some associated attributes. These attributes could be singular values or sets of other values or objects. These attributes are used to define modeling structures and associate correctness criteria in graph notations.

The workflow graph \( G = (N, S) \) is a simple directed acyclic graph (DAG) where:

- \( N \) is a finite set of nodes.
- \( S \) is a finite set of transitions representing directed edges between two nodes.
- \( |G| = |N| + |S| \) represents the total number of nodes and transitions in \( G \).

For each node \( n \in N \):

- \( \text{nodeType}[n] \in \{ \text{TASK,CM} \} \) represents type of \( n \), where CM stands for Choice Merge coordinator.
- \( \text{OutNodes}[n] = \{ m : m \in N \land \exists s \in S \text{ where fromNode}[s] = n \land toNode[s] = m \} \), i.e., a set of succeeding nodes that are adjacent to \( n \).
- \( \text{InNodes}[n] = \{ m : m \in N \land \exists s \in S \text{ where toNode}[s] = n \land fromNode[s] = m \} \), i.e., a set of preceding nodes that are adjacent to \( n \).
- \( \text{CHM} = \{ n : n \in N \land \text{nodeType}[n] = \text{“CM”} \} \), CHM is a finite set of nodes which their type is “CM”
- \( T = \{ t : t \in N \land \text{nodeType}[t] = \text{“TASK”} \} \). \( T \) is a finite set of nodes which their type is “TASK”

The graph \( G \) meets some syntactical correctness properties such as: Choice and merge coordinator nodes are not used in sequential structures, it does not contain cycles and it does not contain more than one begin node and one end node.

**Definition 6: Instance subgraph** represents a subset of workflow tasks that might be executed for a particular instance of a workflow. It can be generated by visiting its nodes on the semantic basis of underlying modeling structures. The subgraph representing the visited nodes and flows forms an instance subgraph. A workflow graph without choice structures would produce exactly the same instance subgraph as the workflow graph. A workflow graph with a single choice structure would produce as many possible instance subgraphs as the number of outgoing flows from the choice structure. A complete execution of a workflow is considered as executing all tasks belong to one of the workflow instance subgraphs. This includes the start node and the end node of the workflow.

In this model, we make the assumption that a workflow consists of several tasks to be executed sequentially. Each task is associated with a permission, which might be held by one or many roles. Then according to RBAC, multiple roles might be authorized to execute a task. We refer to the association of permissions with tasks in a workflow as Workflow Permission Specification (WPS). In order to associate permissions to the workflow we use a similar notation as the Authorization Template (AT) described in [11][12].

**Definition 7:** Given a task \( t_i \), an authorization template \( AT(t_i) \) is defined as a 2-tuple \( (t_i, p_i) \) where \( t_i \in T \) is a task and \( p_i \in P \) is the permission which grants the execution of \( t_i \).

**Definition 8:** Given a workflow \( WF \), a workflow permission specification (WPS) is defined as a list of authorization templates \( [AT_1, AT_2, ..., AT_n] \), where each \( AT_i \) is an authorization template.

According to above we define the function \( TaskPerm \) on the set \( PT \), which is a one-to-many permission to task assignment relation such that:

\[ PT \subseteq P \times T \]

\[ TaskPerm : T \rightarrow P \]

\[ TaskPerm(t) = p : (t, p) \in PT. \]

A workflow may have several workflow instances and we assume that each instance inherits the same workflow permission specification as the original workflow from which those instances can be generated. We also assume that there is a WFMS which contains the knowledge about the processes, the ordering of activities, dependencies and so on. We also assume that there is an authorization system integrated with the WFMS to permit or deny execution of tasks based on both the workflow permission specification and the role of user or entity who is willing to execute a task. It should also be noted that in our presented WPS model we are considering a simple authorization constraint (associated roles) for each task, however this can be further improved and be extended by considering more authorization constraints and policies.

**V. Bottom-up Delegation Algorithms**

Delegating rights through a bottom-up delegation model should be performed on-demand in different phases of a job execution. It can conceptually be divided into two phases: initial phase and run-time phase. The initial phase is considered
as the time that a delegatee starts off the job, and the run-time phase would be when a delegate is proceeding the execution.

Later in this paper when we elaborate our approach and we describe three different models of bottom-up delegation, we will explain each of these phases in more detail. However, before proceeding further, we need to define the notion of Delegation Safety Invariant as the following:

**Delegation Safety Invariant (DSI)** is an essential factor which assures that performing delegation in an on-demand manner ensures that:

- Privileges can only be delegated either at the time of launching tasks or during the execution (just-in-time).
- No privileges can be delegated from a superior to a sub-ordinate unless they are restricted to an intended delegated task (bound-to-task).
- No privileges can be delegated for accessing a resource unless they are less than or equally as powerful as the privileges held by the superior for accessing the same resource or executing the same task (limited-to-boundaries)

In the rest of this section we present three different models of bottom-up delegation when a delegate needs to complete a workflow on behalf of its delegator. For this all required rights are obtained via the appropriate number of call-backs to a delegation service which delegates rights (roles). Depending on the delegation model used, the required rights are obtained by making one callback at the initial phase or through several callbacks at the run-time phase. The main differences of these models are basically the number of callbacks to the delegation service and the level of restrictions for delegated rights. A suitable model can be chosen by administrators or superiors based on the Risk of Delegation factor which is defined as the following:

**Definition 9: Risk of Delegation (RoD)** is a parameter which can be set by security administrators or individual delegators to specify the level of security risks and threats when delegating rights. It gives the ability to adjust risk factors based on the current threat landscape and enforces the security model to operate accordingly. A higher value for this parameter indicates a higher level of security risks and misbehavior when delegating rights and a lower value implies less security risks and threats associated with delegation.

If the grid is under active attacks, it is important for many administrators to be able to modify security operations in reaction to the new conditions. The RoD parameter controls how restricted privileges can be delegated to potential delegates. It also determines how often during a workflow execution a delegate is required to ask (via making call-backs) for more rights. For example, if delegation is performed in safe and restricted environments to highly trusted delegates, the value of RoD can be set to a lower level which allows providing more rights via a fewer number of callbacks; and in case of stronger potential of security threats and risks, a lower value can be set to the RoD parameter to enforce using a delegation model which requires a larger number of callbacks and more restricted range of delegated rights.

How administrators or delegators evaluate risks or attacks and specify the value of RoD parameter is beyond the scope of this document. We believe that this should be completely up to the administrators or delegators to perform their risk analysis before setting this value.

Before we describe different models of bottom-up delegation, we give some preliminary definitions used in the rest of this document.

**Definition 10:** The Least Capable Task-Role (LCTR).

Given a user $u$ and a task $t$, the LCTR is the weakest role among all the roles assigned to user $u$, which permits the execution of $t$ by $u$.

The LCTR procedure defined below can be used to determine the least capable task-role:

**procedure LCTR:**

**Input:** $u \in U$ and $t \in T$ and WPS

**Output:** $r_{\text{minimal}}$ [The weakest capable role of user $u$ for performing task $t$]

**Start**

$r \leftarrow \emptyset$

$p \leftarrow \text{WPS.TaskPerm}(t)$

$r \leftarrow P\text{Roles}(p) \cap U\text{Roles}(u)$

$r_{\text{minimal}} \leftarrow \text{Minimal}(r)$

**End**

**Definition 11:** A Minimal function returns the weakest role among a set of roles in an RBAC hierarchical relationship.

A formal notation of this function is given in below:

$$\text{Minimal}(R) = r : r \in R$$

**Minimal(RSet) = r : r \in RSet \land r \in R \land \forall r \in RS, r \geq r$$

Below is the formal notation for the output of LCTR procedure:

$$\forall t \in T, R_{\text{LCTR}(u,t)} = r \text{ if } r \in R \land r \in U\text{Roles}(u) \land r \in P\text{Roles}(\text{WPS.TaskPerm}(t)) \land \forall r \in R : r \in \text{U\text{Roles}(u)} \land r \in \text{P\text{Roles}(\text{WPS.TaskPerm}(t))} \Rightarrow r \leq r.$$  

It should be noted that $\forall u \in U, t \in T$, if $p = \text{WPS.TaskPerm}(t)$ then $R_{\text{Min}(u,p)} \leq R_{\text{LCTR}(u,t)} \leq R_{\text{Max}(u,p)}$. We have already proved that $\forall u \in U, p \in P, R_{\text{Max}(u,p)} \leq R_{\text{Max}(u)}$ thus it is already proved that $R_{\text{Min}(u)} \leq R_{\text{LCTR}(u,t)} \leq R_{\text{Max}(u)}$.

As we mentioned earlier, satisfying the DSI factor requires that all delegated rights must be restricted to particular task(s). We gather these tasks in a set and we call it the set “PROCESS”. This set may include the entire tasks of a workflow or only a single task. This set has a a simple formal notation given below:

**Definition 12:** Given a workflow $WF$ a PROCESS set is a set of tasks:

$$\text{PROCESS} \subseteq T$$

$$\text{PROCESS} = \{t_i : t_i \in T\}$$

**Definition 13:** Most Powerful Delegatable Roles (MPDR) is a set of roles, determined to be delegated in each call-back phase of a bottom-up delegation, that allows a delegate to proceed the execution of particular tasks from the workflow. These particular tasks are the members of the PROCESS set associated to each MPDR described earlier.
The basic concept of our delegation model is to put the requirement of obtaining rights on-demand via making callbacks to a designated service. We implement this concept in three different models, which basically differ when determining how often a delegate needs to request for additional rights and what set of rights should be delegated in each callback. In general, a model which is more restricted does not allow a delegate to proceed too far and requires making more requests to the delegation service for additional rights; whereas a less restricted model allows a delegate to obtain more rights via each callback and therefore to be able to proceed on the execution path for a larger number of tasks.

