1-1-2007

A web service based architecture for authorization of unknown entities in a Grid environment.

Jordan Rivington
University of Windsor

A Web Service Based Architecture for Authorization of Unknown Entities in a Grid Environment

by

Jordan Rivington

A Thesis
Submitted to the Faculty of Graduate Studies
through Computer Science
in Partial Fulfillment of the Requirements
for the Degree of Master of Science at the
University of Windsor

Windsor, Ontario, Canada
2007

Your file Votre référence
ISBN: 978-0-494-35021-8
Our file Notre référence
ISBN: 978-0-494-35021-8

Abstract

As the "Grid" becomes closer and closer to fruition, not as a static entity, but as a dynamic global solution, certain issues must be addressed. These issues include, but are not limited to, virtualization, discovery, accounting and fault tolerance [Kesselman2002]. Many of such issues are being addressed with the advent of OGSA/OGSI and related service-based changes, which are being applied to grid-based computing [Kesselman2002]. Of course, these changes also include security, but not from the point of view of unknown entities. Before we can successfully achieve pervasive computing, we must leave behind, or at least augment, the current notion of pre-established trust [Rivington2004]. This requires too much administrative overhead, and is not realistic in a distributed environment where processing may occur across thousands to millions of independently administered nodes [Kirschner2004]. Therefore, the purpose of this paper is to present a new architecture for distributed authorization which will surpass current schemes in scalability, due to a lower administrative overhead, while also supporting the notion of arbitrary entities.

Dedication

I dedicate this work to my parents. Without their support, mentally, emotionally, and financially, as well as their influence in general, I would not have become the person I am today. It took a little while for me to realize that I don't know everything and I have them to thank for it. It also took some time for them to convince me that education was in fact important. For this, and everything else, I am forever grateful.

Acknowledgements

Throughout the lifetime of this project, many major events have occurred in my life. I met my fiancé, we have planned a wedding, we have purchased a house, and we now have three pets chewing everything in sight. I am very happy with how my life is progressing. The completion of this project will be the proverbial icing on the cake. That being said, the first person I would like to acknowledge is my fiancé Kristin. She has supported me throughout this long process, and I could not have done this without her. Soon we shall be able to spend more time together without the laptop.

From a technical and mentoring point of view, I could not have asked for a better person. Dr. Robert Kent has played many roles in this project including technical consultant, devil's advocate, motivator, and friend. I still remember the day I went into his office to ask him if I could work with him for my Master's. He abruptly said "you have thirty seconds to tell me why I shouldn't kick you out of my office right now". I guess I said what I needed to say, because I have been working with him ever since. Although at times I became quite frustrated with his suggested changes of topic, I have since realized that it was all part of the learning process, and I cannot thank him enough for the guidance. In the end, I don't consider anything I have done a waste of time.

Most people are lucky if they can find one person to provide them with the necessary help and guidance required to complete this type of project. I was fortunate to have not only one supporter. Dr. Akshai Aggarwal was also always available to help me with the technical details of this project. For this, I cannot thank him enough.

Table of Contents

Preface 1
1 Introduction 2
2 Background Terminology and Technologies 5
2.1 Authorization 5
2.2 Web Services 6
2.3 XML 6
2.4 WSDL 7
2.5 UDDI 7
2.6 SOAP 8
2.7 OGSA 8
2.8 Globus (GT4) 9
3 Literature Review and Related Work 11
3.1 Authorization Systems 11
3.1.1 Akenti (1999) 11
3.1.2 Certificate Based Authorization Simulation System (2001) 12
3.1.3 MAFTIA1 (2001) 13
3.1.4 CAS (GT4) (2002) 13
3.1.5 KEYNOTE (1999) 14
3.1.6 PERMIS (2002) 14
3.1.7 CARDEA (2003) 15
3.1.8 FIDELIS (2003) 16
3.1.9 HP HSA (2003) 16
3.1.10 CONTEXT SENSITIVE (2003) 17
3.1.11 TERA (2004) 17
3.1.12 IBM TE (2004) 18
3.1.13 Shibboleth (2004) 18
3.1.14 VOMS (2004) 19
3.1.15 PRIMA (2004) 19
3.1.16 WS AA Authorization Framework (GT4) (2006) 19
3.1.17 General System Comparison 20
3.2 Methods of Authorization 21
3.3 Comparison 1: Authorization 21
3.4 Comparison 2: Features 22
3.5 Comparison 3: Implementation 24
3.6 Comparison 4: Arbitrary Entities 24
3.7 Authorization System Comparison Summary 25
4 Augmented Authorization System Using Reputation 27
4.1 Justification 27
4.2 Overall Architectural Comparison with AASUR 28
4.3 Problem Statement 28
5 AASUR 31
5.1 Design Context 31
5.2 Design Methodology 32
5.3 Design Assumptions 37

5.4 Reputation and Evidence 38
5.5 Architectural Assumptions 41
5.6 Major Components and Packages 42
5.6.1 org.aasur.ejbca.ca.AassurCaService (ACS) 42
5.6.2 org.aasur.resource.ResourceSiteServiceListener (RSSL) 44
5.6.3 org.aasur.reputation.AASURReputation (AR) 46
5.6.4 org.aasur.policy.UserClass (UC) 48
5.6.5 org.aasur.policy.UserClassPolicy (UCP) 49
5.6.6 org.aasur.policy.UserClassEngine (UCE) 50
5.6.7 org.aasur.policy.RiskFactorPolicyEntry (RFPE) 52
5.6.8 org.aasur.policy.RiskFactorPolicy (RFP) 53
5.6.9 org.aasur.policy.RiskFactorEngine (RFE) 54
5.6.10 org.aasur.policy.AccessDecisionPolicyEntry 55
5.6.11 org.aasur.policy.AccessDecisionPolicy 56
5.6.12 org.aasur.policy.AccessDecisionEngine 58
5.7 Supporting Components 59
5.7.1 org.aasur.user.UserSiteSimulator 59
5.7.2 org.aasur.common.Serializer 59
5.7.3 org.aasur.common.TextCompressor 61
5.7.4 org.aasur.common.FileContentExtractor 62
5.7.5 org.aasur.common.RandomNumberGenerator 63
5.7.6 org.aasur.common.ReputationExtractor 64
5.7.7 org.aasur.common.XMLDomParser 64
5.7.8 org.aasur.simulation.RandomJobGenerator 65
5.7.9 org.aasur.simulation.JobAction 66
5.8 Policy Formats 67
5.8.1 Risk Factor Determination Policy 68
5.8.2 Access Decision Policy 69
5.8.3 User Class Definition Policy 70
5.8.4 Local Blacklisting Actions Definition Policy 71
5.9 Message Formats 72
5.9.1 Resource Access Request 72
5.9.2 Point Action Notification 72
5.9.3 Resource Access Ticket 73
5.10 OTHER FORMATS 74
5.10.1 Certificate Point Format (Reputation) 74
5.10.2 Local Blacklist Format 75
5.11 Extensibility 76
5.12 Sample Walkthrough (SYSTEM FLOW) 77
5.13 Sample Walkthrough (AUTHORIZATION DECISION) 80
5.14 Current State 87
5.15 Limitations 87
5.16 Implementation 88
6 Development/Deployment Infrastructure 90
6.1 Required Components 90
6.2 Selection of Software 91

6.3 Installation and configuration 92
6.3.1 JBoss 92
6.3.2 Java 93
6.3.3 Linux 94
6.3.4 EJBCA 94
6.3.5 MySQL 97
7 Testing and Verification 99
7.1 Code Correctness Verification 99
7.2 Metrics 99
7.3 Procedures and Results 100
7.3.1 Test A: Certificate Retrieval Time (NO REPUTATION) 100
7.3.2 Test B: Certificate Retrieval Time (EMPTY REPUTATION) 102
7.3.3 Test C: Certificate Retrieval Time (0-100,000 ACTIONS) 104
7.3.4 Test D: Certificate Size Vs. Actions in Reputation 106
7.3.5 Test E: Access Decision Time (0-100,000 ACTIONS) 108
7.4 Performance Comparisons 110
7.4.1 Akenti 110
7.4.2 Permis 111
7.4.3 AASUR / Permis / Akenti Comparison Summary 111
8 Conclusion 113
8.1 Future Work 114
References 116
Appendix A: Environment Configuration Instructions 121
A.1 LINUX INSTALLATION 121
A.2 JBOSS INSTALLATION 122
A.3 MYSQL CONFIGURATION 123
A.4 EJBCA INSTALLATION 124
A.5 PREPARATION AND DEPLOYMENT 125
Vita Auctoris 129

Table of Tables

1. Table 1: Legal/Illegal Point Actions.....39

2. Table 2: Other Point Actions.....39

3. Table 3: Additional Evidence.....40

Table of Figures

1. Graphical Representation of the Grid Security Infrastructure (GSI).....10
2. Figure 2: System Comparisons 1: Authorization Methods.....22
3. Figure 3: System Comparisons 2: Features.....23
4. Figure 4: System Comparisons 3: Implementation.....24
5. Figure 5: System Comparisons 4: Arbitrary Entity Support.....26
6. Figure 6: System Comparisons 5: Overall Architectural Comparison.....28
7. Figure 7: AASUR Reputation Object.....41
8. Figure 8: AasurCaService Object.....44
9. Figure 9: ResourceSiteServiceListener Object.....45
10. Figure 10: AasurReputation Object.....47
11. Figure 11: UserClass Object.....49
12. Figure 12: UserClassPolicy Object.....50
13. Figure 13: UserClassEngine Object.....52
14. Figure 14: RiskFactorPolicyEntry Object.....53
15. Figure 15: RiskFactorPolicy Object.....54
16. Figure 16: RiskFactorEngine Object.....55
17. Figure 17: AccessDecisionPolicyEntry Object.....56
18. Figure 18: AccessDecisionPolicy Object.....57
19. Figure 19: AccessDecisionEngine Object.....58
20. Figure 20: Serializer Object.....61
21. Figure 21: TextCompressor Object.....62
22. Figure 22: FileContentExtractor Object.....63

23. Figure 23: RandomNumberGenerator Object.....64
24. Figure 24: ReputationExtractor Object.....64
25. Figure 25: ReputationExtractor Object.....65
26. Figure 26: RandomJobGenerator Object.....66
27. Figure 27: JobAction Object.....67
28. Figure 28: Sample Transaction Walkthrough.....77
29. Figure 29: TEST A: 1000 Certificate Retrievals. No reputation.....101
30. Figure 30: TEST B: 1000 Certificate Retrievals. Empty reputation.....103
31. Figure 31: TEST C. 1000 certificate retrievals. 100,000 action reputation.....105
32. Figure 32: Test D. Certificate size increase with 100,000 actions in reputation....107
33. Figure 33: Test E. Authorization decision time with 100,000 action reputation...109

Preface

This thesis is the direct result of my own work and includes nothing which is the outcome of work done with others, except where specifically indicated in the text. Of course, direction and suggestions have been given to me by my supervisor and other faculty. This thesis is not the same as any that I have submitted for another degree or any other qualification at any other educational institution. No part of this work has already been, or is being currently submitted for any such degree, diploma or other qualification.

1 Introduction

Grid computing encompasses the sharing of computational, storage, and networked resources that are controlled by different resource owners, typically independent institutions, and where the users of grid resources may be members of some, but not all, institutions [Foster2001]. Before such foreign users may be permitted to access remote grid resources it is necessary that their identities be authenticated and their requests for access to resources be authorized. The Open Grid Services Architecture (OGSA) provides a universally applicable and adopted framework for grid system integration, virtualization, and management [Kesselman2002]. It also provides several mechanisms with respect to security.

The various OGSA mechanisms which address security do so within a scope defined by a specified and limited lifetime of application. At the present time OGSA is still in a relatively infant stage of development with primary attention paid to issues of authentication which have been treated effectively, for the most part. Authorization mechanisms, however, are generally considered as underdeveloped and many problems still remain to be treated. New techniques must be integrated into OGSA to provide both new authentication as well as authorization mechanisms to address evolving security problems and schemas in modern distributed systems. [Rivington2004]

Though the security of web services has been addressed in the OGSA specifications, authorization is, at best, rudimentary. Currently, OGSA authorization relies on a static grid-map file, specific to each resource, which provides a mapping from a global ID

obtained using a public key certificate (PKC) to a local account [Keahey2002]. Local account rights are then used as the basis for granting access to resources. While this approach certainly has a place in the general authorization scheme of OGSA, especially in the case of pre-established collaborations and federations, it is not sufficient for an evolving distributed computing environment, where resources may be shared with individuals who may be unknown to the resource provider [Kirschner2004]. Moreover, if each new entity must be manually added to the local grid-map file the scalability of the system becomes limited [Kirschner2004].

Similar types of authorization manifest themselves in several other architectures and systems, through various novel approaches. Within all of these approaches there exist potential issues that affect the overall scalability, performance, administrative overhead requirements, and other characteristics which would limit the overall scale of use of these approaches.

In this thesis we propose an architecture which collects and records actions performed by an entity throughout its lifetime in a grid. These actions are tracked through a subsystem, and are recorded as points which are then used as evidence. When combined with site specific policies, this evidence is used to determine an entity's reputation and to augment other local policies to obtain an authorization decision. This architecture, called Augmented Authorization System Using Reputation (AASUR) [Rivington2004], has been implemented and tested and various performance results are presented and discussed.

The remainder of the thesis is organized as follows. Chapter 2 introduces terminology and discusses a number of relevant existing software tools, components, and systems. In Chapter 3 we review several authorization approaches and systems presented in the literature. This discussion includes a critical review of the problems and limitations of those approaches. Chapter 4 is dedicated to developing the precise context of our AASUR approach to authorization and how it intends to address and overcome the limitations of other systems. In this context we state the thesis problem and provide a set of specific objectives, justifications and intended contributions relating to authorization for grid computing. Chapter 5 outlines in complete detail the Augmented Authorization System Using Reputation, including design context, assumptions, discussion dealing with reputation, components, policies/messages and associated formats, discussion regarding extensibility, and sample transactions to illustrate functionality. We also discuss the current state of AASUR, inherent limitations, and various implementation details. Chapter 6 provides details regarding the development and deployment infrastructure used for AASUR. Chapter 7 discusses the verification and testing of our approach, including procedures, correctness, performance, and associated metrics leading to a comparison with several other existing systems. Chapter 8 contains our conclusions, including what is new and novel with respect to this system, and a discussion of possible future research work. There is one Appendix wherein we present a detailed procedure for installing and configuring required AASUR components to form a working system.

2 Background Terminology and Technologies

In order to provide a foundation and consequently an appreciation of the nature of the authorization problem in grid computing, we begin by defining conventional terminology used in discussions of security and describing the various software tools and technologies pertinent to web services, distributed systems, and authorization.

2.1 Authorization

In the context of grid and distributed computing, authorization is the act of determining what a user may be permitted to do within a computing system, including the set of resources for which access may be granted and also the actions that may be performed on those resources [Opplinger2000].

Typically, in the context of existing approaches, authorization is derived from authentication. This means that once a user's identity has been determined the actions that may be performed are derived from that identity [Lampson2000]. This has worked very well in the context of operating systems and other systems where control is generally centralized. This paradigm does not necessarily shift well into the realm of globally distributed processing where it is very likely that a resource will have no prior knowledge of a user. Creating a mapping between an unknown user and a set of associated rights becomes very difficult. Inevitably, centralized schemes will suffer from a lack of scalability since the central services must handle ever larger amounts of traffic volumes from growing numbers of users requesting authorizations. For this reason a different, more scalable approach must be examined.

2.2 Web Services

Current trends, including grid computing, are increasing the degree of reliance on networks; in short networked elements must communicate with each other necessitating the use of software modules to achieve this intent. Hence, web services are web based, compositions of small, well defined modules [Curbera2001]. The evolution of distributed computing, like networks, is very dynamic and web services reflect this dynamism. Modularity affords easier maintenance, code reuse, abstraction from unnecessary implementation details, and domain-wide access [Fiadeiro1995]. Once properly described (using descriptive languages described below) and maintained in a repository for discovery purposes (also described below), these publicly exposed modules can be accessed as Web Services using well known scripting languages, such as Hypertext Markup Language (HTML) and the eXtensible Markup Language (XML), and protocols, such as the Simple Object Access Protocol (SOAP) [Curbera2001]. All services are offered individually and completely abstracted from their underlying complexity and implementation details. These services can be used independently or composed to create more complex services [Koehler2003].

2.3 XML

The eXtensible Markup Language, or xml, is an Open Source standard proposed by the World Wide Web Consortium (W3C). It is a self-describing data format intended to enable data interchange among various applications and hosts across the Internet [W3C1]. Due to the simple tagged structure of XML, parsing is very straightforward, and many programming languages have built-in support for xml parsing. All xml documents must be well formed; that is, all opening tags must have closing tags unless they are

empty [W3C1]. This ensures that parsing of the xml is unambiguous. We note that all of the policies, formats, and reputation in our system, defined later in this thesis, are defined in xml based formats because of the many benefits inherited through its use.

2.4 WSDL

The Web Services Description Language, or WSDL, is an XML based language whose function is to support the definition of web services. In other words, WSDL defines a web service in terms of its publicly accessible interface, including method names, input parameters, and return types [Orth2002]. This definition specifies web services as a set of network endpoints (that is, ports) operating on a message payload [W3C2]. The services are abstractly defined in terms of messages and available operations, and consequently bound to underlying protocols in a concrete instance [W3C2]. This ensures that the individual definitions can be reused. Although WSDL is extensible, it is typically bound to SOAP, the Hypertext Transfer Protocol (HTTP), and multipurpose internet mail extensions (MIME) [W3C2].