The RoD parameter is used to choose the appropriate bottom-up delegation model. One can presume three different values for the RoD parameter: “Low”, “Medium”, and “High”. The “Low” value of RoD implies using a One-Step delegation model, the “Medium” value implies delegating rights in a Multi-Steps delegation and finally the “High” value, which indicates the highest level of risks, implies a Per-Step delegation model, which implements the most restricted delegation model. In all three delegation models, that we describe below, we consider two sets of output: i) a set of role(s) to be delegated and ii) a set of tasks to which delegated role(s) are restricted. Regardless of which delegation model is used by the security system (enforced by the RoD parameter), we need to make sure that these sets are determined to meet the requirement of the DSI factor.

1) Per-Step delegation: This model is used when the RoD parameter is set to the highest value and indicates the highest level of security threats (i.e. active attacks) when delegating rights. This model is the most restricted because it requires a delegate to make callbacks in any step of WF execution. The delegation service in response delegates only one role, which is the LCTR for the next task of the execution path. The pseudo code of this model is depicted below:

Procedure MPDR-PS:
Input: \( u \in U, t \in T \) and workflow \( wf \)
Output: \( r_{MPDR-PS} \in R, p_{MPDR-PS} \in T \)

\( r_{MPDR-PS} \): The MPDR set of roles to be delegated
\( p_{MPDR-PS} \): The tasks to which the MPDR is bound to

Start
\( r_{MPDR-PS} \leftarrow \{ \} \)
\( p_{MPDR-PS} \leftarrow \{ \} \)
\( r_{MPDR-PS} \leftarrow LCTR(u,t) \)

End

The output of this procedure will be \( r_{MPDR-PS(u,t,wf)} = \{ r \mid r \in R, r = LCTR(u,t) \} \) and \( p_{MPDR-PS(u,t,wf)} = \{ t \} \).

It is important to note that in this model:
\[ \forall u \in U, t \in T, R_{LCTR(u,t)} \leq R_{Max}(u,p) \leq R_{Max}(u) \]
and therefore \( R_{MPDR-PS(u,t,wf)} \leq R_{Max}(u,p) \leq R_{Max}(u) \) where \( p = wf.TskPerm(t) \).

This model enables an extensive level of logging and monitoring at the back-end of the authorization system. The more callbacks that per-step delegation requires, the more fine-grained logging and monitoring is possible at the delegation service, which can be used for intrusion detection.

2) Multi-Steps delegation: This model presents an efficient yet restricted way of delegating rights (roles) for executing a workflow on behalf a delegator. The idea of a multi-step delegation is to implement a restricted model, with a fewer number of callbacks to a delegation service. In this model, when a delegate makes a callback to the delegation service, the required privileges (roles) are determined and delegated in a way to enable the delegate to complete the next task and continue the execution until it meets a task which requires additional rights (either a stronger role or a different role).

For this, the algorithm first determines the MPDR role of the next task, and then continues traversing the workflow to determine a set of consequential tasks from the workflow that can still be executed using the same role.

Procedure MPDR-MS:
Input : \( u \in U, t \in T \) and workflow \( wf \)
Output: \( r_{MPDR-MS} \in R, p_{MPDR-MS} \in T \)

\( r_{MPDR-MS} \): The MPDR set of roles to be delegated
\( p_{MPDR-MS} \): The tasks to which the MPDR is bound to

Start
\( q \leftarrow \text{null} \)
\( \text{weakest} \leftarrow \text{TRUE} \)
\( r_{MPDR-MS} \leftarrow \{ \} \)
\( p_{MPDR-MS} \leftarrow \{ \} \)
\( q.initiate() \)
\( q.push(t) \)
while \( q.empty() \) AND \( \text{weakest} \) do
\( r_{MPDR-MS} \leftarrow LCTR(u,t) \)
\( p_{MPDR-MS} \leftarrow p_{MPDR-MS} \cup \{ t \} \)
\( q.push(wf.outNodes[n]) \)
\( m \leftarrow q.pop \)
if \( (\text{type}[m] == \text{"TASK"}) \) then
\( \text{if} \ (LCTR(u,m) \leq LCTR(u,wf)) \) then
\( p_{MPDR-MS} \leftarrow p_{MPDR-MS} \cup \{ m \} \)
else
\( \text{weakest} \leftarrow \text{FALSE} \)
\( q.push(wf.outNodes[m]) \)
end

End

This model determines the LCTR of the next task (\( r_{MPDR-MS} \)) and continues to find the successor tasks (\( p_{MPDR-MS} \)) which are executable using the same role. By this, with delegating \( r_{MPDR-MS} \), the execution of tasks can continue as long as the delegated role is sufficient and holds all required privileges. In this model, in each callback, delegation is done by delegating only one role which is restricted to a limited number of tasks of the workflow. The outputs of this procedure in formal notation are the following:

\( r_{MPDR-MS(u,t,wf)} = r \) such that \( r \in R \land r = LCTR(u,t) \land \forall t_i \in p_{MPDR-MS} : r \geq R_{LCTR(u,t_i)} \).

\( p_{MPDR-MS(u,t,wf)} = \{ t \} \cup \{ t_i : t_i \in T \land t_i \in wf.successors[t] \land R_{LCTR(u,t_i)} \leq LCTR(u,t) \} \).

In this model it is also important to note that: \( \forall u \in \)}
where $p_i = \text{wf.TaskPerm}(t_i)$ and $R_{\text{MPDR-MS}(u, t, \text{wf})} \leq R_{\text{RMax}(u, p)} \leq R_{\text{AMax}(u)}$.

In this model, there might exist a task $t_k \in \text{pMPDR-MS}$ for which $R_{\text{LCTR}(u, t_k)} \leq R_{\text{MPDR-MS}(u, t, \text{wf})}$ however as it is already proved that $R_{\text{MPDR-OS}(u, t, \text{wf})} \leq R_{\text{RMax}(u, p)} \leq R_{\text{AMax}(u)}$ and it is always restricted to a set of specific tasks, delegation can still meet the requirement of the DSI factor.

### VI. Discussion

Although in this paper we are proposing to leverage workflows for satisfying the DSI factor in bottom-up delegation, it is worth also discussing the benefits of leveraging a bottom-up delegation in workflows in general and stating how it is even crucial for scientific workflows to support such a bottom-up delegation mechanism.

Scientific workflows have some specific and unique requirements, which may impose some special considerations when providing security. They are basically used to automate scientists’ conventional work patterns, when they are conducting their experiments by applying their specific methodology over distributed resources. This requires scientific workflows to operate over a changing experimental context and highly dynamic and distributed infrastructure. Tasks are usually submitted as a “batch” process and are executed wherever and whenever resources are available. Task execution time and the number of participants are very diverse. All these considerations require workflows of scientific applications to be very dynamic, flexible, robust, and dependable with support for incremental workflow construction and to be able to run detached from the user console, because in many cases the end user is not part of the process. Scientific workflows are designed to require minimum level of human approval or intervention during the life-cycle of a process except at some points during the workflow enactment process.

For scientific workflows there are many uncertain factors such as unavailability, incomplete information and local policy changes, which may affect the way that execution of a workflow system is being proceed [7][6]. Scientific applications are usually executed in a collaboration with participants from different organizations, and therefore there is no central ownership and control over resources. The resources are not under the exclusive and complete control of the owner of the workflow systems. The local policies are subject to change without informing the users of workflow systems.

Thus a full-ahead-plan in scientific workflows is very hard to achieve, since it requires the specification of the exact location of resources assigned to every task in advance before the execution of workflows and the assumption of no policy changes at run-time. These are not usually the right assumptions for scientific workflows. In scientific workflows jobs are submitted in “batch” form and users may not participate in execution to admin and adapt the workflow specification to meet the new requirements of workflow execution. Users usually leave after starting off their jobs; they might monitor the execution, but often they just expect the result at the end.

### VII. Related Work

The Workflow Authorization Model (WAM) is aimed at addressing the security issues in workflow management sys-
tems. It was originally presented in [13] and extended in [14]. WAM proposes an authorization model for granting and revoking authorizations, considering some principles of workflow security such as enforcement of separation of duties, handling of temporal constraints and support for role-based application support. Based on the WAM authorization model, the work presented in [15] proposes a Delegation Authorization Model (DAM) to facilitate the delegation of task activities between subjects in an organizational hierarchy. Using this model, subjects in organizations are allowed to delegate task responsibilities to other subjects according to restrictions imposed by security policies.

The work presented in [16][17] provides an authorization model to handle delegation in an RBAC workflow environment. Their model extends the RBAC model to support a user to user delegation, user to a group of users delegation and revocations. Their workflow delegation model addresses dynamic constraint handling by defining specific delegations. Multiple delegation and partial delegation through the chain of delegation is also supported by this model.

The work presented in [18] incorporates delegation in a workflow management system. There is a delegation module which is invoked whenever a delegation decision for task assignment must be made. There is a delegation table which only displays static constraints and relationships of delegations; when a workflow assigns a task, this assignment is screened by the delegation module to determine whether delegation is required.

None of the above approaches address all the requirements of delegation described earlier. All these approaches perform delegation in a full-ahead-plan tied to a particular user or group. They are mainly intended to business workflows in static, predictable and controlled environments.

VIII. CONCLUSION

In this paper we presented the concept of bottom-up delegation for providing restricted delegation in dynamic collaborative environments. We defined the notion of Delegation Safety Invariant (DSI) as an important security factor to meet the requirement of the least privileges principle when performing delegation on-demand. To meet the requirements of the DSI factor we further proposed to integrate workflows in an on-demand delegation framework. We presented three different models of bottom-up delegation aimed at providing delegation with different levels of efficiency and restriction based on the current threat landscape and the Risk of Delegation (RoD) parameter. Using a standard RBAC authorization model and a graph-based workflow model (DAG) we defined and analyzed a formal model of our proposed bottom-up delegation approach. We used these formal definition to evaluate that each proposed model meet the requirements of DSI factor.

As future work we are investigating the integration of our on-demand delegation framework with real grid workflow systems. The work presented in this paper is also based on a standard RBAC authorization system and therefore delegation will be implemented by means of delegating roles to subordinates. Thus, another area to be investigated would be other delegation mechanisms which are not fully based on the standard RBAC authorization systems and perform delegations by other means. In that sense determining the minimum set of rights to be delegated in each callback would be a challenging issue.

Acknowledgment: We would like to thank David Chadwick from the University of Kent for his valuable feedback and comments on this work.

REFERENCES

Paper IV : Grid Delegation Protocol

Grid Delegation Protocol

Mehran Ahsant\textsuperscript{a}, Jim Basney\textsuperscript{b} and Olle Mulmo\textsuperscript{a}

\textsuperscript{a}Center for Parallel Computers, Royal Institute of Technology, Stockholm
\{ mehrana, mumlo@pdc.kth.se \}

\textsuperscript{b}National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign, USA
\{ jbasney@ncsa.uiuc.edu \}

Abstract

We propose a delegation protocol based on the WS-Trust specification, which is applicable for a wide range of Grid applications. The protocol is independent of underlying security mechanisms and is therefore applicable to all security mechanisms of common use in Grid environments, such as X.509 proxy certificates, Kerberos based delegation, and SAML assertions. We emphasize that this is work in progress. In this paper, we document our thoughts and current strategy, and we solicit comments and feedback on our approach.

1 Introduction

Delegation is a common requirement for a wide range of Grid applications. Delegation is also supported by various existing security mechanisms in different protocols (Welch, Foster, Kesselman, Mulmo, Pearlman, Tuecke, Gawor, Meder and Siebenlist, 2004). Considering the fact that current in-place security mechanisms will continue to be used, specifying a standard delegation method that can be used uniformly and independently of underlying security mechanisms is an essential effort. By describing delegation as a standalone Web Services portType, and by providing ready-to-use library implementations of this portType, service implementers do not need to deal with the details of the delegation mechanisms, and can factor this functionality out of the internal application logic. Furthermore, delegation can be made part of the functionality provided by the container that hosts the service. In this paper, we describe a delegation protocol, which can be leveraged by diverse Grid applications and environment scenarios. The protocol is compliant with the WS-Trust specification, a technology that can be used to express the specifications required to define a delegation protocol (WS-Trust, 2004). We emphasize that this is work in progress, and with this document we hope to solicit comments from the Grid security community on our proposed approach.

2 Terminology

In this section, we provide the basic definitions and terminologies used by this document for a Grid Delegation Protocol as follows:

- **Delegation** is the act of transferring rights and privileges to another party (the Delegatee).
- **Delegatee** is the delegation target. It is the entity that the Delegation Credential is delegated to.
Delegator is the entity that delegates the abilities and/or rights to the Delegatee.

Delegation Credential is the desired result of delegation protocol. It is a message conveying the abilities and rights from the Delegator to the Delegatee. While the actual syntax and contents of Delegation Credentials are out of scope for this paper, we note that they are typically integrity protected and digitally signed.

3 Usage Scenarios

A wide range of Grid applications make use of delegation. In this section, we describe some usage scenarios in Grid environments that all have delegation mechanisms as a common requirement.

3.1 Delegation of privileges

When a Grid user makes use of a remote resource to execute a job, that resource may in turn need to access third-party resources (e.g. data repositories) on behalf of the user in order to complete the task. Such access may possibly span across multiple security domains. In this scenario, delegation can be used to delegate (parts of) the user’s rights to the remote resource such that it in turn can access the necessary third parties.

3.2 Renewal of delegation credentials

Credentials in general have a validity or lifetime. In case of delegated Grid credentials, their lifetimes are typically short (less than a day) and thus need to be renewed periodically for long-running tasks such as complex scientific applications or ongoing experiments. The need for delegation renewal is essential for Grids (Welch, Siebenlist, Foster, Bresnahan, Czajkowski, Gawor, Kesselman, Meder, Pearlman and Tuecke, 2003). The responsibility of initiating a renewal process may be put either on the Delegatee, or on some external party such as e.g. the Delegator or the originating user (human) that submitted a long-running job. In the case of external parties, they need a mechanism to trigger the Delegatee to request a new delegation credential.

3.3 Online CA

Online certification authorities (CA) tied to the Kerberos (KCA/Kx509) and PKI infrastructure (CAACL) can simplify the process of certificate management. Online CAs provide a simple way of PKI enrollment and credential management. Users authenticated to the online CA can request credentials on demand or in advance. Online CAs, which are managed and operated under tight and restricted security policies, can issue long-term or short-term certificates in response to the user’s request. Issuing long-term certificates mandates users to manage their credentials in a traditional way of credential management. On the other hand upon on-demand credential request, online CA issues a short-term credential or it might even delegate a short time credential to the user (Basney, Yurcik, Bonilla and Slagell, 2003).

3.4 Online credential repository

Online credential repositories, such as MyProxy, provide secure storage and management services for long-term credentials. Online credential repositories recognize the fact that users are mobile and typically do not properly protect any kind of software-based credentials with a long validity. Instead, users request short-term credentials derived (delegated) from the long-term credentials,
maintained by the repository. Besides providing direct access to the end users to retrieve time-
limited credentials from the online credential repository in a restricted-access manner, the online
credential repository can also act in an “indirect” mode and delegate credentials to a third party
on behalf of the user. Online credential repositories also play a significant role in delegating
credentials to web portals when the native protocols, such as HTTPS, do not support credential
delegation. Moreover, online credential repositories can be leveraged by Grid services to renew
the credentials issued for long lasting tasks before expiration (Novotny, Tuecke and Welch, 2001).

4 WS-Trust

WS-Trust is a specification defined by IBM, Microsoft, RSA and VeriSign that uses the secure
messaging mechanisms of WS-Security and aimed to define additional primitives and extensions
for security token exchange (WS-Trust, 2004). WS-Trust does not describe or mandate using
explicit fixed security protocols: instead, it provides a common SOAP messaging structure that
can be used by a flexible set of security mechanisms. The specification provides a simple re-
quest/response protocol for issuing, exchanging and validating security tokens and establishing
trust relationships between different security realms.

5 GrDP and WS-Trust

We propose a Grid Delegation Protocol, GrDP, building on the WS-Trust specification. The pro-
tocol is explicitly and intentionally aimed to be compliant with the WS-Trust specification. The
protocol proposed by this document might be implemented for different WS-Trust delegation pro-
files such as GSI proxy delegation, Kerberos delegation or SAML. Each profile would describe
how to use the WS-Trust specification for a particular GrDP implementation. Our current imple-
mentation of GrDP, profiling the existing GSI proxy delegation in terms of WS-Trust specification,
has essentially the same steps as the current GSI proxy delegation protocol (Welch et al., 2004).

5.1 GrDP requirements

Below, we identify and describe some general requirements of GrDP, and how WS-Trust can be
used to meet those requirements.

Transparency The protocol should reduce the burden of understanding the format of security
tokens and it should also ease the management of trust relationships with external entities.
Thus developer and administrators should not be anymore worried about security formats
and mechanisms which are used and known by target domains (WS-Trust, 2004).

Generality GrDP should be used by a wide range of Grid applications in diverse scenarios. Thus
it should be general in design and be able to support the most possible scenarios in the
Grid environment. WS-Trust provides different operations such as issuance, exchange and
validation with the same protocol design and different protocol implementations that helps
the GrDP cover various scenarios of delegation in Grid environment (WS-Trust, 2004).

Interoperability GrDP should be interoperable between different security realms which support
different kinds of security tokens. It should be able to translate or exchange security to-
kens into different formats. It should also be able to map the identities and interpret the
name-space and attributes of security tokens into the proper format. The protocol should
address the issue of trust interoperability as well. Having a security token which is compre-
hendible by its syntax in other security realms does not mean that other parties can trust the
issuer of the security token. GrDP relies on WS-Trust which enables clients and services to interoperate without sharing a common security token format or establishing a direct trust relationship (Madsen, 2003).

**Modularity** Apparently no protocol can provide a complete security solution for Grid services. Thus GrDP as other Grid-based protocols should have the property of being a building block for the whole infrastructure. WS-Trust has the capability of being used in conjunction with other Web service specifications and a variety of security models and protocols. Thus a protocol based on WS-Trust would be modular (WS-Trust, 2004).

**Flexibility** GrDP should be able to use different types and models of security tokens. The exchanged credentials by GrDP might be any kind of security tokens. This protocol uses the element “tokenType” specified by WS-Security to pass security tokens inside SOAP messages. The WS-Trust specification allows for describing the type and encoding model of security tokens as well (WS-Trust, 2004).

6 Grid Delegation Protocol

GrDP provides for delegation of rights and abilities in terms of Delegation Credentials between a Delegatee and a Delegator. In this section, we give a general description of this security mechanism agnostic protocol for performing delegation in Grid environments.

6.1 Protocol overview

GrDP is based on a WS-Trust Request/Response message pair from a Delegatee to a Delegator. The Delegatee issues a delegation request and sends that to the Delegator, who verifies the validity of the request, performs the requested operation denoted by the request, and sends the response containing the delegation credential back to the Delegatee (see Figure 1). The UML diagrams below depict different scenarios more clearly: in particular, an example where the initiating party is different from the Delegator will be shown.

We note that this protocol solves a “pull” model (see Figure 3) for obtaining a delegation credential. However, in some use cases, a “push” model is more favorable, for example in case of a renewal of a previously delegated credential. In those cases, the initiating party (which may be different from the Delegator) makes use of an Initiate message, to trigger the Delegatee to make a delegation request to the Delegator. Finally, the Delegatee may report back to the initiating party on the success status of the delegation operation, as an InitiateResponse message (see Figure 2).

In order to keep the number of roundtrips to a minimum, and in order to provide for the (common) use case where the Delegator already has enough information to do a delegation without the Delegatee needing to get involved, the Initiate message may contain an optional Response message payload with the new Delegation credential.

6.2 Protocol operations

GrDP supports several operations that can be requested by the Delegatee to be performed by the Delegator. We divide the operations of the Delegation Protocol into two main categories: Core delegation operations and Extended delegation operations.