2.5 UDDI

UDDI stands for Universal Description Discovery and Integration [ShaikhAli2003]. UDDI is a platform-independent, xml based registry which is available to store information about various businesses that have an internet presence. More specifically, UDDI registers web services that may be discovered and utilized. An entry in UDDI is composed of three parts [ShaikhAli2003]:

1. White pages – address/contact information
2. Yellow pages – business categories
3. Green pages – services exposed by the business

Essentially, UDDI exists to allow SOAP based web service queries to obtain WSDL documents that will enable other businesses or users to make use of the offered services.

2.6 SOAP

SOAP is an acronym which stands for the Simple Object Access Protocol. It is an XML based method of exchanging information between hosts [W3C3]. The SOAP messaging protocol is stateless, and operates within the context of simplex communication but can be extended using underlying protocols [W3C3]. SOAP can accommodate various messaging paradigms, but the most commonly used is the RPC style. From a web services point of view, SOAP is transferred over a common protocol, typically HTTP [W3C3]. In other words, SOAP has HTTP bindings. This allows the transfer of SOAP messages to be accomplished by any single machine on the Internet.

2.7 OGSA

The Open Grid Services Architecture, or OGSA, is the current core architecture that describes and specifies the alignment and augmentation of Grid and Web services technologies [Kesselman2002]. OGSA is a continuing development project coordinated through the auspices of the Open Grid Forum (formerly, the Global Grid Forum) [Foster2005]. The intent underlying OGSA is that it will contain a well defined set of basic interfaces that may be used to build more complex useful systems from a distributed computing viewpoint. This is analogous to the main paradigm of Web Services.

In the spirit of interoperability, OGSA also specifies open, extensible, and vendor-neutral capabilities [Kesselman2002]. OGSA is intended to be implemented using tools such as

WSDL and application development frameworks such as the Globus Toolkit, itself constructed, in part as a service oriented architecture [Foster2005].

2.8 Globus (GT4)

The Globus Toolkit is an application development framework for grid computing [Globus2005]. This middleware layer is intended to support all necessary mechanisms to achieve grid computing including job distribution, accounting, resource discovery, security, etc. For security, the GT4 supplies a component known as the Grid Security Infrastructure (GSI) [Butler2000, Foster1998]. As mentioned above, OGSA recommends employing web services to expose the Globus Toolkit as a Web Service entity which forms a fairly complete distributed processing system [Globus2005].

The GSI provides four distinct functions. All of which deal explicitly with the security aspect of grid systems. These include [Globus2005]:

1. TLS (transport-level) or WS-Security and WS-SecureConversation (message level) are used as message protection mechanisms in combination with SOAP.
2. X.509 End Entity Certificates or Username and Password are used as authentication credentials
3. X.509 Proxy Certificates and WS-Trust are used for delegation
4. SAML assertions are used for authorization

All of these functions are absolutely essential to the proper security of any grid system. For more details regarding these functions see the Figure 1 below. Notice the use of the

grid-map file as well as SAML for the authorization aspect of the GSI. Both of these will be discussed in more detail later in this thesis.

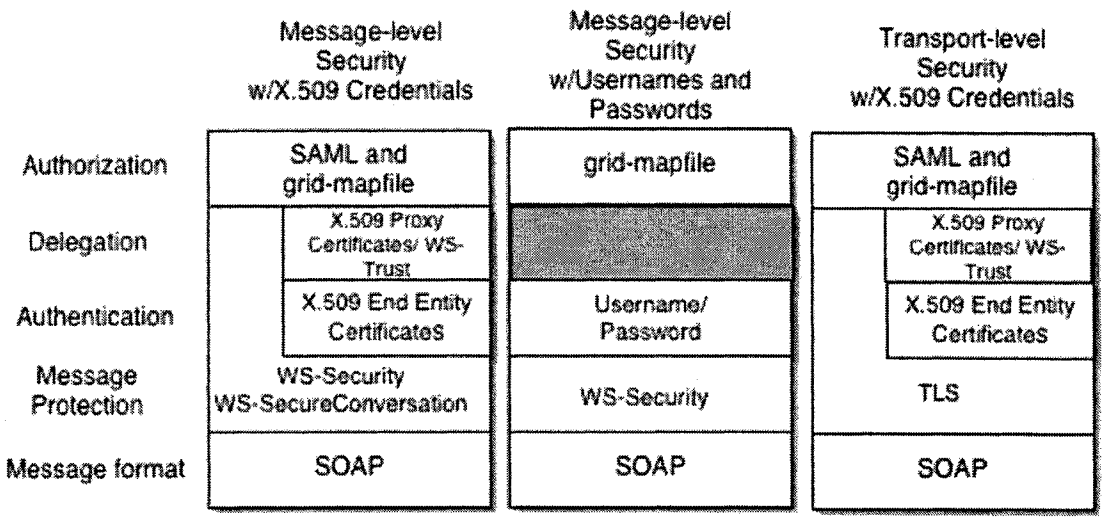


Figure 1: Security layout of the Grid Security Infrastructure (GSI)

3 Literature Review and Related Work

In this chapter we review the relevant research literature. Our focus is on complete authorization systems and approaches that serve as a foundation and comparative basis for developing an augmented system as described later in this thesis. We conclude the chapter with a critique of current problems and limitations of existing authorization systems and approaches and, thereby, provide a foundation for the approach described later in the thesis.

3.1 Authorization Systems

Several systems have been proposed to accomplish authorization in distributed environments. Though all of these systems share at least something in common to our system, they were designed with different goals in mind. Thus, our system is intended to augment other existing systems, not replace them entirely.

3.1.1 Akenti (1999)

The Akenti system was developed under a grant from the Department of Defense (DOE) at Lawrence Berkeley National Laboratories by Mary Thompson, William Johnston, Srilekha Mudumbai, Gary Hoo, Keith Jackson, and Abdelilah Essiari. Akenti is an authorization system which supports "multiple, independent and geographically dispersed stakeholders" [Thompson1999]. These stakeholders state their access requirements in certificates known as use-condition certificates and define those trusted to confirm the user attributes in question [Thompson1999]. A policy engine will then collect all the relevant certificates and make an access decision based on whether or not the user

satisfies all the requirements. Once Akenti has all the necessary certificates, it returns control of the access decision to the server (resource) [Thompson1999]. The resource server then acts on that decision to allow/deny access to the resource on behalf of the client. Akenti is a PKI based system, and thus to operate within Akenti, a user must have an X.509 public key certificate (PKC) at authentication time. Further, "Akenti policies are hierarchical and distributed between proprietary Policy Certificates and Use-Condition Certificates" [Thompson1999]. The main focus of Akenti is classical discretionary access control lists. Akenti always replies with a Capability Certificate, that states what a user is authorized to do [Thompson1999].

3.1.2 Certificate Based Authorization Simulation System (2001)

CBASS is a system which was developed by Jie Dai and Jim Alves-Foss from the Center for Secure and Dependable Software at the University of Idaho. Within CBASS, an AC is used for both authentication and authorization. A user's certificate is the vehicle chosen to carry policies and capabilities, specified via java/prolog [Dai2001]. Note that there are two types of certificates (policy and credential). CBASS also supports anonymity and Discretionary Access Control lists (DAC) [Dai2001].

The client, better known as the requester initiates a resource access request by asking the server (responder) for a credential template which indicates the necessary credential certificates to authorize the request [Dai2001]. After the template has been received, the client performs certificate discovery, including a local search of its own certificate database [Dai2001]. This is followed by credential retrieval from other external objects (e.g. user object), and finally the credentials are presented to the server along with the

request [Dai2001]. The server verifies the credentials against its policies and decides whether to acknowledge permission or to deny the request.

3.1.3 MAFTIA1 (2001)

MAFTIA was developed in France at LAAS-CNRS by Noreddine Abghour, Yves Deswarte, Vincent Nicomette, and David Powell. It supports an authorization scheme that can grant each user the appropriate access rights, while only distributing the credentials and information needed to execute its own task [Abghour2001]. MAFTIA employs an authorization server for the granting or denying rights. If a complex operation is authorized, the server distributes the necessary credentials/capabilities for all the individual basic operations that are needed to carry it out [Abghour2001]. On each host, a security kernel performs the necessary tasks for fine-grain authorization. To ensure that the security kernels on off-the-shelf computers connected to the Internet are adequately protected, critical parts of the security kernel will be implemented on a Java Smart Card [Abghour2001].

3.1.4 CAS (GT4) (2002)

The Community Authorization Service (CAS) is a trusted third party server that is responsible for managing the various necessary policies which govern access to a community's (Virtual Organization, VO) set of shared resources [Welch2002]. It was developed in collaboration between various institutions. The principal participants include Laura Pearlman, Von Welch, Ian Foster, Carl Kesselman, and Steven Tuecke. The CAS server contains individual entries for Certificate Authorities, users, servers and resources sites that combine into groups to form a community [Welch2002]. It also maintains policy statements which define which user or group has the permission, which

resource or resource group that specific permission is granted on, and what rights that permission entitles the user or group to [Pearlman2003]. CAS employs the notion of restricted proxies, which are built from a credential format that is designed to be neutral to the actual policy language employed [Pearlman2003]. An access request is made to the CAS server and if granted, a proxy is returned which contains the appropriate permissions to access the resource in question. Note that CAS can support various arbitrary policy languages such as Controlled English [Bacon2001], ASL [Jajodia1997], or Ponder [Darnianou2001].

3.1.5 KEYNOTE (1999)

Keynote was developed by Matt Blaze, Joan Feigenbaum, and Angelos D. Keromytis at AT&T Labs in 1999. Keynote in three words is a Trust Management System. It uses credential assertions to denote what actions can be done and who may perform them [Blaze1999]. Signature verification is build into the system. The assertions are based on human readable syntax [Blaze1999]. Keynote uses a very simple notation for specifying policy and credentials. Each trusted action is described by a simple attribute/value pair [Blaze1999]. Each application can define its own set of attributes which require application specific credentials to operate on them [Blaze1999].

3.1.6 PERMIS (2002)

Permis is a Java based authorization system developed by David Chadwick and Sassa Otenko at the University of Kent in the UK. It consists of a custom language to define actions, users, policies, and credentials as well as a compliance checker which is invoked as a Java object at a resource sites gateway [Otenko2003]. The credentials in Permis are

built in accordance to the latest X.509 standard [Otenko2003]. Permis leaves the mechanisms for authentication up to the application. The Permis policies are written in XML and are contained within one X.509 Attribute Certificate (AC) [Otenko2003]. The system also has implemented role based access controls which recognize separate hierarchies for authentication (CAs) and authorization (stakeholders) [Otenko2003]. Once the appropriate credentials and policies are retrieved, the compliance checker makes an access decision which is a simple yes or no. Unfortunately, Permis has no ability to return Capability Certificates [Otenko2003].

3.1.7 CARDEA (2003)

CARDEA is a product of Rebekah Lepro at NASA's Advances Supercomputing Division. It is a system which dynamically evaluates relevant characteristics of the resource and requester when processing all requests for access instead of considering specific local identities (not unlike grid-map) [Lepro2003]. Shared resources within an administrative domain are protected by local resource access control policies. All users are identified by X.509 proxy certificates and are then modeled according to individual characteristics [Lepro2003]. The necessary information needed to complete a decision is determined and collected during the decision process [Lepro2003]. Once all necessary information has been collected, it is presented to the policy decision point (PDP) for an access decision [Lepro2003]. From a conceptual point of view, Cardea "contains a SAML Policy Decision Point (SAML PDP), one or more Attribute Authorities (AA), one or more Policy Enforcement Points (PEP), one or more references to an Information Service (IAS), an XACML context handler, one or more XACML Policy Administration Points (PAP) and an XACML Policy Decision Point (XACML PDP)" [Lepro2003].

3.1.8 FIDELIS (2003)

Fidelis was developed by Walt Teh-Ming Yao for a PhD. at the University of Cambridge in 2003. It is a decentralized framework for trust management [Yao2003]. It specifies the necessary mechanisms for defining and managing trust information. It defines principals and policies which define the "who" and the "what" of a particular access decision [Yao2003]. Trust conveyance is the major concept within Fidelis, and is used to specify what a specific principal (subject) is trusted to do and who trusts the principal to do the specified actions [Yaoo2003]. This trust is specified using custom policy language statements. Fidelis also makes use of modified x.509 certificates which contain the required attributes. Fidelis is a web service platform which enables communication between complete strangers via third-party assertions and Role Based Access Control (RBAC) [Yao2003].

3.1.9 HP HSA (2003)

This paper presents a distributed authorization model which is appropriate in a web service scenario where multiple stakeholders are involved in performing a particular action [Mont2003]. This model uses a third party authorization service to compare specific credentials with a set of authorization policies [Mont2003]. The authorization model is built in such a way to enable authorization policies to be checked and built as a component of the web service composition process [Mont2003]. The major factor which makes this architecture different from others is its reliance on a hardware security appliance to perform the necessary security checks [Mont2003].

3.1.10 CONTEXT SENSITIVE (2003)

Context Sensitive authorization is a Role-Based Access Control (RBAC) form of authorization. In this system, authorization is derived dynamically within specific contexts [Mostéfaoui2003]. More specifically, context-based security tries to adapt the security policy according to relevant information which has been collected from the dynamic runtime environment [Mostéfaoui2003]. As the context of a particular action changes, so may the authorization decision. Users which operate under this type of authorization are consequently mapped to roles based on their credentials and associated competencies [Mostefaoui2003].

3.1.11 TERA (2004)

Trust Enhanced Role Assignment (TERA) is an authorization system developed by several individuals at Purdue University in 2004. The main principals include Bharat Bhargava and Leszek Lilien. TERA is a Java framework which is based on uncertain evidence and trust which is determined dynamically [Bhargava20004]. TERA evaluates the trust of a user directly based on their behaviors. The system will then decide "whether a user should be authorized for an operation based on the policies, the evidence, and the degree of trust" [Bhargava2004]. The reliability of the evidence is derived from the trust of the evidence provider [Bhargava2004]. A third party service, known as a reputation server, is responsible for managing the trust information. TERA makes use of two algorithms, which have been developed to determine a user's trust value based on a sequence of the user's past behaviours and interactions [Bhargava2004].

3.1.12 IBM TE (2004)

IBM Trust Establishment (TE) is a Java based, trust management tool which can be used to create trust relationships between strangers using their PKC [Ibm2004]. Rather than local account mapping based on a unique id, TE can determine a user's role(s) by their X.509 certificate [Ibm2004]. Though X.509 certificates are the most commonly used certificates, other certificate types are also supported. Note that this system is not responsible for making any access decisions, instead it provides user to group/role mappings, based on PK [Ibm2004].

3.1.13 Shibboleth (2004)

Shibboleth is a system which is intended to properly secure access to web-based resources under the Single Sign On (SSO) paradigm of access control [Morgan2004]. It has been designed in such a way to allow various federations and collaborations to extend their identity management systems (and PKI) to work together in partnerships, while always protecting the privacy of user's at individual sites [Morgan2004]. This scalable architecture predominantly performs attribute based authorization using the Security Associations Markup Language (SAML). To accomplish these tasks, Shibboleth employs both a Shibboleth Identity Provider (IdP) and a Shibboleth Service Provider (SP) [Morgan2004]. Typically a user will sign on to their local IdP which authenticates the user and returns a SAML assertions stating that the user has logged in. This assertion can then be sent to other sites within the collaboration to enable access without repeating the authentication step [Morgan2004].

3.1.14 VOMS (2004)

VOMS is an authorization system which is similar in nature to CAS. It provides an authorization capability in the context of Virtual Organizations. It is composed of several parts which include a user server (manages and returns information about users, including their certificates), the user client which accepts a user's certificate and returns a list of groups, roles, and capabilities, the administration client which enables management of the overall system, and the administration server which handles the requests from the administration client **[Alfieri2003].** Authorization requests are accompanied by the appropriate capabilities which will enable access to a specific resource.

3.1.15 PRIMA (2004)

The Privilege Management (PRIMA) system was developed as a PhD. thesis by Markus Lorch from Virginia Polytechnic Institute and State University in 2004. The Prima model consists of the necessary components to define a fine-grained authorization framework for distributed systems [Lorch2004]. These components include "a high-level privilege management model, an incremental user-centric trust model, and a dynamic model for authorization and enforcement" [Lorch2004]. Within Prima, all access rights are granted in the form of a privilege, which are secured using the proper mechanisms to prevent theft/tampering. These privileges are specified in a platform-independent policy language, with a limited lifetime, and can be delegated to other users [Lorch2004].

3.1.16 WS AA Authorization Framework (GT4) (2006)

The Authorization framework in Version 4 of the Globus Toolkit (GT4) provides many essential functions. This Java based framework has been designed in such a way that the actual method of authorization is abstracted, which allows authorization to be

accomplished by several different mechanisms [Globus2005]. These authorization methods include a module which supports the typical gridmap based authorization, access control lists, SAML assertions, and others [Globus2005]. The AA framework also supports delegation.