6.2.1 Core operations

Three main operations are supported by the delegation protocol: to issue, renew and exchange delegation credentials.
**Delegator**

**Delegatee**

**InitDelegation()**

**WS-Trust ReqIssue()**

**WS-Trust ReqIssueResponse()**

**InitDelegationResponse()**

**DelegatedToken**

Figure 1: GrDP Request/Response Messaging

**Delegator**

**Delegatee**

**InitDelegation()**

**WS-Trust ReqIssueResponse**

**InitDelegationResponse()**

**DelegatedToken**

Figure 2: GrDP Push Model

**Issue** Creates a new credential according to the needs of the requestor, if the requestor is so authorized. The result is a Delegation Credential that, in a limited and restricted manner, delegates the rights and abilities from the Delegator to the Delegatee.

**Renew** Renews a Delegation Credential. Renewing only can be performed for those delegation credentials that are marked as “Renewable” for a requested period of time (as discussed later in this paper.)

**Exchange** The Initiator asks to exchange a Delegation Credential in a different format or even for a different trusted domain. As an example, the Initiator asks to exchange a proxy certificate into a Kerberos ticket. For this operation to perform there might be a need for other
6.2.2 Extended operations

The protocol’s extended operations support storing and revoking stored credentials and updating access policies attached to each credential stored in the repository.

**Store credential** Storing a long-term credential in a safe repository to be used to create short-lived, restricted credentials. As an example, this operation stores a long-term X.509 certificate into the repository to be used by the Delegator to issue short-lived, restricted proxy certificates.

**Update access policies** For each stored credential in the repository there might be several associated attributes and policies which govern how to access and use the credentials. For example, access policies can specify for which domains and scope delegation is allowed to be performed or the valid formats that credentials can be translated into. The update operation is used to specify and update those attributes.

**Revoke credential** The Delegator asks to revoke a credential. If the credentials are compromised, the credentials must be revoked. In such case, by revocation, the Delegation Service can be prevented to perform delegation on behalf of a Delegator.

7 GrDP messaging structure

The messaging model of GrDP, outlined in Protocol Overview, consists of three messages: Initiate, Request and Response. In the following we describe each message, including a sample XML schema which depicts the attributes and elements used to construct the message.

**Initiate Message:** This message may be used by the Delegator to initiate the delegation process by prompting the Delegatee to send its delegation request. It may also contain a Response...
message payload, in case no information is needed from the Delegatee before a delegation credential can be created. In the following we illustrate a sample XML Initiate message (see Figure 4) for initiating a GSI delegation process. This message is sent by the Delegator to prompt the Delegatee to request for a new X.509 proxy certificate. This message implies that the Delegatee should provide the Delegator with a PKC#10 certificate request:

```xml
</S:Header>
<S:Body wsu:Id="init">
  <RequestSecurityToken>
    <TokenType>grdp:PKCS#10</TokenType>
    <RequestType>wsse:ReqIssue</RequestType>
    ...
  </RequestSecurityToken>
</S:Body>
</S:Envelope>
```

Figure 4: Sample “Initiate” XML schema

**Request message:** This message is issued by the Delegatee to ask the Delegator for a Delegation credential. The request message specifies both the type of operation, as described in Protocol Operation, and delegation attributes as are specified below:

- **Lifetime:** specifies the lifetime of the delegated credential by specifying the “Create time” and “Expiry time”. A “Create time” dated in the future can be interpreted as a request for a post-dated credential.
- **DelegateTo:** asks the Delegatee for a proxy certificate delegated to a third party.
- **Proxiable:** specifies if the credential delegation can be used to prove a claim for another delegation. In other words, it specifies if the proxy certificate can be used to sign another issued proxy certificate.
- **Delegatable:** specifies that an issued credential delegation can be delegated further. The credential delegation bearer might use a delegatable credential to further delegate obtained rights to another entity.
- **Renewable:** describes if the requested delegation credential would be renewable. Thus it identifies if the issued delegation credential is acceptable for renewing upon exceeding the limit of validity.

The XML schema depicted in Figure 5 shows the Delegatee’s request for issuing a proxy certificate based on a provided PKC#10 proxy certificate request. Figure 6 shows a request for Exchanging (renewing) a proxy certificate for a new time interval.

**Response Message:** This message containing a delegation credential, which is sent from the Delegator to the Delegatee in response to a delegation request. Figure 7 shows a sample XML schema for Response message.

8 Security Considerations

All GrDP messages must be properly authenticated and integrity protected using standard techniques and protocols. Encryption may also be necessary, depending on the security mechanism used.
Figure 5: Sample “Issue” XML schema

Figure 6: Sample “Exchange” XML schema
Figure 7: Sample “Response” XML schema

9 Current Status and Future Work

This document is a work in progress. We have already defined a WS-Trust profile for X.509 proxy certificate delegation, which binds the GrDP to the WS-Trust specification. The implementation of WS-Trust-based GrDP for GSI proxy delegation is currently underway. Specifying other profiles, such as a Kerberos to GSI translation service (e.g. KCA) and a password to GSI service (e.g. MyProxy) are work for future.

10 Related Work

In this section we mention some other projects which are defined around federation-related activities. TrustBridge, a new technology supported by Microsoft, enables different organizations running Windows operating systems to share their user identity information (TrustBridge, 2002). By leveraging Industry standard XML Web Services, TrustBridge allows participants to exchange user identities and interoperate in a heterogeneous environment. The Liberty Alliance project, by creating the Liberty specification, describes the way of federating network identities in order to enable single sign-on and attribute sharing (Liberty, 2003).

11 Conclusion

We note the significant importance of delegation in a wide range of Grid applications and use cases. The lack of standard delegation mechanisms necessitates the design of a standard and interoperable protocol for delegation mechanism in Grids. We proposed a Grid Delegation Protocol (GrDP), aimed at solving the delegation requirements in Grid environments. The protocol is based on the WS-Trust specification, which operates across security and organizational realms as it is independent of the actual security mechanism used. We re-emphasize that this is work in progress, and that one of our aims with this paper is to solicit comments and feedback from the community.

Acknowledgment: We would like to thank Von Welch for his valuable feedback and comments on this work.
References


https://www.projectliberty.org/index.html


TrustBridge (2002). Trustbridge project, (Online).


Paper V: Security Credential Mapping in Grids

Security Credential Mapping in Grids

Mehran Ahsant #1, Esteban Talavera González #2, Jim Basney ∗3

#Center for Parallel Computers, Royal Institute of Technology
Valhallavagn 79, 10044 Stockholm, Sweden
1mehran@pdc.kth.se
2esteban@kth.se

∗National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign
1205 W. Clark St., Urbana, IL 61801 USA
3jbasney@ncsa.uiuc.edu

Abstract—Federating security and trust is one of the most significant architectural requirements in grids. In this regard, one challenging issue is the cross-organizational authentication and identification. Organizations participated in Virtual Organizations (VOs) may use different security infrastructures that implement different authentication and identification protocols. Thus, arises an architectural need to provide a mechanism for a lightweight, rapid and interoperable translation of security credentials from an original format to a format understandable by recipients. In this paper, we describe the development and the implementation of an architecture for credential mapping in grids using off-the-shelf technologies and standard specifications. Our open-source implementation of this architecture provides support for an on-the-fly exchange for different types of security credentials used by diverse grid security infrastructures.

I. INTRODUCTION

Dynamic trust establishment and interoperability across multiple and heterogeneous organizations, participated in grids, introduce nontrivial security architectural requirements. There are some fundamental incompatibilities and varieties in underlying security infrastructures used by Virtual Organizations (VOs) in grids [1]. The requests for accessing grid services may cross organizations in which different security solutions are used for identification and authentication. For example, it might happen that a recipient is expected to receive a Kerberos ticket when it can only understand PKI-based certificates. Obviously, to cope with this problem neither replacing in-place security infrastructures nor requiring users/services to collect a set of security credentials in different formats and/or issued by different trust anchors are practical, cost-beneficial and possible solutions.

Credential mapping has been identified as an architectural requirement for grid security infrastructures to overcome the heterogeneity of existing security mechanisms. Such mechanisms could make it possible to establish dynamic and fast trust relationships in grids by enabling organizations to collaborate with verifiable and understandable security credentials regardless of their local established security mechanisms [1].

In this paper, we describe the development of a credential mapping mechanism for an on-the-fly exchange of different types of security credentials. To our knowledge, the work presented in this paper is the only full-mesh and standard credential mapping mechanism that has been developed as a lightweight, easy-to-integrate and open-source service for grids. We emphasize that our work presented in this paper deals only with “authentication tokens” mapping and the syntax aspects of security credentials. Another important aspects of “cross-organizational” interoperability, which are authorization and attribute mapping, have been fully covered in our another work described in [2].

Our paper proceeds as follows. In section II, we briefly review relevant standards. In section III, we describe in detail all supported credential mapping scenarios. The real grid scenarios, by those we have evaluated our solution are described in section IV. In section V, we give the related works and finally we conclude our work in section VI.

II. REVIEW OF STANDARDS AND SPECIFICATIONS

To develop a standard and secure solution we reviewed the WSS-* standards and specifications, which are currently the foundations of any security solutions for both Web services and grid services applications. The most relevant standard specifications to our work are the followings:

WS-Trust [3] 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 protocol to obtain security tokens from Security Token Services. WS-Trust is highly relevant to the sample question of how to obtain an X.509 certificate for accessing a web service given a username/password.

WS-Federation [4] 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 1. In this scenario, a client from domain A authenticates itself to its own organization’s Identity Provider (a type of security token service). In order 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.

The WS-Federation and WS-Trust specifications are general and do not restrict or suggest to using any particular approach.
when establishing trust relationships or performing credential translations. For example, they do not specify how Security Token Services (STSs), clients or services should establish their trust relationships for authentication or verification purposes. These specifications put only some general requirements e.g. to make sure that mappings performed by security token services are consistent with the access policy of the target web services. That is, the mapping should be in accordance to access control policies used by target web services that specify which tokens are required by a user in order to gain access to service (e.g. WS-SecurityPolicy).