3.1.17 General System Comparison

There are other systems which also address some aspects of authorization in distributed environments. Any system which makes use of the Globus Toolkit (GTK) makes use of a grid-map file and Security Associations Markup Language (SAML) assertions in varying degrees, for authorization [Globus2005]. Such systems require accounts to be created prior to job execution. Though the possibility of a set of local generic accounts for unknown entities would solve this problem, it is undesirable from several viewpoints, including accounting. The Akenti Authorization System provides a mechanism for a finer degree of specification of authorization rights from multiple owners [Thompson2003]. Shibboleth, a SAML-based authorization framework, provides a mechanism for combining attribute information of known entities from multiple resource sites for the purpose of an access decision [Morgan2004]. VOMS and CAS have been developed to provide authorization on the basis of Virtual Organization (VO) membership, which enhances scalability, but still requires the manual configuration of trust relationships and VO memberships [Welch2002, Pearlman2003, and Cecchini2004] respectively. IBM Trust Establishment (TE) uses Role Based Access Control (RBAC) to provide authorization on roles rather than individual users [Ibm2004]. This is more scalable, but arguably, still not scalable enough. Several other systems were also researched and compared to ensure our system did not duplicate the functionality of an existing system.

3.2 Methods of Authorization

All architectures considered in this thesis research make a significant contribution to the problem at hand. Further, the culmination of these other systems could achieve the end goal of functional authorization with little to no administrative overhead in a scalable manner. It is the goal of the AASUR architecture not to combine all features of these other systems; rather, to combine the necessary aspects to achieve the desired end. All systems that were compared were considered from several points of view. These include:

1. Authorization Type (C1)
2. Architectural Features (C2)
3. Implementation Details (C3)
4. Arbitrary Entity Support (C4)

3.3 Comparison 1: Authorization

From Figure 2 we see that the majority of the systems tested involved some type of third party assertions. This means that some type of pre-established trust relationships must exist. Also, we can see that Attribute Certificates (AC) and/or capabilities are commonly used to specify what rights a particular user has [Opplinger2000]. The problem with the AC/Capability is that the access specifiers must exist within the AC or capability, prior to a job request. These specifiers may also take the form of access tickets which have been granted to a user or have been delegated to another user. These specifiers must be associated with sites that we have already established trust with. The access level is not determined dynamically. Within many of these systems, RBAC is employed as it is within AASUR, to reduce the overall administrative load.

The most important aspect of this table is the reputation column. Notice that only one other system makes use of some type of reputation to perform the access decision. Please note that although several of thee system may operate within the context of Globus and the notion of map files, the do not rely on map files natively, unless specified in the Figure 2.

	REPUTATION	ASSERTIONS	MAP FILE	ATTRIBUTE CERTIFICATES	CAPABILITIES	RBAC
AKENTI - 2000		X		X	X	
CBASS - 2001				X	X	
MAFTIA1 - 2001		X			X	
CAS (GT4) - 2002					X	X
KEYNOTE - 2002		X			X	
PERMIS - 2002				X		X
CARDEA - 2003		X		X	X	
FIDELIS - 2003		X				X
HP HSA - 2003		X		X		
CONTEXT SENSITIVE- 2003						X
TERA - 2004	X	X				X
IBM TE - 2004						X
SHIBBOLETH - 2004		X				X
VOMS - 2004				X		X
PRIMA - 2004				X	X	
WS AA AUTHZ (GT4) - 2006		X	X			X

Figure 2: Authorization Comparison

3.4 Comparison 2: Features

Figure 3 illustrates various features including what type of policy languages the system supports to define access rights to its resources, whether or not the system operates in the context of a public key infrastructure (PKI), and whether the system supports multiple stakeholders, anonymity, and delegation. Several of these features are considered essential in this type of environment, while others are simply features albeit very useful ones.

From Figure 3 we see that almost every system is PKI based, and that most also support delegation. Delegation is useful for sharing access rights to trusted users, but it is not

included within AASUR at this stage of design/development. Note, however, that it could be included in the future. We note that there is a wide range of policy languages which are supported by the various systems. More than half of the systems make use of a custom policy language which is a detriment to interoperability. This means that every site must now use the specific custom policy language in order to interact. A couple of systems, including AASUR, are completely policy language independent, meaning access policy specifications can be presented in different languages at each site. This is due to the fact that only the site which makes the access decision and generates the access ticket will have to understand it. Finally, anonymity is the least important attribute in this table. Though it has a place in some use cases, typically we want each user to be held accountable for their actions; knowing who they are helps to ensure that no malicious actions take place. It also provides a way of enforcing certain actions if any rules are broken. Most systems do not support anonymity. Note that both FPL and TPL are custom policy specification language developed specifically for Fidelis and IBM TE respectively.

	ACCESS POL. LANG.	PKI	MULTIPLE STAKEHOLDERS	ANONYMITY	DELEGATION
AKENTI - 2000	CUSTOM	YES, X509	X		
CBASS - 2001	PROLOG	YES, CUSTOM		X	X
MAFTIA1 - 2001	CUSTOM	YES, CUSTOM		X	X
CAS (GT4) - 2002	INDEPENDENT	YES, X509			X
KEYNOTE - 2002	CUSTOM	YES, X509			X
PERMIS - 2002	CUSTOM	YES, INDEPENDENT	X		X
CARDEA - 2003	XACML	YES, X509			
FIDELIS - 2003	FPL	YES, CUSTOM		X	X
HP HSA - 2003	CUSTOM	YES, X509	X		
CONTEXT SENSITIVE- 2003	CUSTOM	NO IMPL			
TERA - 2004	CUSTOM	YES, X509			X
IBM TE - 2004	TPL	YES, X509			
SHIBBOLETH - 2004	INDEPENDENT	YES, X509			X, limited
VOMS - 2004	INDEPENDENT	YES, X509, via GT4			X, via globus
PRIMA - 2004	CUSTOM	YES, X509	X		X
WS AA AUTHZ (GT4) - 2006	CUSTOM	YES, X509			X

Figure 3: Feature Comparison

3.5 Comparison 3: Implementation

The implementation comparison involved which language was used (if implemented at all), whether or not the system was web service based, and whether or not it relied on any type of external servers other than Certificate Authorities (CA) (as nearly almost every system needs at least one CA). Figure 4 illustrates this comparison appropriately. Most systems are implemented in Java, due to its security model and web capabilities. The performance of Java has been under attack by many, but it is in fact very close to C++ in many ways. Note that nearly all systems need some type of external server which will increase the overall message transfer time requirements, thus hindering performance. Finally, due to the shift in design paradigms towards web services, most of these systems have been implemented in such a way which exploits web services.

	JAVA	C/C++	OTHER LANGUAGE	THEORY / NOT IMPL.	SERVICE BASED	EXTERNAL SERVERS
AKENTI - 2000		X				X
CBASS - 2001	X					
MAFTIA1 - 2001	X					X
CAS (GT4) - 2002	X	X			X	X
KEYNOTE - 2002		X				
PERMIS - 2002	X					X
CARDEA - 2003	X				X	X
FIDELIS - 2003			X		X	X
HP HSA - 2003			X		X	X
CONTEXT SENSITIVE- 2003				X		X
TERA - 2004	X					X
IBM TE - 2004	X					
SHIBBOLETH - 2004			X			X
VOMS - 2004	X	X			X	X
PRIMA - 2004	X					
WS AA AUTHZ (GT4) - 2006		X			X	

Figure 4: Implementation Comparison

3.6 Comparison 4: Arbitrary Entities

The meaning of arbitrary entity support is straightforward. Supporting arbitrary users means that the system has the capability to allow an unknown user (that is, with no prior knowledge of the user) potentially to use the resources at a specific site. There is,

however, a caveat (in general and specifically directed to Figure 5). Technically, this includes knowledge of a user both directly and indirectly. If a user is trusted (but not known) through another site, there is still a requirement for pre-established trust. The best way to ultimately make a system scalable from an administrative overhead point of view is to ensure that a site does NOT require ANY direct or indirect knowledge of a user. In that scenario, the reputation is the only contributing factor to the access decision.

	ARBITRARY ENTITIES
AKENTI - 2000	X
CBASS - 2001	X
MAFTIA1 - 2001	X
CAS (GT4) - 2002	
KEYNOTE - 2002	X
PERMIS - 2002	
CARDEA - 2003	X
FIDELIS - 2003	X
HP HSA - 2003	
CONTEXT SENSITIVE- 2003	X
TERA - 2004	X
IBM TE - 2004	X
SHIBBOLETH - 2004	X
VOMS - 2004	
PRIMA - 2004	
WS AA AUTHZ (GT4) - 2006	

Figure 5: Arbitrary Entity Support

3.7 Authorization System Comparison Summary

As with any new development many issues have come to light. Most have been very minor, but one major issue came up. The solution to the main issue involved striking a balance between the amount of information kept which will severely affect the certificate size and consequently the scalability, and the flexibility available to the individual

resource sites to determine who may have access to their resources based on their past actions. In the end, we have gone with a minimalist approach to ensure unlimited scale.

The closest match to AASUR is the TERA (Trust-Enhanced Role Assignment) system [Bhargava2002]. Though TERA encapsulates ideas, similar to AASUR, it has several key differences. First, the reliability of the evidence is based on the trust of an evidence provider [Bhargava2002]. This implies that evidence may come from a third-party source and be potentially less reliable. Our system maintains a globally consistent representation of an entity's past, which allows a resource site to make an access decision based on a local interpretation. Secondly, TERA relies on a separate reputation server to manage user reputation [Bhargava2002], whereas we store the actions an entity has performed in its PKC, managed by a Certificate Authority (CA). In AASUR, the reputation is calculated using local processing power to maintain scalability. Finally, whereas our architecture has been developed as a web service, TERA is an application based system.

In this chapter we have critiqued several approaches and systems for authorization presented in the literature. In the next chapter we introduce a system that augments the important features of several of these systems.

4 Augmented Authorization System Using Reputation

In this section will give a detailed description about our system, the Augmented Authorization System Using Reputation (AASUR). At the most general level, the main contribution of this architecture is the notion of providing authorization on the basis of points which are collected in a standardized way, and which in turn may be used by individual resources to determine reputation. The only guarantee that this architecture makes is that points will be collected consistently for accepted actions. Any decision that results in a breach at a resource site is completely the responsibility of the resource site that misinterpreted the points presented at the time of resource request. There is no requirement to obtain or analyze any assertions from any third party though, in the future, this capability could be exploited to enhance this system further. From an administrative point of view, there is no need to establish anything prior to a job request/allowance. Though these features may exist in other systems individually, this combination, and consequently the overall ability of the system, does not exist anywhere else to the best of our knowledge.

4.1 Justification

The reason for this architecture is simple. If the current notion of the "grid" is to reach a level where the masses are able to use/sell spare cycles on a global scale (not unlike selling power back to the grid) then minimally, from an administrative point of view, changes needed to occur. Scalability in current systems is, without a doubt, a hindrance to this goal. Of course, this architecture (AASUR) is not the end solution, but it is a step in the right direction.

4.2 Overall Architectural Comparison with AASUR

Although all of these previous comparisons are valuable in and of themselves, an active comparison of the most important characteristics which AASUR possesses is also needed. This can be found in Figure 6. This table takes the most important aspects of the AASUR system, and shows how the other systems compare to it. Although many systems show some similarities in various categories, none provide the complete set which AASUR does, including support for reputation. The details regarding the inner workings of AASUR will be dealt with next in Chapter 5.

	RepBAC	RBAC	FLEX POLICY	PKI	JAVA	SERVICE BASED	EXTERNAL SERVERS
AASUR - 2006	X	X	INDEPENDENT	X509	X	X	
CAS (GT4) - 2002		X	INDEPENDENT	X509	X	X	X
PERMIS - 2002		X	CUSTOM	YES, IND.	X		X
CARDEA - 2003			XACML	X509	X	X	X
FIDELIS - 2003		X	FPL	YES, CUSTOM		X	X
TERA - 2004	X	X	CUSTOM	X509	X		X
SHIBBOLETH - 2004		X	INDEPENDENT	X509			X
VOMS - 2004		X	INDEPENDENT	X509, via GT4	X	X	X

Figure 6: Overall Architectural Comparison

4.3 Problem Statement

We have introduced existing tools, components, and systems which are used as parts of complete authorization systems. We have also discussed the various systems which have been created to solve problems with the current state of authorization. Further, we have defined the major characteristics of our system, AASUR, and provided a comparison of it alongside the other existing systems. We will now define clearly what the current problem is and how we intend to overcome it.

The major problems which exist in current systems include the following:

1. The requirement for pre-established trust among user-to-resource and resource-to-resource (in the case of delegation of credentials), is not feasible due to the administration overhead.
2. Further, scalability in general due to the pre-established trust, is significantly hindered, which results in limited scale for any architecture which mandates pre-established trust.
3. Many existing systems require the use of a particular policy language which creates the need for every site to use a specific policy language.
4. Though many systems are Web Services based, some are not. Evolution in current distributed computing certainly requires an architecture which is based on Web Services.

With this background information, and the comparison of exiting systems, we state the following:

> *The current requirement for pre-established trust, the lack of consensus for the use of web services, and the specific requirements for policy languages are a detriment to the overall adoption of the various authorization schemas in existence. This is not only due to the significant overhead, but also due to the lack of flexibility inherent in existing systems. We present a solution to this problem using the AASUR architecture, which will achieve what other systems have not. We accomplish this through the use of reputation metrics which allow individual resource sites to make authorization decisions without having any prior knowledge of a particular user. AASUR will also be completely independent of*

resource access policy language thereby providing much flexibility to individual resource sites.

Through various simulations and consequent numerical results we show that AASUR contends well in distributed authorization from a performance point of view and also from an administrative overhead point of view. We show also that AASUR does in fact solve many of the problems facing existing authorization systems.

5 AASUR

AASUR was developed to solve the problem discussed throughout Chapters 3 and 4. It is written in Java and its functionality is exposed as web services described by WSDL documents. All messages and formats are XML based. It can be deployed to any application hosting environment, but must be used alongside EJBCA, an Enterprise Java Bean Certificate Authority. They (AASUR + EJBCA) have all of the necessary components to retrieve certificates, request resource access, perform authorization decisions, collect job actions, and update a user's certificate to reflect newly performed actions. Of course, there are several components which are needed to support these critical tasks as well. All of this will be discussed in detail further on in this paper.

5.1 Design Context

The context which this architecture was developed within is very straightforward. The major goal was to develop a system in which automated discovery and authorization could occur. Existing systems provide other aspects of automated discovery. In fact other systems provide automated authorization, but in the context of pre-established trust and local accounts with rights setup prior to authorization/execution. The AASUR system has been developed within the context of automated authorization without any pre-established trust, and very minimal administrative overhead at any participating site. The goal of AASUR was to provide not only automated authorization, but also automated collection of a user's history, which would allow for this authorization to take place with almost no user interaction and zero pre-established trust.

5.2 Design Methodology

The methodology that was followed was simple. It started with isolation of specific characteristics within current systems which would prevent the goals of this architecture from being realized. Several issues were very quick to surface, and are as follows. First, the requirement for existing accounts to exist which provide a mapping to locally defined rights, although functional, were not conducive of an architecture which could scale easily. Second, although a possible solution to the previously mentioned issue, the notion of shared accounts at resource sites does increase scalability, but eliminates any possibility of maintaining an individuals actions or providing any kind of accounting or non-repudiation. Third, though receiving assertions from other trusted sources enables a site to use some combinational logic to determine a best action, it is still subjective to some to degree. This may require some clarification, as this (AASUR) approach also is subjective in some ways. The major difference is that this architecture is subjective with respect to the resource sites decision for access, not the collection or portrayal of a user's reputation. Finally, many systems lack flexibility when it comes to the decision of whether or not access will be granted. This should always be left completely up to the resource owner.

The goals then of AASUR are the converse of these issues found within distributed authorization systems. Once these initial goals had been established it was time to begin designing the system. From the outset we wanted to develop a set of metrics that would allow the storage of a fairly detailed history which could be used to derive a user's reputation. We developed a core set of 27 point categories which, in our opinion, cover

what a user could potentially do (both good and bad) during the execution of a job. The specific details regarding these categories will be discussed in the Section 5.4.

Once the point categories were determined, we began designing how and what else would be stored. Always, the maintenance of the highest degree of resource site flexibility was constant. For example, we wanted an individual resource site to be able to deny a user because that user had performed a Buffer Overflow (BOF) at Site X on May 3, 2007 at 6:03 pm.

Our initial format to represent a user's reputation is as follows:

```
<? xml version='1.0' encoding='ISO-8859-1' ?>
<reputation>
    <collection type='LFC'>
        <site sha1='A123A3E4B32CD3E2145CC32DDEF4A123A3E4B32C'>
            <date>
                1158654532123
            </date>
            <date>
                1156785432345
            </date>
        </site>
        <site sha1='45CC32DDEF4A123A3E4B32CD3E21F678A3B5C332'>
            <date>
                1165434567897
            </date>
            <date>
                1124567543567
            </date>
        </site>
    </collection>
    <collection type='LMO'>
        <site sha1='D3E21F678A3B5C33245CC32DDEF4A123A3E4B32C'>
            <date>
                4532245678878
            </date>
            <date>
```

```
                    2232357889764
                </date>
            </site>
        </collection>
    </reputation>
```

This format allows for several things. First, everything is broken down into point categories, so that the act of summing them to make the necessary calculations is more efficient. Second, within each category the occurrences are divided up by location so that if a user has ever been involved with a specific resource site, the points earned from those interactions can be quickly ignored. Third, the absolute date/time in milliseconds has been recorded to allow for maximum flexibility with respect to filtering of actions based on date and time. There was one caveat however, to this design. Due to the detail which was kept in this format, the overall scalability could be an issue. This was not only because an X.509 certificate was not meant to store a lot of extra information, but simply because of the scale of information that would be involved.