Thus, from an architectural perspective, for our credential mapping solution it was important to figure out how to establish trust relationships between target services, clients and STSs in a practical and manageable way. The obvious and practical solution was to co-locating two of the services (depicted in Figure 1) so that one trust relationship operates within a single trust domain. For this, co-locating the STS and the target web service could be one approach. This could make it easy to define the meaning of tokens, and to digitally verify their 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, and create a new digital authentication path between the new user domain and the STS. Another model, which is approached in our solution, is more practical and places the STS in the same domain as the client that implies co-locating the STS and the user authentication service. This approach required the mapping mechanism to handle user tokens only in a single administrative domain and could make it much easier to establish authentication relationships. It had also many benefits when we needed to integrate our credential mapping framework with a dynamic authorization and attribute mapping mechanism in order to achieve a dynamic trust federation in grids as we elaborate in [2].

III. CREDENTIAL MAPPING SERVICE

A Credential Mapping Service (CMS) is a standalone Web Service, kinds of WS-Trust STS, for issuing 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. For example, if a user holds a Kerberos ticket asserting his identity, but target service need X.509 certificates for authentication, the Kerberos ticket can be presented to the CMS that will issue the holder an X.509 certificate asserting the same identity in different format, e.g. a Distinguished Name (DN). The CMS developed in our work has been focused to give a full translation functionality between different types of security credentials used by existing grid security infrastructure such as X.509 certificates [5], SAML assertions [6], Kerberos tickets, X.509 proxy certificates [7] and UsernameTokens [8]. In the followings of this section we first start by drawing a generic credential mapping architecture and then elaborate that separately for those mapping scenarios, which are mostly required by real grid applications. In the followings we describe the steps of a general mapping architecture as depicted in Figure 2.

The User authenticates with her local IdP (if she already doesn’t hold a valid credential), and obtains an authenticator token (Step 1). The requester makes a request message and attaches all proof-of-possession “Claims” required by the CMS for verifying the request (Step 2). The message then will be encrypted and sent to the CMS. How the request is encrypted and verified depend on each particular mapping scenario and the way that the trust relationship between the local IdP and the CMS is established (Step 3). We elaborate on this later when we describe each particular scenario separately. The CMS receives the message, decrypts it and verifies its integrity. The CMS checks the request against existing policies and verifies if the request provides all the “Claims” required. It then uses an appropriate mapping mechanism (in form of a plugin) to generate requested credentials (Step 4). If the CMS is already configured to use a third-party attribute provider, the CMS may then need to request for a set of attributes to be incorporated in a newly generated security credential (Step 5). The generated credential then will be packed, encrypted and sent back to the requester (Step 6). The requester receives the response message, decrypts it, verifies its integrity, extracts the generated credential from the message and stores that in a configured credential storage (Step 7).
1) Kerberos → PKI/SAML: This mapping particularly considers a use case in which a Kerberos-based requester needs a X.509 (proxy) certificate or a SAML assertion to communicate with a grid service. For this token mapping to take part the requester needs to identify a credential mapping service that it is trusted for converting his security token (e.g., the one which is closely tied to his home organization’s Key Distribution Center).

For this mapping the requester must be able to get a valid Kerberos token from his local Key Distribution Center (KDC), through a normal Kerberos authentication mechanism. It should be noted that even obtaining a Kerberos ticket can be implemented using the WS-Trust protocol. This would be completely compliant with the WS-Trust specification as the Kerberos KDC conceptually implements what the WS-Trust specification calls a STS, as it generates security tokens (e.g., Kerberos ticket) in exchange for other tokens (e.g., a user-name and password). We found that using the traditional Kerberos mechanism for obtaining Kerberos tickets is a more practical and reliable approach than using the WS-Trust protocol and it substantially could reduce the complications of our implementation. Although approaching a WS-Trust-based mechanism could instead help to overcoming some known Kerberos protocol limitations like traversing firewalls [9], however, our design architecture requires co-locating the CMS and the local IdP in one single administrative domain and therefore traversing firewalls could not be a problem. Before describing different mapping scenarios we should note that all the SOAP messaging described in the rest of this paper are fully compliant with the “issuance binding” of the WS-Trust specification. The steps of mapping a Kerberos ticket to an X.509 certificate or a SAML assertion are depicted in Figure 3 and described below (Note that steps 1, 2, and 3 are standard Kerberos message exchanges, and they have to be performed only once until the expiration of existing ticket).

1. The user, through the “Kerberos login” obtains a Ticket Granting Ticket (TGT).
2. The TGT is stored in the local cache of user’s home directory. This should be done using the traditional Kerberos mechanisms.
3. The client fetches the TGT from the cache and sends that to the Ticket Granting Service (TGS) part of the Kerberos KDC to obtain a Service Token (ST) for any further communications and authentications with the CMS. The ST is used as a proof-of-possession token (a claim) and allows the client to authenticate to the CMS (in fact on behalf of the user). The ST also contains the session key encrypted for the client. Now using the shared secret in the ST, all the messages between the requester and the CMS can be signed and be encrypted.
4. The client uses an appropriate identity mapping module and converts the Kerberos principal identity to an appropriate DN or a SAML subject. The client also generates an RSA public/private key-pair and depending on what type of security token is requested the client should proceed appropriately. For the Kerberos → X.509 mapping, the client uses the public key of the generated key-pair and the DN to create a PKCS#10 certificate request (CR) [10]. This CR will be attached as the “claims” part of the request message. For the Kerberos → SAML assertion, the client sends the generated public key to the CMS. This key later will be included in the “SubjectConfirmation” element when constructing the SAML assertion.
5. The client then attaches the ST along with either the CR or the public Key as the claims of request, encrypts the message and provides a signature on it using the session key obtained from the TGS and sends the request to the CMS.
6. The CMS uses the ST session key to verifying the integrity of the incoming request and decrypting the message (as any Kerberized service does). The CMS validates the claims provided by the requester from the request message. It fetches the identity of the requester in native format, uses the appropriate identity mapping module and converts that to an appropriate DN or a SAML subject and checks with the one requested by the requester. If those match, it then generates and signs the requested security token. Before issuing requested token the CMS may check some local policies and put some restrictions on the generated token.

![Fig. 3. Kerberos → PKI/SAML credential mapping](image-url)

For this mapping the CMS and the local KDC are located in the same administrative domain, and they share the same secret key (Kerberos keytab file). By this the CMS can verify the requests from only valid and authenticated Kerberos users. The CMS indeed holds a long lived certificate to sign generated security tokens. Then any relying party, who trusts the CMS, can trust and verify the security tokens issued by the CMS. Our design of the CMS has been aimed to be compliant with the Short Lived Credential Service profile (SLCS) when issuing short lived X.509 credentials [11]. The CMS acts as a SLCS X.509 Public Key Certification Authority (SLCS PKI CAs), which issues short-term credentials to end-entities, who themselves control their key pair and their activation data. The CAs described by this profile act as independent trusted third parties for both subscribers and relying parties within the infrastructure. These authorities have to use a long-term signing key, which is stored in a secure manner as defined in the Profile.

2) SAML/PKI → Kerberos: For exchanging an X.509 certificate or a SAML assertion for a Kerberos ticket we investigated on two approaches: i) issuing temporal Service Tickets and ii) approaching a Kerberos cross-realm authentication.
The former, requires the CMS to issue a temporal Kerberos ST, given a X.509 certificate or a SAML assertion. This requires a trust relationship between the CMS and target services that must be set up by sharing corresponding master keys between the CMS and the KDCs of target domains.

The latter, approaches a solution similar to the Kerberos cross-realm authentication in which a principal in one realm can authenticate to principals in another realm. This is technically possible by sharing an encryption key (a cross-realm secret) between the KDCs of both realms and allowing principals to use their TGTs for obtaining a “cross-realm TGT” from their local KDCs. Such cross-realm TGTs can be used to request service tickets from remote KDCs. In this approach, the CMS could technically act as a local KDC that upon receiving a request provides a cross-realm TGT to be presented to remote TGSs for obtaining a ST for using remote service.

For the first approach, the master keys are the subjects of compromised, since they leave the KDC, and the key management becomes arduous when many KDCs and CMSs are involved (a change of a master key could involve many key updates). Moreover, if administrators of a Kerberos domain want to revoke the right of accessing a particular service by the CMS, the key for that service must be changed for all other CMSs who share the same key.

For the second approach, an interaction between the client and the KDC is needed. The KDC treats the client as a Kerberos principal from another trusted Kerberos domain. It could also be easier to manage the keys, as only one key is being delivered to the CMS. There is also no shared master keys to be compromised. If the key shared by the CMS and the KDC is stolen, it is enough to only revoke or change that key, and finally the KDC can change the shared keys as needed, without any changes on the CMS (the shared key will still work). However, there is still the well-known scalability issue when leveraging a cross-realm authentication. To reduce the number of shared keys, Kerberos V5 proposes the transitivity of trust. Depending on the number of realm participated in trust establishment, the difference is $O(n^{n+2})$ versus $O(2^{n}n)$ which can make a big difference on the number of keys needed to be managed.

We approach a kind of Kerberos cross-realm authentication for this mapping as depicted in Figure 4.