The initial testing was strictly intended to determine how scalable this particular setup would be. This involved creating one certificate (to represent an individual user) and begin running some simulated jobs to see how a certificate would handle the reputation. The job simulator, which is discussed in detail later (5.7.9), has the ability to configure the percentage of good actions which are generated as well as the percentage of bad actions generated. Initial testing involved a 90/10 split (good/bad). I generated 100 actions (corresponding to random point categories) for each job. I decided to make the first run consist of 1000 jobs, for a total of 100,000 actions, including a date for each one and the resource site it occurred at. Very quickly (approximately 15 jobs, 1500 actions)

the code threw an exception. The certificate had reached 45K and the CA could no longer accommodate it. Although not that large, it is much larger than it was designed to be. This is aside from the performance hit that would have been experienced had a certificate of this size, and larger, worked without error in the context of this CA. A larger certificate would affect not only the network transfer times, but also the available storage space on the CA.

Obviously this format consumed far too much space, so an alternative would have to be developed. The important question then is, could be done without losing any of the flexibility for the resource site. Further testing revealed that 1000 site signatures amounted to approximately 50K, which equates to the previously determined limit. This did not include the actual actions which are the most important aspect of reputation. We decided that since within typical globally distributed systems, a job may span more than 1000 sites, this level of information was simply not scalable or feasible.

The second design revision involved a long integer to represent each point category. Since the long data type in Java is 64 bits, each number has an unsigned maximum value of 18,446,744,073,709,551,615 (2^{64}). This design also included a sliding window of resource site signatures. These signatures would enable a resource site to see the last X sites a user had been involved with. Since 1000 site signatures were far too big, we determined that 250 was a good number to try. After a little testing, it seemed that this solved the problem, but raised another issue. Due to the scale of global grid computing, it is not unreasonable to say that a particular job could span more than 1000 sites. This, to

us, made the value of the last 250 sites a user has been involved with very minimal. We came to the conclusion that this extra information would simply waste valuable space in the certificate and mean longer certificate retrieval times, as well as longer transfer times between the CA and the resource site. With that, we removed the sliding window of resource sites altogether.

In the end, the format of the reputation is significantly different from the original version. We simply maintain a counter (64 bit) for each point category, along with some basic information such as most recent request, average job time, etc. Depending on the context, this is good and bad. The bad is that we loose some flexibility, but that type of detail is simply not scalable in this type of computing environment. The good is that an upper bound of the certificate size can be determined and it is relatively small. To analyze the maximum size, we must consider the average certificate size using only the standard information. In the case of EJBCA, which should not differ significantly from other Certificate Authorities, the initial certificate size with no non-standard extensions is 2-3K. There is a total 27 categories, each requiring a Java long integer. A long integer in Java is 64 bits (8 bytes). This gives us a total of 216 bytes. Add the certificate base size to this and we arrive at approximately 3.2K. It is not likely to ever reach 5K, but if we use that as an upper bound, the max size is still very small and is unlikely to cause any performance/scalability issues from a message transfer or certificate retrieval point of view.

5.3 Design Assumptions

Several assumptions were made prior to and during the design phase of AASUR. These assumptions were required in the context of the thesis, though some could be removed if further development was to take place.

The first assumption is that a middleware layer exists to provide the necessary mechanisms to collect all actions that occur during processing at a resource site. This assumption is rather large as, without it, the thesis will not function properly. Such a component would make use of various operating system specific hooks to pull various pieces of information such as read, writes, memory usage, disk usage, CPU time, etc. Along with this raw information, this component would need some type of interpreter to maintain what levels of access have been granted. This is needed so it will be possible to determine when these various access level thresholds have been crossed. Not only should this be able to determine when thresholds have been breached, but it must also be able to intelligently determine if gray area events, such as buffer overflows, have occurred.

The second design assumption also involves the mechanisms which are responsible for collecting user actions at individual resource sites. This code, which must be deployed at every participating resource sites on the grid, is expected to be contained within secure and signed binaries to prevent tampering and to ensure integrity.

Third, it is also assumed that this collection component signs each and every job action message destined for a CA with its public key (PK) so that we may ensure the integrity of

the message during transport. This stops any malicious resource sites from falsifying what actually happened during processing and sending these inaccurate job actions to a CA for addition to a user's reputation. In other words, this ensures that collection will be completely objective. Effectively meaning that, if any improper access decisions are made, it is strictly due to improper policy configuration at the resource site in question.

The fourth and final assumption involves global participation. Typically, the definition of an entity known as the "grid" (which has yet to exist) implicitly involves many, if not all, users across the globe. In fact, the goal of researchers is to create a system that is pervasively and unobtrusively embedded in the environment, completely connected, intuitive, effortlessly portable, and constantly available [Tang2004]. Thus, for this architecture to become as useful as it possibly can, it must be adopted and trusted on a global scale. That being said, this is not a complete requirement, just an assumption that was made prior to design. This system could be used within the context of any collaboration that exists under a common CA hierarchy and that wishes to share its resources in this automated fashion.

5.4 Reputation and Evidence

If an entity is to execute a job on foreign resources, the system must use some globally accepted metric to determine whether or not the entity can be trusted. Typically this metric, outside of the computing world, is the reputation of that entity. It seems reasonable that this notion of reputation can be used within the context of the computing realm as well.

To enable assessment of reputation we use the various points earned by a user at various resource sites. Points are acquired by tracking important actions performed by a user (e.g., overuse of resources) during job processing. This point base serves as evidence to be used by resource sites, along with local policy, to determine a particular user's reputation, and consequently, to make an access decision.

We have developed a set of actions which, we believe, directly pertain to the act of making an authorization decision. From a high-level view, the actions (called point categories), are divided into three main sections: Illegal/Legal Actions, Other Performed Point Actions, and Additional Information (Tables 1, 2, and 3 respectively) for a comprehensive listing of the various point categories. Please note that the current set of point categories may change as our system evolves. Please also note that although the illegal and legal categories contain the same actions, it is necessary to maintain both.

In AASUR, the act of performing one of the applicable actions is equivalent to earning one point which will be added to the associated point counter in the user's PKC. See 5.10.1 for an example of a user's reputation in our XML based format.

Reads/Attempts	Communication attempts
Writes/Attempts	Deletes/Creates
Execution/Process Spawn	Resource allocations
System commands	Service invocations
Device Access	Various I/O Operations

Table 1 – Legal/Illegal Point Actions

Forcibly terminated jobs	Buffer overflows
Overuse of Resource	Compilation Errors
Runtime errors	

Table 2 – Other Performed Point Actions

It is necessary to note that many of the multiple instance actions such as all of the actions listed in Table 1, as well as Denied/Permitted Job Requests (DJR/PJR) in Table 2, have a matching pair of positive and negative categories. Certain multiple instance categories however, do not. These categories include Buffer Overflows (BOF), Compilation Errors (CE), Runtime Errors (RTE), Resource Overuses (ROU), and Forcibly Terminated Jobs (FTJ). The reason is that these particular categories do not possess an opposite value, is simply due to the nature of their values, not because it was decided to not include them.

Although legal/illegal actions are very important, there may be additional information which a resource site may consider. Our architecture also makes use of this other numeric information. Like the illegal/legal numeric data, this numeric information is also transmitted in each entity's PKC. **Table 3** provides examples of this additional evidence.

First job request	First job completed
Most recent request	Most recent job completed
Average job time	Total Jobs
Country	Successful jobs
Permitted job requests	Denied job requests
Culture	Geographic location

Table 3 – Additional Evidence

Below, Figure 7 shows a complete representation of the structure of an AASUR Java based reputation object in our system.

Figure 7: AASUR Reputation Object

5.5 Architectural Assumptions

The layered design of OGSA permits straightforward integration of AASUR within existing and future systems as a web service [Foster2005]. This means that this authorization system could be used to provide the authorization capability to almost any system which supports web services and OGSA.

We assume the proper and secure use of web services including the necessary standard web service mechanisms such as SOAP and WSDL. This also includes ensuring that the application hosting environment is configured properly and securely according to associated specifications. Further, this requires the use of either the TLS suite of protocols or the WS-Security specification to ensure message protection and integrity. In

the case of this thesis, TLS has been selected as it handles everything for us at other layers automatically. With WS-Security, encoding and interpretation of keys must be manually dealt with and the key information must be sent along as payload and extracted at the receiving end [WSS2004].

We also assume that the Certificate Authority functionality may be accessed as SOAP based web services [W3C3]. In the case of EJBCA, this functionality was not available, and thus it had to be added.

5.6 Major Components and Packages

5.6.1 org.aasur.ejbca.ca.AassurCaService (ACS)

This is the major component which exists on each CA. It provides the system with many necessary functions. Not only does it act as an interface between a web service consumer and the CA, but it also performs several of the critical processing procedures needed by AASUR. There are four methods publicly exposed via the web from this service. Two of these methods are used to directly retrieve a certificate (one via serial number and one via username). Both of theses methods, findCertificateByUsername(username) and findCertificateBySerno(serno) return the certificate (if found) in the form of a byte array. The third method used for certificate retrieval is called findUsernameBySerno(serno). It returns a String representation of a username.

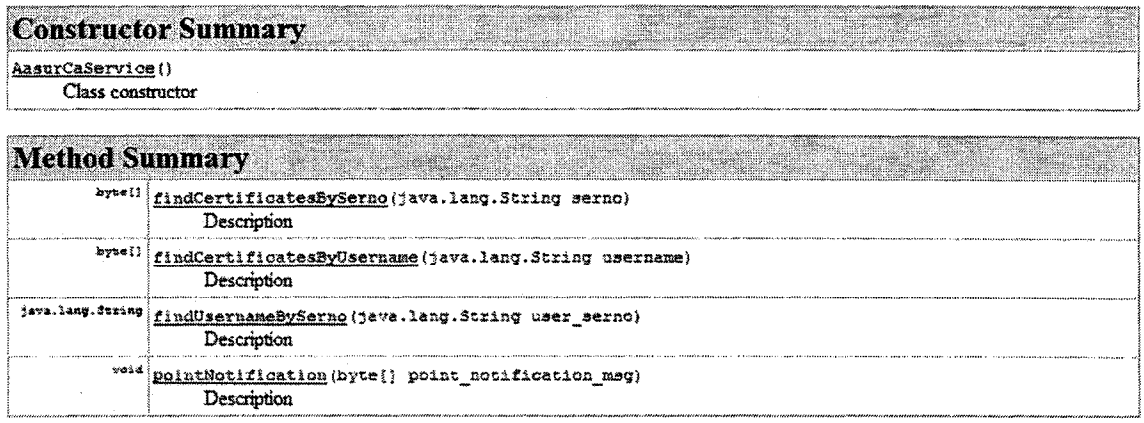
It was previously mentioned that one of these methods provides indirect retrieval capabilities. This is because findUsernameBySerno(serno) provides the capability to discover the mapping between a username and a serial number. The serial number can

then be used to retrieve a certificate. Multiple certificate retrieval methods exist for flexibility depending on the specific requirements of a given situation.

The fourth public method is called pointNotification(actions), and has the duty of accepting the point actions message (see 5.9.2) from a resource site, and performing all necessary processing. This processing will include the following steps:

1. retrieval of the user in questions certificate
2. parsing of the message with the assistance of a supporting class called org.aasur.common.XMLDomParser.parseXmlString(xml) (see 5.7.7)
3. instantiation of a new reputation object populated with the user's reputation with assistance of org.aasur.reputation.AasurReputation(certificate) (see 5.6.3)
4. a call to the merge function of org.aasur.reputation.AasurReputation called mergeNewActions(actions) (see 5.6.3) which combines the current reputation with the new actions list.
5. a call to an internal private method in ACS called generateCertificate(username, serno_string, password, reputation), which does what its name hints at, and stores the new certificate in the internal CA database. Since generateCertificate() is only used internally, it does not get exposed publicly.

Further details of the AasurCaService methods are shown in Figure 8.



Constructor Summary

AasurCaService()
Class constructor

Method Summary

byte[]	findCertificatesBySerno(java.lang.String serno) Description
byte[]	findCertificatesByUsername(java.lang.String username) Description
java.lang.String	findUsernameBySerno(java.lang.String user_serno) Description
void	pointNotification(byte[] point_notification_msg) Description

Figure 8: AasurCaService Object

5.6.2 org.aasur.resource.ResourceSiteServiceListener (RSSL)

The resource site service listener is the next major component in the AASUR system. This particular service resides on each site where a resource(s) will be made available to the public. RSSL may be on individual machines, but the most likely scenario involves an instance of it on the gateway. This class has only one public method, but there are also two additional private methods contained within for internal use. The major public method is called resourceRequest(certificate). It is this service which a user that is requesting some resource access will connect to when they want to submit their credentials to for an access decision. This service/method performs several duties which include:

1. certificate validation
2. extract the CN of the CA so we know which CA to contact with the reputation update
3. extract the reputation from the certificate
4. load Risk Factor (RF), User Class (UC), and Access Decision Engine (ADE) policies (discussed later)

5. call RF determination functions
6. call User Class determination functions
7. call ADE to make final authorization decision

If authorization is granted, this method will also perform additional functions, including:

1. job timing calculations
2. job action simulations
3. notification formatting
4. CA notification

Note that if authorization is not granted, there is still a need for notification formatting and a point notification message containing a Denied Job Request (DJR) (see Table 2).

As previously mentioned, this class has two private methods used internally. These are called formatPointNotification(actions, startTime, endTime, siteSignature) and generateDjrNotification(siteSignature). Their main function is to create/format the messages which will be sent to the parent CA of the requesting user. The second method is used when a request for resource access is denied, as a message is still required to be sent to the parent CA of the user to record a denied request for statistical purposes.

Further details of these methods are shown in Figure 9.

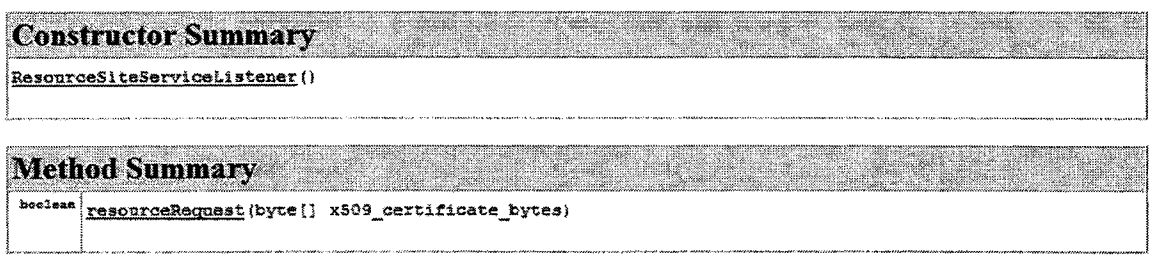


Figure 9: ResourceSiteServiceListener Object

5.6.3 org.aasur.reputation.AASURReputation (AR)

This object is the foundation for a Java based representation of a user's reputation. It provides the necessary functions to allow reputation manipulation, perusal, and merging as well as the actual act of parsing the XML into reputation objects. This class has two main methods and six supporting methods. It also has two constructors (Figure 9). All supporting methods are public (mostly involving get/set operations for private object variables). The major public method is mergeNewActions(actions) and the major private method is populatObjectWithReputation(reputationXml).

The mergeNewActions(actions) method is called (within the parent CA of the user which performed the processing) when a job has been completed and there are new actions which need to be added to the existing reputation. There is only one parameter to mergeNewActions which is a String representation of the XML based Point Notification message sent from the resource site. This method performs the following functions:

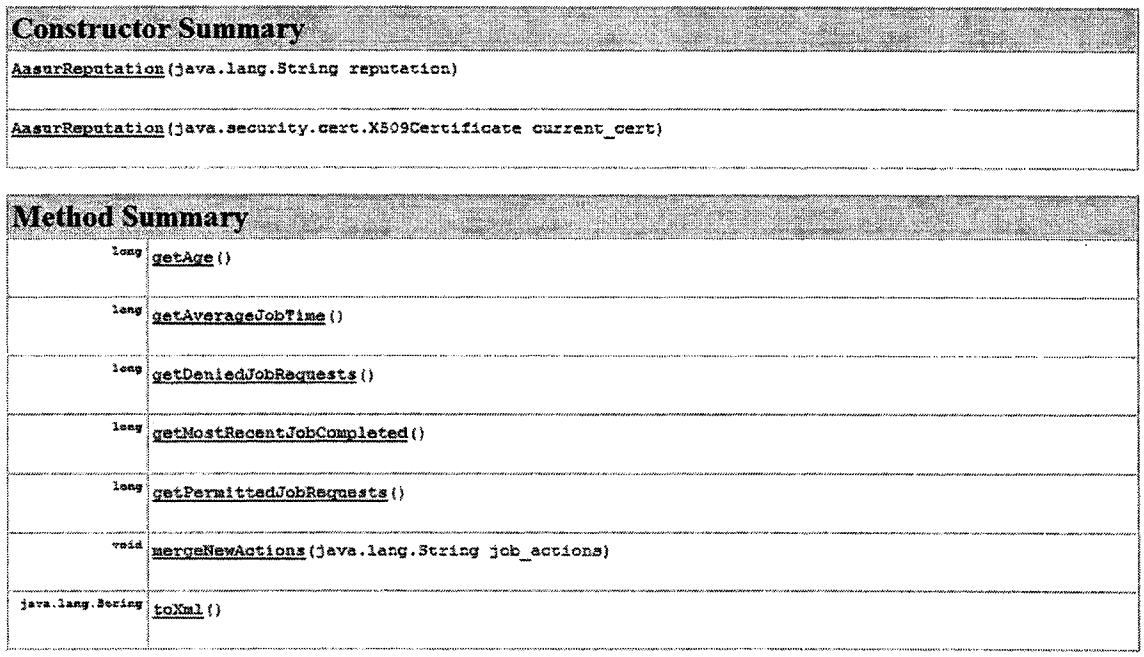
1. message parsing
2. average job time, first job/most recent job (requests/completions) calculations
3. incrementing the proper reputation counters to reflect new actions in the reputation

The other method, populateObjectWithReputation(reputation), also performs a very critical function. It is the responsibility of this method, which gets called from the AasurReputation(certificate) constructor upon instantiation of an AasurReputation object (and nowhere else), to create a local, Java based, representation of a user's reputation. It performs the following functions:

1. reputation extraction from certificate
2. parsing of the user's reputation
3. cycle through reputation, incrementing local variables (long) which are used to represent the various point categories when a specific type is encountered

The remaining methods, with the exception of toXml(), in the AasurReputation class are used to get/set private local objects from outside of the class. These can be used to retrieve age, average job time, denied job requests, permitted job requests, and most recent job completed. The remaining point categories are public by default. The toXml() method is used to format the current reputation for insertion into the user's certificate.