1. The client generates a request, and attaches the X.509 certificate’s information. It signs the request with the client’s private key (whose corresponding public key is inside the certificate) and sends that to the CMS. The message must also specify the realm of the target service to let the CMS know which shared key and remote TGS principal must be used when constructing the ticket.
2. The CMS receives the message and fetches the requester’s public key from the certificate and verifies the signature. If the verification is successful according to the CMS’s policy, the CMS generates a cross-realm TGT. A random session key for the communication between the remote TGS and the requester will also be generated. The encryption algorithm of the key must be supported by the remote realm specified in the request (this is part of trust establishment process). The requester’s principal name will be the Canonical Name (CN) of the user’s DN in the certificate, and the realm name would be the CMS local realm name. The mapping from a DN to a Kerberos principal name, like other mapping scenarios, is performed by an appropriate plugin module. The “authentication time” and the “start time” are set to the current time, and the “end time” will be set appropriately. All this information will be encrypted using the key shared between the TGS of the specified remote realm and the CMS. Then, the TGS will be able to decrypt it and get the session key to communicate with the client. The principal name of the service is now filled with the remote TGS principal name. The generated credential is packed as the response message and will be sent to the requester.
3. The client receives the TGT, stores it in its local cache, decrypts the session key with its associated private key.
4. As any usual Kerberos cross-realm operation, the client presents the generated TGT to the remote TGS, proving that the client possesses the session key included in it. The TGS will then issue a ST for the requested service with a new key for the client-service communication.
5. The user may now request the service with the ST, and if authorized the application server will execute the requested actions.

The trust between the CMS and any remote KDC is established by a shared key. The remote KDC’s administrator creates a secret for the remote TGS principal, which gives a certain amount of rights to the tickets issued by the CMS. This means that the CMS will be able to access some resources at the remote realm, which might not be all services provided. Then, when issuing the Kerberos ticket to a requester, the CMS will delegate the right of using all services that it is entitled to use at that requester. However, depending on the TGS and the target service’s policy, the access could be denied for certain requesters according to the service’s access control mechanism.

3) UsernameToken → PKI/SAML: Exchanging a UsernameToken for an X.509 certificate or SAML assertion takes almost the same request/response SOAP messaging as described for the Kerberos approach. The main differences are when generating and attaching claims to the request message and establishing trust relationship between the CMS and the local identity provider.

If the CMS is enabled to access the database of passwords or the hash format of users’ passwords, then the username and...
password of requester can be sent to the CMS as described by the WS-Security specification and the UsernameToken profile. This enables direct way of authenticating requesters to the CMS. For this the RST element of the SOAP body must include a “wsse:UsernameToken” element as the claim of request to specify for which identity the exchange should be performed. Within the “wsse:UsernameToken” element, a “wsse:Password” element may be specified with the type “wsse:PasswordDigest”. In this case the requester must use both the “nonce” and “create” timestamps in the request as described by the WS-security specification and the Username-Token profile.

If the CMS has access to neither a plain text nor a digested format of passwords, the local IdP is required to share a secret with the CMS for enabling secure communication, and also provides the requesters with an authenticator token. An authenticator token is a tuple including a username, a time-stamp, a nonce and a session key which is signed and encrypted by the shared secret key. Along with this authenticator, the IdP sends also a session key encrypted with the user’s password to the requester. For the reasons of scalability and the ease of management, the secret key can alternatively be replaced by a public/private key pairs belonging to both the local IdP and the CMS.

The identity mapping module for this use-case maps the username of a user to an equivalent X.509 DN or a SAML subject corresponding to the agreed trust relationship between the CMS and the relying domains. The trust between the local IdP and the CMS should also be established either via sharing the database of passwords or a shared secret key which is used for secure communications between the CMS and the IdP. This key is the one used for encryption of the authenticator token.

A full implementation of this framework has been built on top of several standards and open-source components and technologies including SOAP, WS-Security and WS-Trust specification, Apache WSS4J, Apache Axis and JAAS (Java Authentication and Authorization Service). The CMS components can be used as a standalone service to be invoked by clients for issuing security tokens or as a set of Java API’s and ready-to-use libraries for developers (For this work we have completed and extended the existing WS-Trust implementation WSS4J by more functionality and fully implementing the OASIS Kerberos token profile).

IV. REAL GRID USAGE SCENARIOS

We have integrated and used our framework in two real grid usage scenarios for which the required application technologies were already available. Those are the NextGrid KINO application and PURSe.

In the NextGrid project we performed a validation of our CMS components, based on a real grid usage scenario with KINO application [2]. 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. However, this facility is only needed for a small subset of their work, thus KINO cannot justify buying a large computational cluster. In this case we successfully made a KINO end-user, who was authenticated via his local Kerberos domain, to access some rendering services provided by GRIA [12] service provider. The service provider could only support the PKI infrastructure for its authentication purposes. We used the CMS in this dynamic infrastructure to convert identity credentials between Kerberos tickets and X.509 certificates [2].

The Portal-based User Registration Service (PURSe) [13] is a set of tools and Java APIs, developed for constructing portal-based systems that automate the user registration, the creation of PKI credentials, and finally the subsequent credential management. A typical PURSe-based portal allows users to register via a Web page and later use a username and password for obtaining X.509 proxy certificates for their authentication purposes. The original implementation and functionality of PURSe were limited to the registration of users and further generating X.509 proxy certificates. We integrated PURSe with the CMS to make this portal capable of providing other formats of security credentials such as SAML assertions, X.509 certificates and Kerberos tickets in exchange for a username/password via a standard interface and service call.

V. RELATED WORKS

The KX.509 protocol [14], provides a mechanism for obtaining X.509 identity certificates based on a Kerberos domain login identity. Two main components of this protocol are the KCA (Kerberized Certification Authority) and the KX.509, which is a standalone client program that acquires a short-lived X.509 certificate from the KCA for an authenticated Kerberos user. As for the differences, the CMS uses an standard interface for exchanging diverse kinds of security tokens where the KX.509 does not. Our approach is more general and provides support for more use cases in the sense that it can let both the user or any delegated entity request for exchanging a security token (not only a Web browser as approached by KX.509).

MyProxy [15] is a software for managing PKI security credentials. It combines an on-line credential repository with an on-line certificate authority to allow users to securely obtain credentials when and where needed. MyProxy supports storage of multiple credentials (X.509 proxy certificates) per user and retrieval as needed using username and password. SACRED (Securely Available Credentials) [16] is an IETF framework and protocol for credential portability to allow secure, authenticated access to security credentials. This project provides SACRED client and server implementations. CredEx [17], is a Web Service which provides a secure storage of credentials and enables exchanging of different credential types using the WS-Trust token exchange protocol. It can only performs a credential mapping mechanism for exchanging username-password token to X.509 certificates.

1http://www.nextgrid.org

2This integration has been done as part of the Google Summer Of Code (GSoC 2008) and the code is available from the Google Code.
The works described above basically develop an on-line credential repository to ease and secure the process of credential management. Although the need for such on-line credential repositories are fully motivated in many use-cases, however we should emphasize that our solution was intentionally designed for just not storing credentials on a remote and online repository; it is rather designed to perform an on-the-fly credential mapping to make a non-vulnerable, lightweight, easy operate and quick to configure service. Furthermore, none of existing solutions can support a full mesh credential exchange as provided in our work.

Shibboleth [18] is a project that has created an architecture and an open source implementation of it to provide federated identity and single sign-on (SSO) features across different institutions. This means that members within the same federation can share identity information, agree in a common set of policies, and create a trust relationship. Shibboleth is designed to be used in Web browsers, and checks if a member who navigates with the browser is entitled to access a resource located in another organization, based on the information that the home institution holds about that users.

Trust Negotiation is an approach presented in [19] by which two strangers carry out a bilateral and iterative exchange of attribute credentials to gradually establish trust in one another. The credential mapping solution can complement a system employing trust negotiation by providing security credentials in different formats. Our approach sets out to solve the problem of translating credentials when parties involved in a trust negotiation process are requested for a particular format of a security credential which does not exist locally. There are indeed some benefits of leveraging a trust negotiation mechanism in our approach, that is determining what kinds of security credentials are required for accessing a Web service before requesting them from the CMS.

VI. CONCLUSION AND FUTURE WORK

In this paper, we highlighted that the heterogeneity of security infrastructures used in grids, requires a credential mapping mechanism which is capable of exchanging different kinds of security credentials. We described the development of a credential mapping mechanism that is built based on off-the-shelf standards and technologies. The architecture presented in this work is formalized as part of the next generation grid architecture and implemented software components were used in some real grid scenarios which are available to the community in open source format. Due to the importance of supporting shibboleth identity we are currently investigating on how to integrate the CMS with the Shibboleth infrastructure in order to exchange the Shibboleth identity to and from other security credentials supported by our framework.

Acknowledgment: We would like to thank Rachana Ananthakrishnan and Frank Siebenlist from the Argonne National Laboratory for their feedback on the integration of the CMS with PURSe. We would also like to thank Masashi Nakamura and Sina Khaknezhad who took part in the implementation and integration of the CMS with PURSe.

REFERENCES

Paper VI: Dynamic Trust Federation in Grids

Dynamic Trust Federation in Grids

Mehran Ahsant1, Mike Surridge2, Thomas Leonard2, Ananth Krishna2 and Olle Mulmo1

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.

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.

**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 encompassing 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].

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.

Background

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.

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.

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.

Review of Standards and Specifications

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.
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.

**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.

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.

**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.

![Figure 4. Key architectural challenges](image)

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.

**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.

**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.

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.

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.

**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-
sent, 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.

References

2. EC IST Project 511563: The Next Generation Grid.
18. Shibboleth v 1.2.1 Scott Cantor, Steven Carmody, Marlena.Erdos, et al. h
23. See the work of the TERENA Task Force on Authentication, Authorisation Co-ordination for Europe, via http://www.terena.nl/tech/task-forces/tf-aace/
Paper VII : Streamlining Grid Operations: Definition and Deployment of a Portal-based User Registration Service

Streamlining Grid Operations:
Definition and Deployment of a Portal-based User Registration Service

Ian Foster,1 Veronika Nefedova,1 Mehran Ahsant,3 Rachana Ananthakrishnan,1 Lee Liming,1 Ravi Madduri,1 Olle Mulmo,3 Laura Pearlman,2 Frank Siebenlist,1

1Argonne National Laboratory, 9700 S. Cass Ave., Argonne, IL, 60439
2Information Sciences Institute, University of Southern California, 4676 Admiralty Way, Marina del Rey, CA 90292
3KTH, Teknikringen 14, 100 44 Stockholm, Sweden

Abstract

Manual management of public key credentials can be a significant and often off-putting obstacle to Grid use, particularly for casual users. We describe the Portal-based User Registration Service (PURSE), a set of tools for automating user registration, credential creation, and credential management tasks. PURSE provides the system developer with a set of customizable components, suitable for integration with portals, that can be used to address the full lifecycle of Grid credential management. We describe the PURSE design and its use in portals for two systems, the Earth System Grid data access system and the Swegrid computational Grid. In both cases, the user is entirely freed from the need to create or manage public key credentials, thus simplifying the Grid experience and reducing opportunities for error. We argue that this capturing of common use cases in a reusable “solution” can be a model for how Grid ease-of-use can be addressed in other domains as well.