Further details of these methods are shown in Figure 10.



Constructor Summary

AasurReputation(java.lang.String reputation)
AasurReputation(java.security.cert.X509Certificate current_cert)

Method Summary

long	getAge()
long	getAverageJobTime()
long	getDeniedJobRequests()
long	getMostRecentJobCompleted()
long	getPermittedJobRequests()
void	mergeNewActions(java.lang.String job_actions)
java.lang.String	toXml()

Figure 10: AasurReputation Object

5.6.4 org.aasur.policy.UserClass (UC)

As The User Class is useful not only when a user has not performed any actions in the grid yet (authorization base case) but it also helps to augment the RF in the overall authorization picture. It makes use of additional user information (Table 3). This additional information belongs to the Java Object as local private variables and includes the following:

1. age
2. average job time
3. denied job requests
4. permitted job requests
5. most recent job completed

It is our opinion that these pieces of information are as valuable when making an authorization decision as the individual occurrences of the various point categories. That being said, the UserClass object is one that has a local placeholder for each one of these variables. For a given instance of UserClass, these local variables hold specific rules that define what range of values these variables may hold. These values define the requirements a user must meet in order to belong to a specific User Class. A User Class Policy is what contains one or more of these UserClass instances. An example will be provided to shed more light on this in 5.13. There are two constructors for this class, as defined in Figure 11 below. This class also contains the necessary get methods for these instance variables, as well as a toXml() method which provides a mapping from the Java UserClass object to one that is XML based to be inserted (as one of potentially many) into the UserClass policy file for long term storage.

Further details of these methods are shown in Figure 11.

Constructor Summary
UserClass()
UserClass(java.lang.String className, long age, long pjr, long djr, long ajt, long mrjc)

Method Summary	
long	getAge()
long	getAVGJobTimeMillis()
java.lang.String	getClassName()
long	getDeniedJobRequests()
long	getPermittedJobRequests()
long	getRecentCompletedMillis()
java.lang.String	toXml()

Figure 11: UserClass Object

5.6.5 org.aasur.policy.UserClassPolicy (UCP)

The UserClassPolicy class serves as the Java representation of the AASUR User Class Determination Policy. It is a place where multiple UserClass objects can be stored and accessed when an authorization decision is going to be made. During instantiation of this object, we pass a filename to the constructor which defines the location of the systems User Class policy file. This file is extracted and parsed, and the corresponding UserClass objects are created for each defined class. These objects are stored in the UserClassPolicy object instance within a local Vector. There are actually two constructors for this class, as defined in Figure 12.

This class has a public method called getClasses() which returns a Vector of all classes which have been defined in the system. The org.aasur.policy.UserClassEngine will look through these classes one by one and compare the values of the individual pieces of information with the values which are associated with the user in question. This class also has a toXml() method which produces a complete policy with all defined user classes that can be written to a file on the local file system for persistent storage. This is the same format that is parsed by the constructor of this class when a policy file is being read from the file system.

Further details of these methods are shown in Figure 12.

Constructor Summary
UserClassPolicy()
UserClassPolicy(java.lang.String policyFileName)

Method Summary	
java.util.Vector	getClasses()
java.lang.String	toXml()

Figure 12: UserClassPolicy Object

5.6.6 org.aasur.policy.UserClassEngine (UCE)

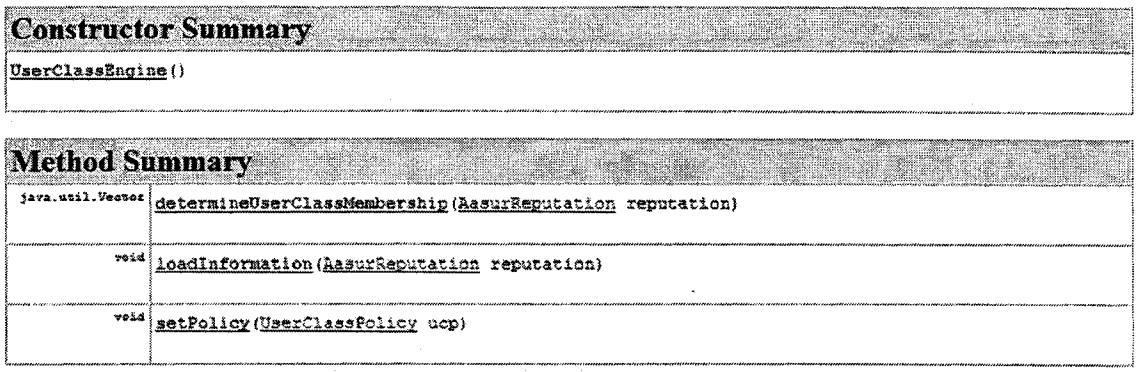
The UserClassEngine is what performs the necessary comparisons to determine which, if any, class the user in question belongs to. It has three methods, all of which are public and very important. The first of these methods is setPolicy(policy). This method takes a UserClassPolicy object as an argument and sets a local instance of UserClassPolicy to it. The second method is called loadInformation(reputation). LoadInformation accepts an

AssurReputation object as input and will extract the additional information (and assign to local variables) which we will use as a comparison to the various user classes to determine which classes a user is in. The final method in the UserClassEngine class is the determineUserClassMembership(reputation) method. This method looks at one UserClass from the policy at a time, comparing the individual values to the user's values. If each comparison passes then the user is a member of that specific class, and the engine will move on to the next defined class. Note that it is possible for a user to be in several user classes. When making the comparisons, the UserClassEngine operates under the following rules:

1. AGE – The user has been around for **at least** x milliseconds
2. AVG JOB TIME – A user's average job is **no longer than** x milliseconds
3. DENIED JOB REQUESTS – A user has **no more than** x denied job requests
4. PERMITTED JOB REQUESTS – A user has **at least** x permitted job requests
5. MOST RECENT COMPLETED – A user has completed a job **no longer than** x milliseconds ago

If all of these rules are met, then a user belongs to this class.

Further details of these methods are shown in Figure 13.



Constructor Summary

UserClassEngine()

Method Summary

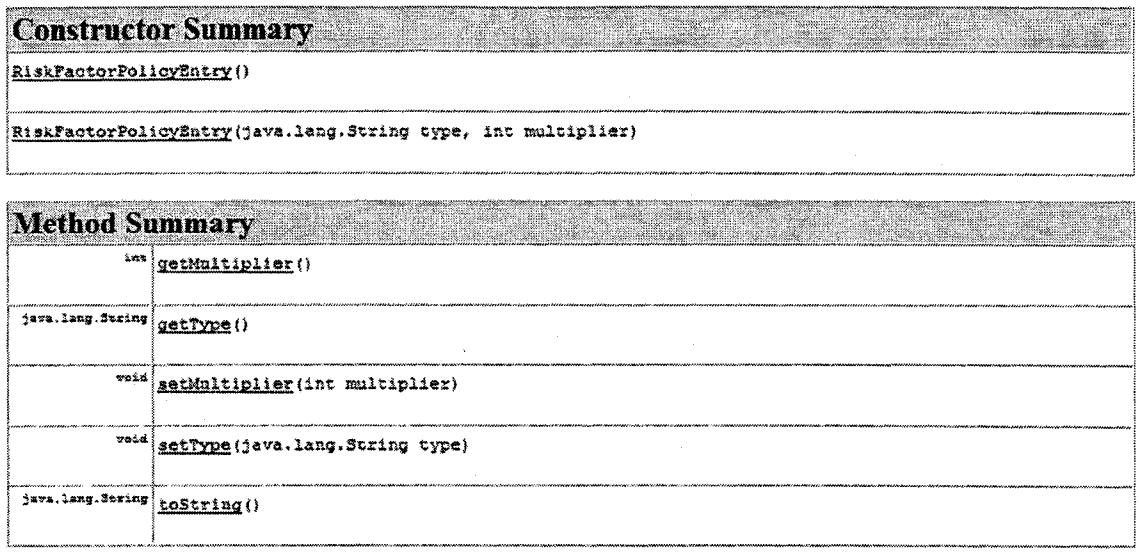
java.util.Vector	determineUserClassMembership(AasurReputation reputation)
void	loadInformation(AasurReputation reputation)
void	setPolicy(UserClassPolicy ucp)

Figure 13: UserClassEngine Object

5.6.7 org.aasur.policy.RiskFactorPolicyEntry (RFPE)

Very similar to the various user class entries which may exist within a UserClassPolicy object, the RiskFactorPolicyEntry is one of 27 entries that exist within a RiskFactorPolicy. Each entry corresponds to a particular point category. The design is deliberately general to facilitate the easy addition of new point categories, should the system need to be extended. Each entry contains a risk factor type and multiplier. Both of these values can be accessed and set with the corresponding public get/set methods which belong to this class. This class also has a toString() method which is only used for debugging purposes. Instantiation of this class can involve the default empty constructor or a constructor which takes a type and multiplier as input.

Further details of these methods are shown in Figure 14.



Constructor Summary
RiskFactorPolicyEntry()
RiskFactorPolicyEntry(java.lang.String type, int multiplier)

Method Summary	
int	getMultiplier()
java.lang.String	getType()
void	setMultiplier(int multiplier)
void	setType(java.lang.String type)
java.lang.String	toString()

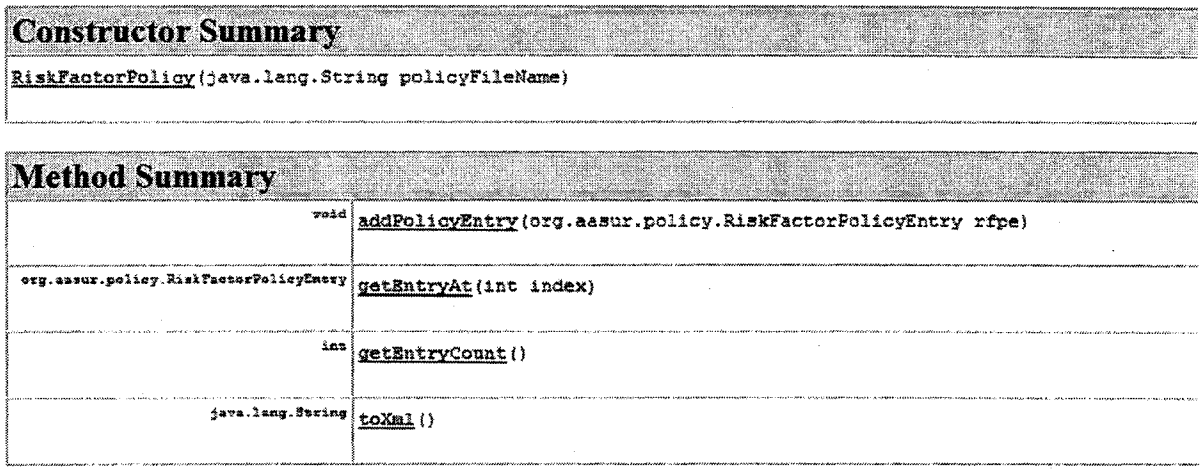
Figure 14: RiskFactorPolicyEntry Object

5.6.8 org.aasur.policy.RiskFactorPolicy (RFP)

The RiskFactorPolicy object is the collection of all defined multipliers for each point category. It contains a Vector of RiskFactorPolicyEntry(s). It does not have to contain an entry for each category. If a particular category is not defined, the value of it when the RF calculation is taking place is defaulted to 0 so that the points associated with that category are ignored. Instantiation of this class involves passing the filename which contains the persistent Risk Factor policy to the constructor. This extracts the policy from the file, and passes it to the private internal method called populatePolicy(policyFilename). This method parses the policy and for each entry creates a new instance of RiskFactorPolicyEntry and inserts it into the local vector of entries, using another local method called addPolicyEntry(entry). This vector will later be used by the RiskFactorEngine (5.6.9). Other methods within this class include getEntryCount() which allows us to see how many entries exist in this policy, and getEntryAt(index) which

returns a policy entry at a specific index. This is useful for cycling through the entries when the RiskFactorEngine is making its calculations.

Further details of these methods are shown in Figure 15.



Constructor Summary

RiskFactorPolicy(java.lang.String policyFileName)

Method Summary

void	addPolicyEntry(org.aasur.policy.RiskFactorPolicyEntry rfpe)
org.aasur.policy.RiskFactorPolicyEntry	getEntryAt(int index)
int	getEntryCount()
java.lang.String	toXml()

Figure 15: RiskFactorPolicy Object

5.6.9 org.aasur.policy.RiskFactorEngine (RFE)

The Risk Factor Engine is what takes a RiskFactorPolicy and an AasurReputation instance as input and calculates the user's overall RF. This RF along with the user's classes will be used within the AccessDecisionEngine (5.6.12) to determine what level, if any, of access will be given. From an implementation point of view, the RFE is quite simple. It only has two methods, both of which are exposed publicly. One of these methods is used to load a specific RF policy into the engine, called setPolicy(policy), which takes a RiskFactorPolicy object as input, and the other is used to calculate the Risk Factor, called calculateRiskFactor(reputation), which takes an AASURRepuation object as input. The RF is returned as a double value.

Further details of these methods are shown in Figure 16.

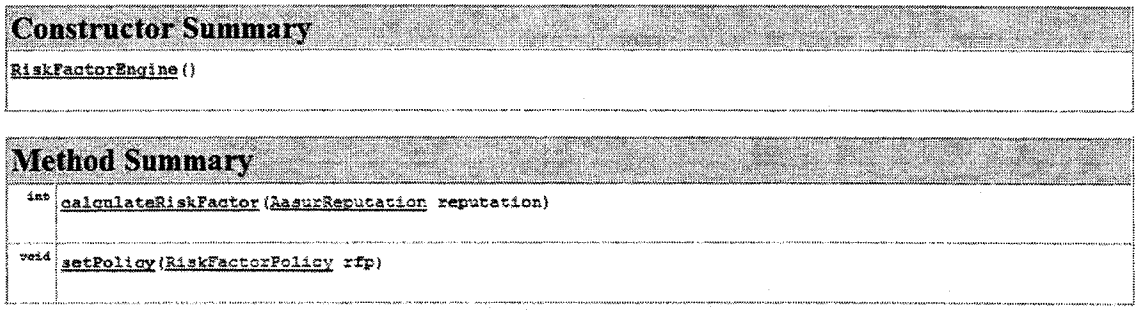


Figure 16: RiskFactorEngine Object

5.6.10 org.aasur.policy.AccessDecisionPolicyEntry

The Access Decision Policy Entry (ADPE) is very similar to the Risk Factor Policy Entry (RFPE). It is essentially a generic format to hold a specific instance of one of the entries that exist within the AccessDecisionPolicy (see 5.6.11). Each entry specifies a required RF, a level identifier (simply a unique identifier), a list of user classes and an Access Specifier. Access is granted if the RF of the user in question is larger or equal to the specified RF and the user belongs to at least one of the classes specified in the entry. If both of these conditions are satisfied, the user will receive the access defined within the Access Specifier (arbitrary site specific resource access policy string).

This class has two constructors, the default and one which accepts the four values which correspond to those associated with an entry. This class also contains the necessary get/set methods for all four variables as well as a corresponding toXml() method. The last method, called addUserClass(class) enables more User Classes to be added to the User Class Vector within the entry.

Further details of these methods are shown in Figure 17.

Constructor Summary

AccessDecisionPolicyEntry()
AccessDecisionPolicyEntry(int level, java.util.Vector classes, double rf, java.lang.String accessLevel)

Method Summary

void	addUserClass(java.lang.String className)
java.lang.String	getAccessLevel()
java.util.Vector	getClasses()
int	getLevel()
double	getRf()
void	setAccessLevel(java.lang.String accessLevel)
void	setClasses(java.util.Vector classes)
void	setLevel(int level)
void	setRf(double rf)
java.lang.String	toXml()

Figure 17: Access Decision Policy Entry Object

5.6.11 org.aasur.policy.AccessDecisionPolicy

The AccessDecisionPolicy class is used to represent the resource sites Access Decision policy as a Java object. This object will be used by the AccessDecisionEngine (5.6.12) to perform the necessary calculations to determine whether or not access should be granted.