Key Words – certificate, certificate authority, credentials, grid, grid security, portal security, user registration, public key.

1 Introduction

A typical Grid application requires that a set of users share resources of various kinds in a controlled manner. To this end, many existing Grid deployments use the public-key infrastructure (PKI)-based Grid Security Infrastructure (GSI) [10] as a basis for secure user single sign on and subsequent authentication of users and resources prior to authorization. GSI defines and implements useful algorithms for authentication and delegation. However, the tasks of creating and managing the PKI credentials [14] used by GSI can be significant sources of complexity, user difficulty, and even error (and thus insecurity) in Grid deployments.

These considerations motivate our design of the Portal-based User Registration Service (PURSE), a set of tools for developing portal-based systems that automate user registration, the creation of PKI credentials, and subsequent credential management. A typical PURSE-based portal allows users to register via a Web page, triggering a vetting process, upon successful completion of which a credential is created and managed on their behalf. Subsequent access is provided via a username and password, an authentication scheme that is familiar to most users The system also allows a user access to short term credential, if needed. It is important to note that while the authentication scheme looks to the user like a username/password scheme, it is in fact not. After the user authenticates to the portal with his username/password, the short-term PKI credential is generated for the user and the portal will use that credential to further authenticate the user to the...
remote Grid sites. A separate administrator interface allows a portal administrator to approve requests, revoke credentials, and so forth. By streamlining and codifying these various steps, PURSE-based systems can significantly reduce barriers to the integration of new users, overheads associated with credential management, and opportunities for error—and thus simplify the development of usable Grid applications.

An important PURSE design goal was to support the creation and use of PKI credentials of varying “quality.” Different access control policies often are associated with different resources and operations. For example, write access to archival storage may require stringent verification of the identity or attributes of a requestor, while read access to Web pages may require only audit of a (by comparison) weakly authenticated identity. The definition and enforcement of such policies can be a significant source of complexity in Grid application deployments, because of the need not only to implement policies correctly but also to achieve appropriate tradeoffs between operational security and ease of use. Thus, PURSE mechanisms allow for the automatic creation of credentials following either simple online registration or stringent identity verification, and for the upload of existing credentials.

The PURSE implementation is not particularly complex, being based on an integration of a number of existing components, including GSI libraries, the MyProxy online credential repository, the SimpleCA credential generator, and portal tools. This implementation approach of integrating existing components to construct a reusable “solution” that addresses an important set of use cases is one that we hope will be pursued by many other Grid developers.

The rest of this paper describes in turn the PURSE system (Section 2), two PURSE-based portals (Section 3), the sample registration portal distributed with PURSE (Section 4), and a security analysis of our system (Section 5). We conclude in Section 6 with a brief look at future work.

2 System Description

The PURSE user registration system is a collection of Java APIs designed to work as a backend for a front-end user interface, typically a Web portal, to ease registration and credential management. Driven by user requests through the interface, this Java code stores user contact information, generates and stores new credentials for users, and allows for subsequent use of those credentials to access Grid resources. The system has functionality to support credential renewal and revocation. This functionality can be accessed through a well-defined API and is easily configurable.

The system is built on some common tools, as follows:

- A JDBC-compliant database to persist user data. (MySQL is currently used.)
- A certification authority to generate and sign user credentials. Depending on application requirements, either SimpleCA [6] or an external CA can be used for generating and signing users credentials.
- The MyProxy server [3, 11] to store user credentials
- JavaMail [2] to send and receive notifications to the user and CA operator. In addition, an operator can perform certain administrative tasks remotely by sending cryptographically secured emails using S/MIME technology [12].

2.1 Typical Usage Scenarios

A PURSE user must first register with the PURSE system. This is a one-time event that must precede any other use of the system. Registration involves three principal steps.
1. The user accesses the registration page on the portal and enters relevant information (e.g., contact information, desired user name, desired password).

2. PURSE stores the user information in a database and, using the contact information provided, sends the user an email message requesting confirmation of the request. The message provides a link that the user can click to confirm the request. This step (Figure 1a) helps to prevent registration errors and to verify the legitimacy of the email address.

3. Upon confirmation, the submitted request is sent via an email message to the registration authority (RA) configured in the PURSE system. The RA operator reviews the information provided by the user, and decides whether to approve or reject the request based on criteria of their choosing. If the request is rejected, an email is sent notifying the user of the decision. If the request is approved, PURSE generates long-term user credentials, signed by a certification authority using SimpleCA, and stores them in the MyProxy server (Figure 1b).

4. An email is then sent to the user notifying them that registration has completed successfully.

In a variant of this scenario, the user may instead supply an existing credential during the registration process. The same registration and approval process is followed, but following approval by the RA, the user is instructed to upload his existing certificate into the PURSE MyProxy.

In a second variant of this scenario, an external (possibly offline) CA may be utilized instead of a local SimpleCA instance. To allow for a wide range of deployment constraints, this communication may either be performed out of band by the RA or in a more automated fashion by PURSE itself via email notifications, secured by S/MIME. While slightly more complex to deploy, it provides for a clean separation of the registration and user credential management from
the operations of the CA, which is a highly tenable goal from a security point of view. It also provides for a backwards-compatible integration with an existing CA infrastructure.

Following successful registration, the user can use the username and password to log in to the portal. The portal then retrieves a short-term credential (a proxy certificate [15]) for the user from the MyProxy service and uses that credential on behalf of the user to access VO resources as directed by VO-specific logic in the portal. Moreover, experienced users can interact with the MyProxy server directly to obtain short-lived credentials locally to their desktop, should they require that. This interaction is presented in Figure 1c.
2.2 Overview of Registration System APIs

PURSE is structured as a set of building blocks that can be used to create a fully functional Web-based portal for accessing the Grid. The modules are available as Java archive ("jar") files and can be plugged into any front-end interface such as an existing portal. We describe the high-level functionality and APIs for these building blocks in the following.

New User Registration

Register user: This step initiates user registration by storing relevant user information, including requested username and user email address in the backend database. Once the information is stored, an email is sent to the user requesting confirmation of request.

Process user request: This step is triggered by the user’s confirmation of the request to the registration system. An email is sent to a configured RA email address with instructions for the RA to access the user details.

Accept user: This module is invoked when a RA accepts a particular user’s request. The following steps are performed.

- If the user wishes to use his own credentials, the user is sent an email with a link to a simple Java MyProxy client that uses Webstart. The user can use the client to upload his credential to the PURSE MyProxy server.
- If the user does not have his own credentials:
The PURSE code base generates a user certificate request.

- The request can be signed either by a local CA configured in the portal (via SimpleCA) or by some external CA, depending on application requirements.
- The resulting long-term credentials are loaded onto a MyProxy server.
- The database is updated to set the user’s request status to “accepted.”

- In both cases, an email is sent to the user indicating that registration is complete.

Reject user: If the RA rejects the user, this module is invoked. It sends an email to the user and updates the user request status to “rejected.”

Managing Registered User

Revoke user: This module deletes the user from registration system. The user’s credentials are removed from the MyProxy server, and the user’s status in the database is set to “revoked.”

Renewal notice: This operation can be run as a periodic task to send mail to all users whose credentials are due to expire in some configured timeframe.

Renew user: This operation is triggered by a user attempting to renew membership and sets the user status in the database to “renew.” If the renewal request is granted, an API is provided to generate new long-term credentials for the user and store them in the MyProxy server.

Tools for Registered Users

Change password: This operation allows a registered user to change his password.

2.3 PURSE Setup

Establishing a PURSE-based portal involves two sets of steps. The first set involves developing the portal code (or integrate PURSE calls into an existing portal). The second set includes establishing the backend database used to maintain user information (e.g., MySQL), a SimpleCA certificate authority (or configuring PURSE to access an existing CA), and the MyProxy server used to store user credentials. Complete instructions for PURSE installation and testing are on the PURSE Web site [5].

3 Deployment Use Cases

We describe two production deployments that have served both to drive PURSE requirements and to validate PURSE functionality.

3.1 Earth System Grid

PURSE was initially developed for the Earth System Grid (ESG) [8], a U.S. Department of Energy project to provide online access to climate data. We describe here the ESG production deployment as an example of how the registration system can be used. The following details are specific to deploying the Registration System for ESG.

The ESG portal needs to support two different classes of users: a small number of ‘privileged’ users who can access all the data on ESG, including the newest data produced (by CCSM model [13]), and the rest of the users who can access only the publicly available, previously published data. All users must have valid GSI credentials in order to access the data stored on various storage systems (NCAR MSS, NERSC HPSS, GridFTP servers) throughout ESG. This combination of authentication and authorization requirements motivated the development of PURSE. All users are assigned to the specific user groups during the registration process based on ESG policy. When a user wants to access some data via ESG portal, the request is validated by the portal using the user’s group assignment. The number of user groups is configurable and
depends on ESG policy. ESG is using the standard workflow for user registration, described in Section 2.1.

ESG has more than 1800 registered users as of January 2006. New users can register with the ESG portal by following the Registration link from the main ESG site (https://www.earthsystemgrid.org). To allow for PURSE trials, the ESG registration system provides limited access to ESG data to anyone if the “Statement of Work” is filled to express interest in seeing PURSE in action. The functionality is available today and can be tried out by anyone.