This class has one constructor which accepts a policy filename that specifies where the text file based instance of the resource sites Access Decision policy resides. Upon instantiation of the AccessDecisionPolicy object, the file is opened and the contents

extracted. Next, the XML is parsed, the individual AccessDecisionPolicyEntry(s) are created, and the policy is ready for use.

This class has 4 public methods and one private method. The private method , called populatePolicy(policy) is used in association with the constructor to populate this object with the necessary data from the persistent reputation file. The public methods include the following in figure 18. The first method, addPolicyEntry(entry) allows a new entry to be added to the current policy. The getEntry(index) method allows for an entry at a specific location to be retrieved. The getEntryCount() method returns the number of AccessDecisionPolicyEntry(s) within this AccessDecisionPolicy, and is useful for defining loops which will be used to cycle through these entries. Finally, we have a toXml() method to format the policy for persistent file storage.

Further details of these methods are shown in Figure 18.

Constructor Summary

AccessDecisionPolicy(java.lang.String policyFileName)

Method Summary

void	addPolicyEntry(AccessDecisionPolicyEntry adpe)
AccessDecisionPolicyEntry	getEntryAt(int index)
int	getEntryCount()
java.lang.String	toXml()

Figure 18: Access Decision Policy Object

5.6.12 org.aasur.policy.AccessDecisionEngine

The Access Decision Engine component is responsible for making the final access decision based on a user's reputation. There is only a default constructor and it takes no input for instantiation. It has three methods. These methods include the necessary functionality to set/load the Access Decision Policy, called setPolicy(policy), to load the information about a user including their RF and which User Class(s) they belong to called loadInformation(rf, classesTheyBelongTo), and most importantly to make the access decision, called determineAccess(). The determination of whether or not access will be given involves cycling through the individual AccessDecisionPolicyEntry(s). For each entry, we must ensure that:

1. the user's RF is greater than or equal to the specified RF
2. the user belongs to at least one of the specified User Classes within the current AccessDecisionPolicyEntry

Further details of these methods are shown in Figure 19.

Constructor Summary

AccessDecisionEngine()

Method Summary

java.lang.String	determineAccess()
void	loadInformation(double rf, java.util.Vector classes)
void	setPolicy(AccessDecisionPolicy adp)

Figure 19: AccessDecisionEngine Object

5.7 Supporting Components

5.7.1 org.aasur.user.UserSiteSimulator

The user site package is very minimal. In fact, in typical situations, this package (with one class) would never be used. Normally, a user site would have a particular resource discovery mechanism in place and it would be that which makes a resource request to various resource sites for access. In this implementation, we do not assume the use of a particular resource discovery mechanism. The UserSiteSimulator class contains the necessary functions perform the following duties:

a. connect to various resource provider sites

b. make a request for resource access (presenting a certificate with reputation)

c. obtain the result

5.7.2 org.aasur.common.Serializer

One of the first real challenges that I faced was the limited set of types that could be passed over HTTP/SOAP to a waiting web service. In this system, we use the Java API for XML based Remote Procedure Calls (JAX-RPC) for mapping Java types to XML/WDSL definitions [Jax2006]. In general, JAX-RPC supports the following types [JaxTypes]:

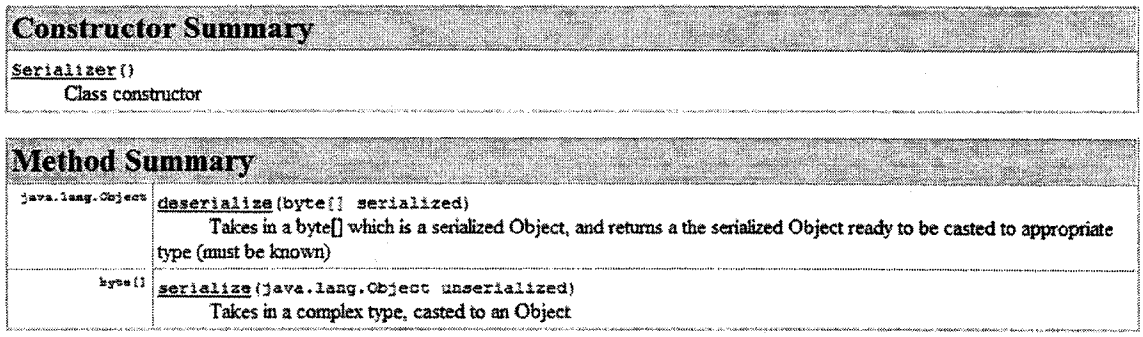
- `java.lang.Boolean`
- `java.lang.Byte`
- `java.lang.Double`
- `java.lang.Float`
- `java.lang.Integer`
- `java.lang.Long`
- `java.lang.Short`
- `java.lang.String`
- `java.math.BigDecimal`
- `java.math.BigInteger`
- `java.net.URI`
- `java.util.Calendar`
- `java.util.Date`

JAX-RPC also supports the passing of "complex types" which are basically compositions of these elementary types. This means that if you wish to pass a more complex type, like an X.509 certificate, you may have a serious problem on your hands. We considered the possibility of reconstructing the X.509 cert type with these basic types, but came to the conclusion that it simply wouldn't work from a standards point of view.

The solution to this problem came after following a discovery of serialization. In Java there is an interface called Serializable, which conveniently is implemented by almost every class. This means that each class can be serialized, with the exception of a few. Typically, serialization is used to store Java objects persistently in files. After being serialized, the object is represented as an array of bytes. Coincidentally, one of the possible types that JAX-RPC supports for passing is an array of bytes. Thus, as long as an object is serialized first, it can be sent as a web service parameter. Deserialization is the opposite action at the receiving end (to restore the Object), and happens the same way in reverse.

With this background information, the Serialization class is used to perform the serialization/deserialization functions needed to pass complex types to web services. The serialize(object) method takes an Object and returns a byte[], while the deserialize(bytes) method takes a byte[] and returns an Object. Note that you must know what type of object you are deserializing into otherwise you will not be able to cast it to the appropriate type once deserialized.

The methods which belong to Serializer are shown in Figure 20.



Constructor Summary

`Serializer()`
Class constructor

Method Summary

java.lang.Object	`deserialize(byte[] serialized)` Takes in a byte[] which is a serialized Object, and returns a the serialized Object ready to be casted to appropriate type (must be known)
byte[]	`serialize(java.lang.Object unserialized)` Takes in a complex type, casted to an Object

Figure 20: Serializer Object

5.7.3 org.aasur.common.TextCompressor

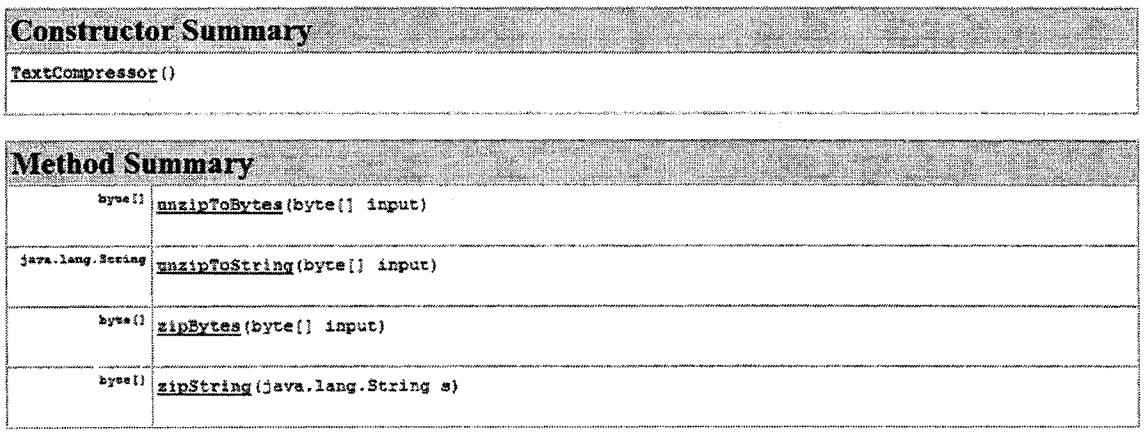
Performance is of utmost importance in any system. This is doubly so when we are talking about distributed/grid processing where resources may be geographically distributed and messaging our take a significant portion of the overall overhead. For this reason, the smaller that each message can be made, the less time/bandwidth an individual transaction will cost.

In AASUR, the reputation is stored within a user's PKC. As reputation grows larger, the amount of space required to store a user's reputation will also grow. Although this was taken into consideration in the design phase of AASUR, and it is already quite minimal, compression can make the reputation in a certificate even smaller. Of course, a smaller, more compact reputation is desirable.

For both of these reasons, compression was a natural thing to include in the system. On the fly compression/decompression would enable the reputation which is stored in the

certificate to take up less space. It would also ensure that the messages sent between hosts would be as small as possible. The TextCompressor class makes use of Java's built in zip capabilities and makes inline compression very straightforward. There are four methods available in TextCompressor. They include the necessary capabilities to zip a set of bytes (into bytes), to zip a string (into bytes), to unzip (bytes) into a set of bytes, and to unzip (bytes) into a string.

Further details of these methods are shown in Figure 21.



Constructor Summary

TextCompressor()

Method Summary

byte[]	unzipToBytes(byte[] input)
java.lang.String	unzipToString(byte[] input)
byte[]	zipBytes(byte[] input)
byte[]	zipString(java.lang.String s)

Figure 21: TextCompressor Object

5.7.4 org.aasur.common.FileContentExtractor

Several policies exist within ASSUR. These policies are used to control how a Risk Factor (RF) and User Class (UC) are determined, who belongs to the Local Blacklist, and how final decisions are made. These policies are stored persistently in text files located on the local system. When the server starts up, these policies must be loaded so that when an access decision must be made, they are available. The policies are stored in plaintext, so reading them is very straightforward. The FileContentExtractor class was simply created to avoid multiple instantiations of the necessary objects to read from a file. The

major method in this class takes a file name as input and returns a String representation of the contents of the file.

Further details of these methods are shown in Figure 22.

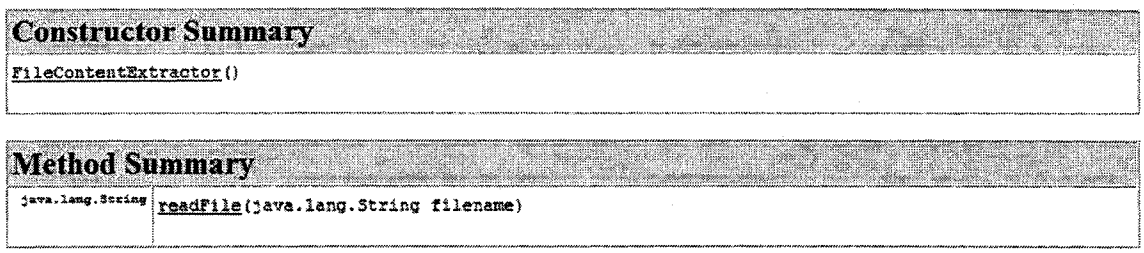


Figure 22: FileContentExtractor Object

5.7.5 org.aasur.common.RandomNumberGenerator

In the context of AASUR the RandomNumberGenerator class is strictly used for simulation purposes. A random number is used in part to generate a list of random jobs. Also, initially, a random number was used for an access decision prior to completion of all authorization decision logic. Similarly to FileContentExtractor, the RandomNumberGenerator class was simply created to avoid multiple instantiations of the necessary objects to create random numbers. The major method called generateRandomNumber(low, high) in this class takes two integers as input. One of these integers denotes a low and the other denotes a high of a range which defines where the random number that will be generated should fall.

Further details of these methods are shown in Figure 23.

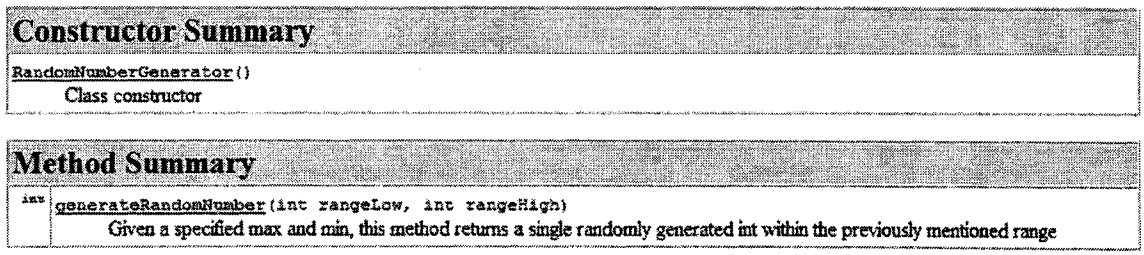


Figure 23: RandomNumberGenerator Object

5.7.6 org.aasur.common.ReputationExtractor

The act of extracting a particular extension from an X.509 certificate involves several steps. These steps have been combined into a method within the ReputationExtractor class. The major method extractReputation(certificate, ExtensionId) in this class takes two parameters as input. The first parameter is the particular certificate from which the extension in question will be extracted from. The second parameter is the unique object id (OID) which identifies said extension. This method returns a String representation of the extension from the certificate.

Further details of these methods are shown in Figure 24.

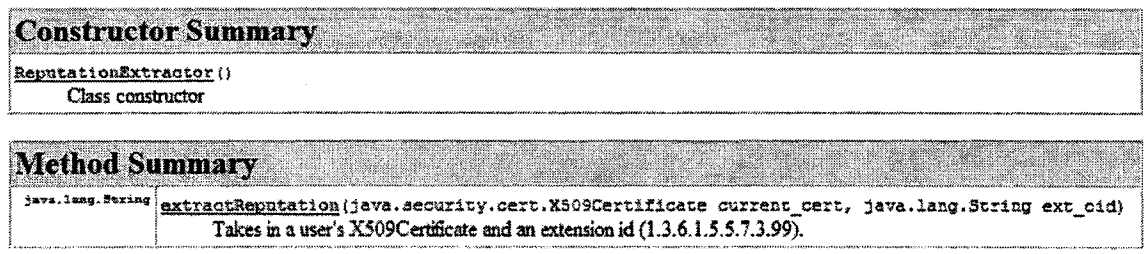


Figure 24: ReputationExtractor Object

5.7.7 org.aasur.common.XMLDomParser

XML is integrated heavily into AASUR. All policies, messages, and job actions, as well as the representation of a user's reputation are contained in specifically defined XML

formats. With this much XML, parsing takes place quite frequently. Thus, the purpose of the XMLDomParser class is to facilitate the parsing of XML. The major method in this class called parseXmlString(xml) takes a String as input which is the XML representation of the data in question. This method uses the Document Object Model (DOM) to parse this data into a Document class object instance and returns this instance for use. This document will allow the code to cycle through and target individual tags which were contained in the data for specific processing.

Further details of these methods are shown in Figure 25.

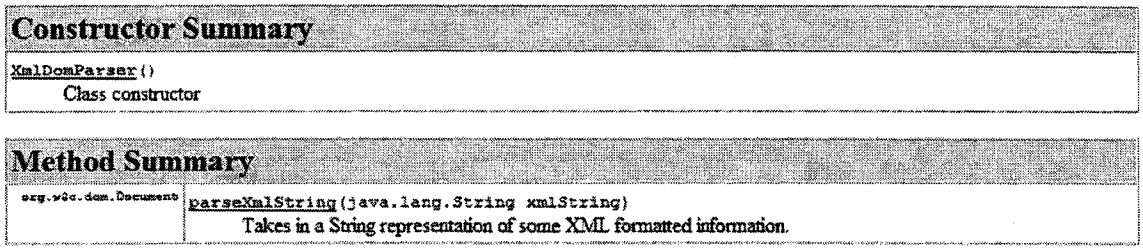


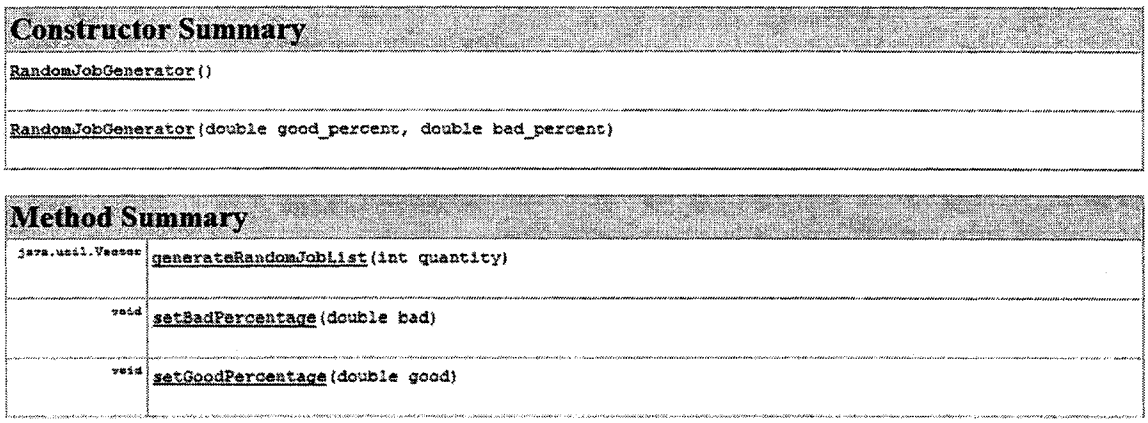
Figure 25: ReputationExtractor Object

5.7.8 org.aasur.simulation.RandomJobGenerator

The RandomJobGenerator class is the heart of the simulation aspect of AASUR. The major role of this class is to generate the list of JobActions (see 5.9.2), corresponding to the various point categories, which occurred during a particular job simulation at a resource site. These actions will then be sent to the parent CA of the user so that they may be added to the user's certificate to represent the user's constantly evolving reputation. There are several methods in this class. These include setGoodPercentage(percent), setBadPercentage(percent), and generateRandomJobList(quantity). The first two methods are used to set the percentage of good and bad actions that will be generated

during a simulated job execution. Note that these percentages may be sent to the RandomJobGenerator constructor upon object instantiation. If they are not, these methods must be used to set the values prior to any job generation. The third method, generateRandomjobList(quantity), is the major method in this class. If performs the actual random selection of the actions which occurred during processing. It takes an integer as input which defines how many actions will be generated during processing. The result is a Vector of JobActions (5.9.2).

Further details of these methods are shown in Figure 26.



Constructor Summary	
RandomJobGenerator()	
RandomJobGenerator(double good_percent, double bad_percent)	

Method Summary	
java.util.Vector	generateRandomJobList(int quantity)
void	setBadPercentage(double bad)
void	setGoodPercentage(double good)

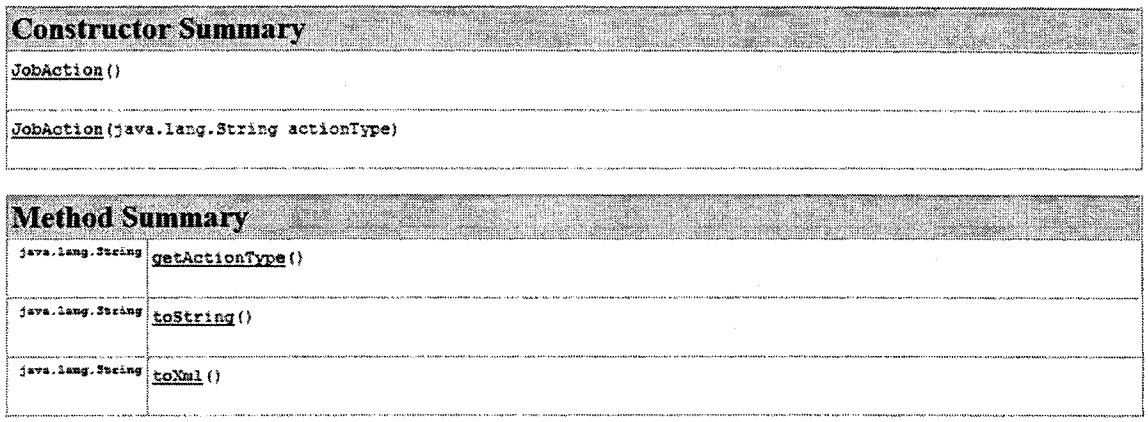
Figure 26: RandomJobGenerator Object

5.7.9 org.aasur.simulation.JobAction

A JobAction is used to represent one particular action that occurred during processing. The result of a job execution is a Vector of these JobAction(s). A JobAction is very basic, and is only composed of a type. This type is one of the many codes which represent the various point categories in AASUR. There are three methods that belong to the JobAction class. They include getActionType() to retrieve the action type as a String, and a toString() and toXml() method. None of these take any parameters. The toXml will return

a String formatted in the appropriate xml (to be returned later to the parent CA of the user) and the toString method returns a String representation which is only used for debugging purposes.

Further details of these methods are shown in Figure 27.



Constructor Summary
JobAction()
JobAction(java.lang.String actionType)

Method Summary	
java.lang.String	getActionType()
java.lang.String	toString()
java.lang.String	toXml()

Figure 27: JobAction Object

5.8 Policy Formats

A set of policies, formats, and messages has been devised so that resource owners are able to retain ultimate control over their resources. All policies, messages and formats are defined using XML. It should be noted that many of the formats are quite similar. This was done to maintain consistency which helps to enable an easy understanding of the system, as well as to ensure that as much complexity as possible has been removed from processing.

These policies are stored persistently in text files on the resident file system of the local server. They are secured via typical file system permissions. From the application server point of view, these files are read only, so that no malicious modifications may be done to

the file. Further, these files are loaded once when the server begins, and are persistent in memory for the duration of the application server lifetime. If any changes are desired in these policy files, the changes must be applied and the server environment restarted. This is to increase the overall security of the system.

There is a very important aspect regarding the definition of these files which must be discussed. As previously mentioned, we have tried to maintain a high degree of flexibility for resource owners to specify what and how access is granted through the various policies. This flexibility is somewhat of a double edged sword however. Should an administrator not fully comprehend the intent of each policy, there is a risk of improper decisions (unintentionally too easy or too difficult). To alleviate this issue, a default set of policies will be included in the system, but the best practice is to know what each number in a policy means, and how it will affect the overall decision making process. Setting values which you do not understand, can have undesired results.

5.8.1 Risk Factor Determination Policy

This policy contains the necessary information to define how the specific RF is derived, including the relative importance (weight) of each point category.

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<policy type='rf'>
        <multiplier type='djr'>     -4     </multiplier>
        <multiplier type='rou'>     -5     </multiplier>
        <multiplier type='ida'>     -3     </multiplier>
        <multiplier type='ifc'>     -3     </multiplier>
        <multiplier type='ifd'>     -4     </multiplier>
        <multiplier type='imo'>     -6     </multiplier>
        <multiplier type='inc'>     -5     </multiplier>
        <multiplier type='ips'>     -3     </multiplier>
        <multiplier type='ird'>     -5     </multiplier>
```

```
        <multiplier type='iwr'>     -5     </multiplier>
        <multiplier type='isc'>     -6     </multiplier>
        <multiplier type='isi'>     -7     </multiplier>
        <multiplier type='lbl'>     -7     </multiplier>
        <multiplier type='bof'>     -9     </multiplier>
        <multiplier type='rte'>     -6     </multiplier>
        <multiplier type='cce'>     -5     </multiplier>
        <multiplier type='lda'>     2      </multiplier>
        <multiplier type='lfc'>     2      </multiplier>
        <multiplier type='lfd'>     1      </multiplier>
        <multiplier type='lmo'>     3      </multiplier>
        <multiplier type='lnc'>     3      </multiplier>
        <multiplier type='lps'>     4      </multiplier>
        <multiplier type='lrd'>     2      </multiplier>
        <multiplier type='lwr'>     4      </multiplier>
        <multiplier type='lsc'>     3      </multiplier>
        <multiplier type='lsi'>     2      </multiplier>
        <multiplier type='pjr'>     1      </multiplier>
</policy>
```

5.8.2 Access Decision Policy

This policy corresponds to the AccessDecisionPolicy object previously defined. The AccessdDecisionEngine is the last step of the access decision. Its input includes a user's RF and User Class, and its output includes arbitrary blocks of a site specific access policy language (one for each level of access). These blocks define what a user has the permission to do if they are granted access to that level. This policy may have as many different levels as an administrator desires. Each level is associated with a required RF range (which the user's RF must fall into) and a specific User Class.

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<policy type='ade'>
        <level id='all' rfh='1000' rfl='3000' >
                <classes>
                        <class> userClass1 </class>
                        <class> userClass2 </class>
                </classes>
                <access>
                        arbitrary access specification
```

```
            </access>
        </level>
        <level id='al2' rfh='2500' rfl='5000' >
            <class>
                <class> UserClass2 </class>
            </class>
            <access>
                arbitrary access specification
            </access>
        </level>
        <level id='al3' rfh='4000' rfl='10000' >
            <class>
                <class> userClass3 </class>
                <class> userClass 4 </class>
                <class> userClass 5 </class>
            </class>
            <access>
                arbitrary access specification
            </access>
        </level>
        <level id='al4' rfh='10000' rfl='50000' >
            <class>
                <class> userClass5 </class>
                <class> userClass6 </class>
            </class>
            <access>
                arbitrary access specification
            </access>
        </level>
</policy>
```

5.8.3 User Class Definition Policy

This policy defines how the user's class is derived from non-numeric evidence and various policies. This policy gives us the ability to group a user into one or many categories which will allow us to give out various levels of access. It also helps to deal with the authorization base case where a user is brand new and has no points because they have not yet carried out any processing within the system.

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<policy type="uc">
```

```
        <class name='testClass1'>
                <rule type='age'>123456789</rule>
                <rule type='pjr'>12345</rule>
                <rule type='djr'>32</rule>
                <rule type='avg'>29</rule>
                <rule type='mrjc'>876543219</rule>
        </class>
        <class name='testClass2'>
                <rule type='age'>234567891</rule>
                <rule type='pjr'>23451</rule>
                <rule type='djr'>132</rule>
                <rule type='avg'>39</rule>
                <rule type='mrjc'>765432198</rule>
        </class>
        <class name='brandNew'>
                <rule type='age'>0</rule>
                <rule type='pjr'>0</rule>
                <rule type='djr'>0</rule>
                <rule type='avg'>0</rule>
                <rule type='mrjc'>0</rule>
        </class>
</policy>
```

5.8.4 Local Blacklisting Actions Definition Policy

This policy defines exactly which particular action(s) will cause a user to become locally blacklisted. This policy can contain any number of actions including zero (if the resource site does not feel that the use of a local blacklist is valuable).

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<policy type='LBL>
        <action type='IMO' />
        <action type='ISC' />
        <action type='IDA' />
        <action type='BOF' />
        <action type='CCE' />
        <action type='RTE' />
        <action type='IWR' />
        <action type='IPS' />
        <action type='INC' />
</policy>
```

5.9 Message Formats

5.9.1 Resource Access Request

This is the initial message sent from a user that desires some type of resource access. It is sent to the resource site that it would like to participate with. It will contain the requesting sites PKC signature (user identification). At a later time, this request could contain various detailed levels of requested access. The only potential issue with requesting a level of access involves the language used to specify the access level request. We have developed an architecture which allows a resource site to use an arbitrary access policy language for the ticket, however, there must be some accepted language for the access request. This implies that all sites must conform to a certain language, at least for the request aspect of authorization. See below for an example of a resource request.

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<resource_request>
        <sha1>
                6AC824B3EDDD8748A2B1261249D84EE48B37A96
        </sha1>
</resource_request>
```

5.9.2 Point Action Notification

This particular format defines the messages that will be sent from a resource site to the parent CA of the user which has just completed a job at that site. The message will contain both the signature of the user and resource site in question, the start and end time (in millis) of the job, and a list of all individual actions which took place during processing.

```
<?xml version='1.0' encoding='ISO-8859-1'?>
```

```
<message
  type='PN'
  site_sha1='U7H99YC6THTD1B3W8FGL35RKUM53FHDVFFLLZZZ1'
  user_serno='7H99YC6THTD1B3W8FGL35RKUM53FHDVFFLLZZZ1'
  start_time='1178467068203'
  end_time='1178467068250'>
      <action type='LMO' />
      <action type='LSC' />
      <action type='LSC' />
      <action type='LDA' />
      <action type='LFC' />
      <action type='LWR' />
      <action type='LRD' />
      <action type='LFD' />
      <action type='LFD' />
      <action type='LFD' />
      <action type='LRD' />
      <action type='LSI' />
      <action type='LRD' />
      <action type='LSC' />
      <action type='LFD' />
      <action type='LMO' />
      <action type='LNC' />
      <action type='LWR' />
      <action type='LFC' />
      <action type='LWR' />
      <action type='LWR' />
</message>
```

5.9.3 Resource Access Ticket

This is the access ticket which will be generated upon completion of the access decision at the resource site. It will contain PKC of the owner (user which has been given access), the PKC of resource, and the granted permissions.

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<access_ticket>
      <user_sha1>
            6AC824B3EDDD8748A2B1261249D84EE48B37A96
      </user_sha1>
      <resource_sha1>
            6AC822B1261249D84EE48B37A964B3EDDD8748A
```

</resource_sha1>
<access_granted>
this will contain a chunk of policy language, specific to the site which has given this ticket, that contains a specification of access rights which have been granted to the requesting user. The benefit of allowing arbitrary policy to be inserted into the access ticket is that our architecture is completely policy language independent.
</access_granted>
</access_ticket >

5.10 OTHER FORMATS

5.10.1 Certificate Point Format (Reputation)

All points within our architecture will be stored in the user's PKC in this format. As previously discussed, this format was developed, after several design changes, to be very efficient and to have a very minimal footprint. See below for a very young sample certificate point format.

```
<?xml version='1.0' encoding='ISO-8859-1'?>
      <reputation fjr='1178467068203'
          fjc='1178467068250'
          mrjr='1178467068203'
          mrjc='1178467068250'
          ajt='47'
          tj='1'
          c='canada'
          pjr='1'
          lda='6'
          lfc='7'
          lfd='14'
          lmo='9'
          lnc='9'
          lps='3'
          lrd='8'
          lwr='20'
          lsc='3'
          lsi='11'
          djr='0'
          rou='0'
          ida='0'
          ifc='1'
          ifd='0'
          imo='0'
          inc='2'
          ips='0'
          ird='2'
```

```
        iwr='1'
        isc='0'
        isi='3'
        bof='0'
        rte='0'
        cce='1'>
    </reputation>
```

5.10.2 Local Blacklist Format

Any user/resource which performs an action which the local resource deems worthy of addition to the local blacklist (as defined in 5.8.4), will be added to the local sites blacklist. As previously mentioned these are only applicable at, and are restricted to, individual resource sites to ensure that the unwarranted addition of a user to a blacklist does not affect the overall trust of a user in the global grid.

Each entry to the local blacklist contains the signature of the user which was added to the blacklist and what action caused that user to be added to the local blacklist. See below for an example of a resource sites local blacklist.