3.2 Swegrid

Swegrid [7], a distributed computational resource in Sweden, uses the PURSE libraries to provide a registration system for its users. This system uses PURSE to meet the Swegrid requirements for providing users with a certificate signed by an external certification authority. While a SweGrid-local CA is currently used, the goal is to switch to an external CA that is a member of, and operates in accordance with policies defined by, the European Grid PMA (http://www.eugridpma.org). That way, the user’s credentials will be honored in all of Europe (and beyond), should the user want to access non-Swegrid resources.

The main difference between the Swegrid and ESG registration system is the workflow for issuing certificates. In contrast to ESG, the Swegrid portal after registering the user in its local database sends a notification to the Swegrid registration authority containing a link that can be used by the RA to validate the user’s information and to verify their identity against the papers signed and sent by that user. Upon the RA’s approval of the identity of the user, the portal then accepts the user and generates a certificate request that is sent to the configured external CA, using a cryptographically signed email. The CA may also use a similar link to access the local information saved on portal database in order to verify the user’s identity. Upon approval, CA signs the certificate and sends it back to the Swegrid portal. The portal receives the signed certificate from the CA and uploads this to the MyProxy server. A confirmation email then is sent to the user.

4 Sample Portal User Registration Interface

The PURSE distribution includes code for a sample registration portal that may be adapted to meet specific application requirements. Figure 1 shows the architecture of this system.

The sample registration portal solicits basic data from the user, generates a certificate request to the VO operator, (following approval) generates a certificate and stores it in the MyProxy server, and gives the user an identifier and password for MyProxy access. A separate administrator interface allows a CA operator to accept or reject user requests and also to revoke issued certificates.

User registration involves the following steps.

1. The user fills in the sample registration portal’s entry page, shown in Figure 2, to submit his registration request. The password will be used later to log into the portal application.

2. The Sample Registration Portal verifies the user’s email by sending the mail in Figure 3(a) to the provided email address.

3. Following user acknowledgment, the CA operator receives an email notification when a new account is being requested, as in Figure 3(b).

4. After receiving this notification, the CA operator logs in to a secure web site (Figure 4) and views the request.
5. After the user’s credentials are generated and uploaded into MyProxy the user receives an email notification, as in Figure 3(c).

---

**Figure 2:** Screenshot of the PURSE sample user registration interface
(a) Email confirmation step: message sent to user

Date: Thu, 1 Jul 2004 14:25:47 -0600 (MDT)
From: esgport@ucar.edu
To: john_smart@ucar.edu
Subject: ESG Registration

The Earth System Grid (ESG) Portal received a request for a new user account that uses your email address. Click on the link below to confirm your request (NOTE: you will not be able to login until you receive an email from the portal administrator indicating your request has been approved):

https://www.earthsystemgrid.org/security/confirmRequest.do?token=000000fd-7c62-605c-fffd0e-766ad9819840

If you did not request this account, please inform us at esg-admin@earthsystemgrid.org.

Thank you,
ESG System Administrator

(b) Email sent to CA operator for approval

From: esgport@ucar.edu
Date: July 1, 2004 12:17:07 AM MDT
To: esg-ca@ucar.edu
Subject: ESG Registration

A request has been made for user account on the ESG Portal. You may access the details of the request by clicking on the following link.

https://www.earthsystemgrid.org/administration/accountRequestData.do?token=000000fd-2e0e-5d33-0006ac0-8387f64897be

(c) Registration confirmation email sent to user

Date: Thu, 1 Jul 2004 14:34:52 -0600 (MDT)
From: esgport@ucar.edu
To: john_smart@ucar.edu
Subject: ESG Registration

Your request for an account with the ESG portal has been approved.
5 Related Work

While a number of efforts solve the various issues PURSE addresses, we view PURSE as an integration of solutions to provide an end-to-end system for user registration and credential management. Here we discuss some of the point solutions available:

**CACL**

Tools like CACL and MyProxy Online CA address the need for a simple and automatic way of issuing certificates to users. CACL provides a server daemon that can be contacted to get certificates. The PURSE system accomplishes that by providing a web interface to a SimpleCA backend.

**MyProxy CA**

The MyProxy service has recently been enhanced with the addition of an integrated online CA, which allows users to obtain short-lived End Entity Certificates. As PURSE is already fully integrated with MyProxy, we are currently investigating how and in what form we can make this additional functionality available to PURSE deployments in the future.
The Grid Account Management Architecture (GAMA) activity [1, 9] has produced a similar system in parallel with our PURSE development, using roughly the same architecture. GAMA differs from PURSE primarily in implementation details. For example, GAMA is implemented as a server and a set of portlets that communicate with that server, while PURSE is implemented as a Java library that can be called from servlets, portlets, JSP pages, etc. GAMA uses CACL for generating and storing credentials while PURSE uses SimpleCA and MyProxy. GAMA does not support uploading existing credentials nor does it support an email driven notification system to trigger the various steps in user registration and management. We view this parallel evolution as a demonstration of the importance of this technology.

GridShib

GridShib is an NSF-funded project at Argonne National Laboratory, NCSA and the University of Chicago [GRIDSHIB]. Its goal is to leverage the Shibboleth [SHIB] attribute services for the webservices clients and grid application services. The GridShib and PURSE technologies are very complementary, and many GridShib deployment scenarios involve MyProxy services to bootstrap the initial authentication where PURSE could potentially be used for the initial user registration and credentials creation.

6 Security Considerations

PURSE provides a complete life-cycle solution for end-user credential management. It is targeted primarily at solving several operational problems often associated with PKI:

- Automate as much of the registration process as possible, making it more useful. Access to the registration pages are secured using HTTPS.
- Ease the burden on the users and eliminate the security risks associated with users managing their own credentials. PURSE provides a controlled and auditable environment where the long-term credentials are stored, secured by well-known and trusted technologies.
- Secure remote access to the credentials for the mobile computer user. This has been one of the principal problems with Grid portals in the past.
- Control credential renewal and removal.

The long-term credentials are stored on a MyProxy server running on the same machine as the portal, which in turn must be considered the “weakest link” from a security perspective. In order to mitigate this risk, the portal can be isolated and given a limited and controlled access to the SimpleCA and MyProxy installations by using privilege separation techniques. To illustrate this, we have successfully experimented with hardening the Swegrid installation by running the registration portal and the MyProxy software in different local user accounts. We then use the sudo software, configured to allow the user account of the portal the additional privilege of executing a few well-defined administrative commands in the MyProxy user account, for upload, renewal, and removal of long-term credentials. With this approach, the long-term credentials cannot easily be stolen or copied from the system in the case that the portal account (which is the most exposed) gets compromised. In addition, we have strengthened the security of PURSE such that a local privilege escalation attack (the attacker has to obtain root access) is required in order to obtain physical access to the (still password-protected) long-term credentials.

In analogy with the MyProxy and portal isolation approach above, it should be a deployment option to completely separate the CA installation from the other components. As mentioned, the
Swegrid installation will be configured to use an external CA in the near future, and such a configuration will be fully supported by the PURSE credential processing commands.

Another attack vector is to target the MyProxy server or the operating system directly (the SimpleCA and database components do not expose any external interfaces that an attacker can utilize). The MyProxy technology has existed for five years and is widely used and trusted in the Grid community. MyProxy has a well-known threat model with well-established principles on operational policies and management. We recommend that PURSE be installed in line with these guidelines: for example, PURSE should run on dedicated hardware with limited administrative access (console login only).

We emphasize that the PURSE system does not remember or store any passwords. The user supplies a password when he/she registers with the portal application, and this password is used to generate a key pair and encrypt the private key. The password is not stored. When the user logs into the portal, he/she supplies the password again so that the private key can be decrypted. Thus, in order to obtain access to any long-term credential (including the SimpleCA signing key), the password has to be provided (or can be guessed/cracked through some time-consuming brute-force method). This situation holds true even in the case of stolen hardware.

When PURSE generates new credentials for users at registration time, the credentials are “signed” by the portal application’s SimpleCA service. The portal administrator established SimpleCA when PURSE was installed. Typically, the signature identifies the portal application, e.g., “Earth System Grid Certificate Authority.” In order for these credentials to be recognized and trusted by Grid services operated by third parties, the operators of those services must configure their code to trust the portal’s certificate authority. If this is not acceptable, users may obtain credentials from a trusted CA using other mechanisms and supply those existing credentials to PURSE at registration time.

7 Summary and Next Steps

PURSE provides a set of tools that can be used to construct Web-based user and administrative interfaces for user registration, credential management, and Grid access. PURSE automates the process of obtaining PKI credentials for users; provides for the secure storage of credentials; allows users to use existing Grid credentials, if available; and provides for Grid access via Web portals and secure username-password authentication.

In future releases, we plan to work toward simplifying PURSE installation by creating an easy packaging solution for this system. In addition, we need to adapt the current implementation to separate the credential repository from the rest of the portal logic, so as to permit hosting of the credential repository on a secure system and provide support for multiple RA authorities for varied vetting strength requirements. The Open Grid Collaboration Environment (OGCE) project has produced a set of JSR-168 compliant portlets to simplify the process of incorporating PURSE into a web portal.

Communities currently using PURSE include the Earth System Grid, SweGrid, and the Large Synoptic Survey Telescope (LSST) community.

Acknowledgments

PURSE was first developed at Argonne National Laboratory in collaboration with the Earth System Grid, with support from the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract W-31-109-Eng-38. Staff at NCSA and the University of Chicago supported by NSF Middleware Initiative’s GRIDS Center [4] contributed
to the extensive pre-release testing. KTH made a detailed security analysis of the original software, and provided the sample portal interface and added several features to the API, as part of a project funded by the Swedish Research Council.

References


The submitted manuscript has been created by the University of Chicago as Operator of Argonne National Laboratory ("Argonne") under Contract No.W-31-109-ENG-38 with the U.S. Department of Energy. The U.S. Government retains for itself, and others acting on its behalf, a paid-up, nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government.