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<local_blacklist>
        <entry action='cce'>
                <user>C824B3EDDD8748A2B1261249D84EE48B37A96</site>
        </entry>
        <entry action='imi'>
                <user >C824B3EDDD8748A2B1261249D84EE48B37A96</site>
        </entry>
        <entry action='isc'>
                < user >FTGRC3EDDD87482B1261249D84EE48B37A96</site>
        </entry>
        <entry action='ida'>
                < user >D874AC824B3ED8A2B1261249D84EE48B37A96</site>
        </entry>
<local_blacklist>
```

5.11 Extensibility

Many aspects of this system have been designed in such a way that extension of this architecture is very straightforward. The question then becomes what would someone want to add to this system most commonly, or to what degree can the system be easily extended. The answer to the first question is point categories. Various resource sites may desire to add new point categories to the system for their own custom use. In theory this is straightforward for much of the system, but you will be quick to notice that it is in fact quite involved.

As far a simply adding a new point type to the RF policy and the user's permanent reputation, this is very simple. Everything was designed in such a way that adding a new category to the policy would automatically be recognized, and the same applies with the reputation within the certificate. If some site defined a category in their RF policy which did not exist, it would have no bearing on the overall RF since there would be 0 occurrences of it which amounts to nothing in the RF. Further if a new point category was created in the system and a site did not define it in the RF policy, again a zero value in the multiplier would render it neutral. The entries in an RF policy are not statically defined, so new entries could be added at any time.

There is however a problem with respect to extension. The problem involves the point collection middleware which sits at each site. This code is signed to ensure everyone has a consistent view of how to collect points to eliminate any subjectivity from a collection point of view. The problem is that any changes (new category) will have to be added to

the middleware at each and every one of these sites in the world. Although this hinders extensibility, an update system could make the distribution of an update very straightforward. This is left as future work. It must be noted that no site may operate in the context of the grid while using non-standardized point categories in their AASUR system. If this was the case, the objectivity and standardization which makes the system valuable, no longer exist.

5.12 Sample Walkthrough (SYSTEM FLOW)

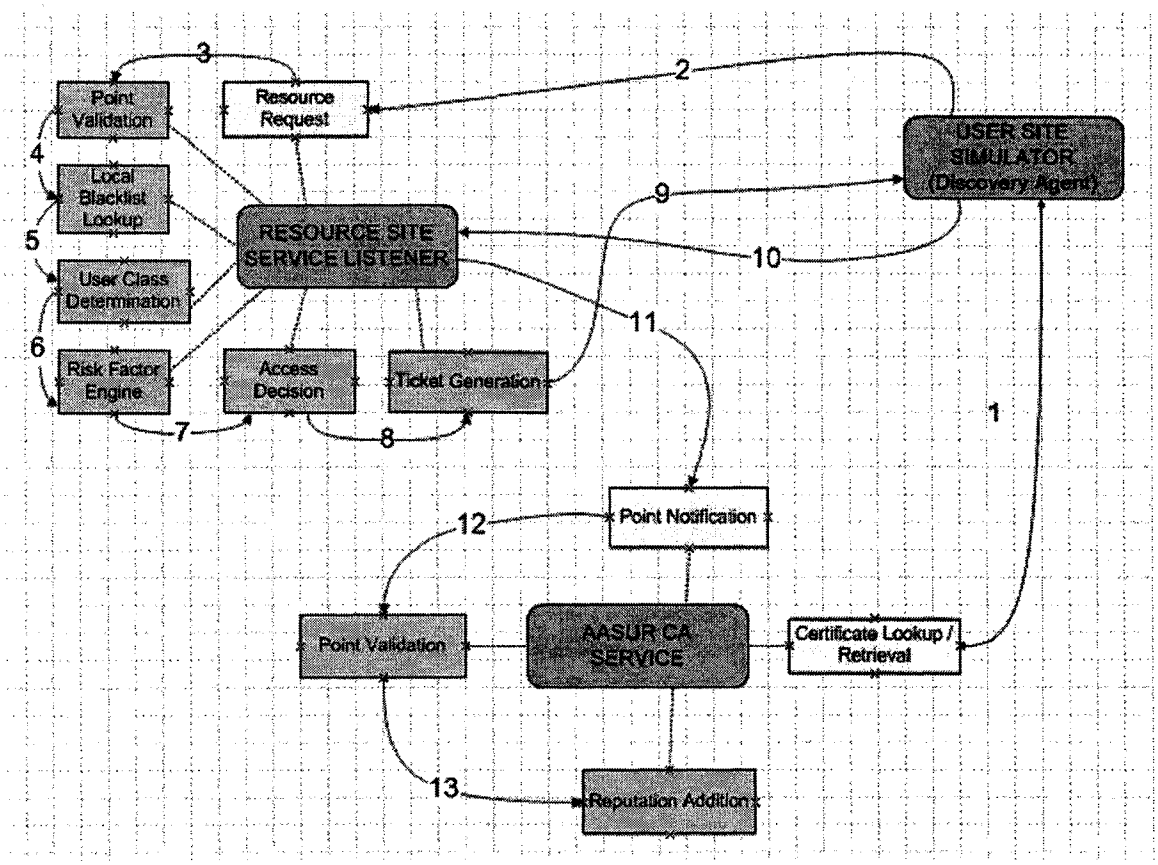


Figure 28: A step-by-step representation of the AASUR system workflow.

The AASUR system workflow is shown in Figure 28. We discuss the workflow steps, in detail, below.

1. The user, needing access to some remote resource, contacts the ***AasurCaService*** which resides on its parent CA and performs a certificate lookup/retrieval. The returned certificate will be used for the upcoming resource request.
2. Resource discovery agent from the User sends a request to the ***ResourceSiteServiceListener*** at the Resource Site. This request includes an X.509 PKC containing past actions of the user, specified in the Certificate Point Format
3. The ***ResourceSiteServiceListener*** will then validate the certificate to ensure that it hasn't expired or been revoked.
4. Next, the ***ResourceSiteServiceListener*** must check to see if the user in question has lost our trust by checking to see whether they belong to our local blacklist. If so, we may immediately deny access to out resources.
5. Along with the Risk Factor, the access decision may also take into account the requesting user's class. A user's class will be determined based on additional non-point based information as defined in 5.8.3. This will also include several of the attributes define in Table 3. The notion of user class is important because it is used to govern what degree of resource access will be given, assuming that some level of access will be granted.
6. Next, the Risk Factor (RF) is generated based on local relative multipliers specified in policy as defined in 5.8.1.

7. Now, an access decision must be made. Essentially, it provides a yes or no answer, with associated policy specifying access. This decision is based on the derived Risk Factor, the user's class, and the Access Decision Policy defined in 5.8.2.
8. If access is granted, it will manifest itself in the form of a ticket, as defined in 5.9.3, which will be presented upon job submission at the resource site. This ticket is similar to that used in Kerberos [Kohl1991]. We use the notion of a ticket for two reasons. First, within the context of grid based computing, this architecture allows us to offer our authorization service, minimally, to resource discovery mechanisms. Rather than discovering appropriate resources and distributing a job simply to find out that access will no longer be given, we can determine where access will be given based on the acquisition of a ticket. This ticket, with an explicit lifetime, will be presented upon distribution of the job to the resource site for processing. Secondly the notion of delegation is very crucial to distributed processing. This ticket would enable delegation to occur, similar to a proxy. It must be noted however that this current version of AASUR does not support delegation, though it could be incorporated into AASUR with some additional work.
9. The generated ticket is then returned to the requesting user for submission with the job.
10. When a user is ready to submit the job, assuming the ticket has not expired, it will be sent to the resource site along with the access ticket. Once a job has begun execution, we must monitor exactly which actions are performed by the job. We

require a middleware layer (outside the scope of this paper) which handles the detection and collection of actions performed during processing. It should be noted that if at any time during execution, the job performs an action that is considered locally blacklistable (defined in 5.8.4), the job will be terminated and the proper entries will be added to the local blacklist defined by format 5.10.2.

11. Once processing has finished, the resource site will connect to the parent CA of the user which owns the finished job. The CA will have an instance of the **AasurCaService** running locally. The resource site will then send a list of all actions which occurred during processing. The listed will be formatted as defined in 5.9.2, and will contain all local actions performed.
12. Once the CA receives the notification, it will perform a few simple validations, lookup the proper associated PKC, and format the actions into the appropriate XML.
13. Finally, the actions are merged into the user's certificate using the format defined in 5.10.1. Note that the new points must be added by the parent CA of the entity which earned the points, not the CA of the resource site. This is due to the fact that only the parent CA of the certificate owner can sign the certificate.

5.13 Sample Walkthrough (AUTHORIZATION DECISION)

In order to illustrate the details of our approach, we examine a sample authorization decision including specific examples of policies and reputation. In order to reach the final decision it is necessary to deal with each separate aspect of the decision making process, starting from user reputation, to determination of the user class and ending with the actual access decision.

We begin with the reputation of a specific user.

USER REPUTATION

Below we show an example xml document, contained within the extension of the X.509 certificate, to be used for the user reputation calculation. The document contains the various point categories and values pertinent to this example.

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<reputation fjr='1178467068203'
        fjc='1178462068289'
        mrjr='1178467068203'
        mrjc='1178467068250'
        ajt='47'
        tj='1'
        c='canada'
        pjr='19'
        lda='6'
        lfc='7'
        lfd='14'
        lmo='9'
        lnc='9'
        lps='3'
        lrd='8'
        lwr='20'
        lsc='3'
        lsi='11'
        djr='5'
        rou='2'
        ida='0'
        ifc='1'
        ifd='0'
        imo='1'
        inc='2'
        ips='0'
        ird='2'
        iwr='1'
        isc='6'
        isi='3'
        lbl='3'
```

```
        bof='0'
        rte='4'
        cce='1'>
</reputation>
```

Notice that not every point category has a non-zero number in it. This simply means that this particular action has never been performed by this user. The second piece of information provided is the RF policy to be used to determine the overall Risk Factor.

RF POLICY

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<policy type='rf'>
    <multiplier type='djr'>     -4    </multiplier>
    <multiplier type='rou'>     -5    </multiplier>
    <multiplier type='ida'>     -3    </multiplier>
    <multiplier type='ifc'>     -3    </multiplier>
    <multiplier type='ifd'>     -4    </multiplier>
    <multiplier type='imo'>     -6    </multiplier>
    <multiplier type='inc'>     -5    </multiplier>
    <multiplier type='ips'>     -3    </multiplier>
    <multiplier type='ird'>     -5    </multiplier>
    <multiplier type='iwr'>     -5    </multiplier>
    <multiplier type='isc'>     -6    </multiplier>
    <multiplier type='isi'>     -7    </multiplier>
    <multiplier type='lbl'>     -7    </multiplier>
    <multiplier type='bof'>     -9    </multiplier>
    <multiplier type='rte'>     -6    </multiplier>
    <multiplier type='cce'>     -5    </multiplier>
    <multiplier type='lda'>     2     </multiplier>
    <multiplier type='lfc'>     2     </multiplier>
    <multiplier type='lfd'>     1     </multiplier>
    <multiplier type='lmo'>     3     </multiplier>
    <multiplier type='lnc'>     3     </multiplier>
    <multiplier type='lps'>     4     </multiplier>
    <multiplier type='lrd'>     2     </multiplier>
    <multiplier type='lwr'>     4     </multiplier>
    <multiplier type='lsc'>     3     </multiplier>
    <multiplier type='lsi'>     2     </multiplier>
    <multiplier type='pjr'>     1     </multiplier>
```

</policy>

Once we have the user's reputation and the RF policy we perform the first major step in the access decision, namely the calculation of a user's RF. If we define the resulting RF as the equation:

$$RF = \sum_{k=1}^{K} m_k \, \Theta(C_k) C_k \Big/ \sum_{k=1}^{K} m_k \left| C_k \right|$$

In this equation, K is the total number of point categories chosen, m_k and C_k are the weight and number of occurrences, respectively, for the kth point category and we utilize the step function $\Theta(x) = 1$ for $x \geq 0$, and 0 otherwise. Thus, the RF determines the relative fraction of positive to total weighted multiplier sums.

For illustration purposes, first, the negative multipliers are used to calculate the negative part of the point sum:

(5 * -4) + (2 * -5) + (0 * -3) + (1 * -3) + (0 * -4) + (1 * -6) + (2 * -5) + (0 * -3) + (2 * -5) + (1 * -5) + (6 * -6) + (3 * -7) + (3 * -7) + (0 * -9) + (4 * -6) + (1 * -5) = -171.

Next, we use the positive multipliers to calculate the positive part of the point sum

(6 * 2) + (7 * 2) + (14 * 1) + (9 * 3) + (9 * 3) + (3 * 4) + (8 * 2) + (20 * 4) + (3 * 3) + (11 * 2) + (19 * 1) = 252.

Now the next step prior to obtaining an RF is to add the absolute values of the negative part to the positive part to obtain the absolute point sum:

|252| + |-171| = 423

Finally, we divide the positive part only by the absolute point sum (rounded to three places):

252 / 423 = 0.596

This value (0.596) is known as the RF. Essentially, the RF defines what percentage of the time this user has been "good" (as an overall, or collective, measure) using the individual weights defined in the Risk Factor policy for each point category.

Two remaining policies must be defined. These include the User Class policy and the Access Decision policy.

USER CLASS POLICY

In order to arrive at an access decision it is necessary to know what kind of user we are dealing with. The following xml document contains the information needed to make this determination (see 16.9.3).

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<policy type="uc">
        <class name='newUser'>
            <rule type='age'>1178467096503</rule>
            <rule type='pjr'>12</rule>
            <rule type='djr'>5</rule>
            <rule type='avg'>50</rule>
            <rule type='mrjc'>1178467037203</rule>
    </class>
    <class name='oldUser'>
            <rule type='age'>11784674354567</rule>
            <rule type='pjr'>250</rule>
            <rule type='djr'>35</rule>
            <rule type='avg'>35</rule>
            <rule type='mrjc'>1178469068203</rule>
    </class>
</policy>
```

Using the User Class Policy (above) we can determine which classes a particular user belongs to. Note that a user may belong to none, one, or more than one class depending on the local policy, and their reputation. To calculate the user classes, we must use the following information from the user's certificate:

Age (first job request was `1178467068203`)

Permitted Job Requests (19)

Denied Job requests (5)

Average Job Time (47 ms)

Most Recent Job Completed (`1178467068250`)

This particular user class policy dictates that to belong to it (newUser) a user must:

a. Have become active in the grid before: 1178467096503. For this we check to see if a user's First Job Request (FJR) is less than the policy age value. This is because the milliseconds specified denote the number of milliseconds from January 1, 1970. This means that a lower number is older.

   Check: 1178467068203 < 1178467096503 **TRUE**

b. Have at least 12 Permitted Job Requests

   Check: 19 > 12 **TRUE**

c. Have no more than 5 Denied Job Requests

   Check: 5 <= 5 **TRUE**

d. Have an Average Job Time of 50ms or less.

   Check: 47 <= 50 **TRUE**

e. Has requested access to a resource site no longer ago than `1178467037203`

   Check: 1178467068250 >= 1178467037203 **TRUE**

Since all of the five checks for the User Class known as newUser pass (ie. are TRUE), this user belongs to the **newUser** class. In contrast, if we were to check to see if the user is a member of oldUser, we would see that they are not (for example, 250 is the lower limit for Permitted Job Requests that qualify an **oldUser**, hence the current value of 19 fails).

ADE POLICY

Finally, we must define the Access Decision policy which will make the final access decision. We start with the following xml document:

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<policy type='ade'>
    <level id='1' rfl='0.500' rfh='0.700' >
            <classes>
                    <class> newUser </class>
                    <class> oldUser </class>
            </classes>
            <access>
                    arbitrary access specification
            </access>
    </level>
    <level id='2' rfl='0700' rfh='0.900'
            <class>
                    <class>oldUser</class>
            </class>
            <access>
                    arbitrary access specification
            </access>
    </level>
</policy>
```

We consider the user's RF and Class(s) and see what type of access is associated with them within the Access Decision policy. According to the AD policy above, this user will

be given access. The access specifier string will be returned is the 'arbitrary access specification' located between the <access></access> tags. The specification will be given in a site specific resource access policy language. One such example policy language framework is Ponder [Damianou2001].

5.14 Current State

The current state of the AASUR system is as the design has specified. It is complete and several sets of results have been obtained from the working prototype. Items specified/contained within the future work section would be very beneficial as additions to this system, but they have not yet been implemented. The codes for this system are available upon request for your perusal, but are not included in this document as an appendix due to the large amount of space which would be required.

5.15 Limitations

It is an accepted fact that the only way to secure a resource is to ensure that no access is possible, either direct or remote. That being said, the need for resources to be available via a network is extremely important; therefore, necessary security precautions must be taken. A multi-level approach is preferable with respect to security to ensure that a failure of one layer will not make the system as a whole vulnerable.

The nature of this system, from a reputation collection point of view, is completely objective. The collection code is protected, so there will be not question about whether or not a user's reputation is correct. However, an improper decision on the part of a resource when developing an access policy can result in an undesirable access situation. Further, a historical pattern of proper usage by a user does not guarantee that the same user's future

actions will be completely proper. This is an element of risk that exists within this system. This aspect however, arguably exists within any system. There is always potential for an authorized used to perform malicious actions. A systems saving grace is the ability to detect and record that this occurrence and put some type of punishment into action. That may take the form of banishment, lessening of access right, etc.

In the case of AASUR, we have the capability to detect when such an action occurs and AASUR records it as part of a user's reputation. The major function of AASUR is to collect objective information and make it available to resource site for decision making purposes. However, even though illegal actions are recorded, that does not imply some type of punishment may take place. It is up to an individual resource site to be able to view all actions that a user has performed and make a decision based on them. That is, if a particular user has performed many "bad" actions, it is likely that many resource sites will not give access. This is analogous to the previously mentioned punishment. There should be no reason that, if a resource site still wishes to provide access for a user that has performed many improper actions, it should not be permitted to. After all, resource access should be controlled completely by the resource owner.

5.16 Implementation

The first step of development was to create a service (AasurCaService) that provided a publicly exposed WS based interface to the local EJBCA functions such as certificate lookup and creation/signing. The next step was to write a simple client piece of software (which became the UserSiteSimulator) that could simply connect to the AasurCaService and send simple messages. Since SOAP only allows the passing of simple types and

compositions of simple types, the next step was to determine how to pass complex types such as certificates. The solution was serialization. Serialization allows most Java objects to be converted into an array of bytes. Typically this is for persistent storage, but since byte arrays can be passed via SOAP to web services, it fit perfectly. Once a complete certificate could be passed successfully, the next important step was to write the appropriate code to write and access custom extensions to standard X.509 certificates. This is where the persistent reputation will be stored. These were the most important hurdles to overcome, not only to function properly, but also to ensure that development of this architecture was even feasible.

The remaining steps included development of various custom objects for reputation storage and policy representation, code for merging of existing reputation with newly performed job actions, XML formatting/parsing code, supporting code including serialization/de-serialization packages, and finally simulation code to produce randomized lists of job actions which occurred during simulated processing.

6 Development/Deployment Infrastructure

Implementation involved four major steps, which we will outline in this section. The first step involved creating a list of required components and their required functionality. The second step was to make a list of all existing software packages, determine what individual packages can do, and finally to determine which software would best compliment each other while simultaneously meeting the needs of this system. The third step, and probably the most involved, was to install and configure the selected software to provide a completely functioning infrastructure which provides the necessary framework for step 4 (See **Appendix A** for detailed installation instructions). Step 4 was to develop the AASUR specific software including service code, as well as all required supporting codes. Each of these steps will be outlined in the following subsections.

6.1 Required Components

At the most basic level, this architecture requires a certificate authority and the capability to host web services. These two simple requirements become more involved when we delve deeper into the required supporting software. The ability to host web services requires several pieces of software, including an application hosting environment which allows them to be exposed to the public for use. It also requires the necessary XML libraries for parsing of messages and formats and security libraries to provide authentication and message protection. A runtime environment may also be needed depending on choice of implementation language.

6.2 Selection of Software

Once the basic requirements had been outlined, the specific choices of software became the next step. For several benefits, Java was selected as the implementation language. Thus, the need for the Java Runtime Environment was obvious. Due to the limitations imposed from selecting one language, as well as a desire to remain within the open source community (for cost benefits, as well as the requirement to modify the software as needed), a clear winner emerged for both the hosting environment and certificate authority. The hosting environment which was selected was JBoss [Jboss], and the CA software was EJBCA [Ejbca]. Both are Java based, open source, meet various standards, and have large feature sets. The selection of EJBCA also created a requirement for a database. Again, due to the widespread use and open source nature, MySQL [MySql] was selected. For XML, the selected tools are included with built in support from Java for DOM/SAX perusal/manipulation. From a supporting software point of view, this implementation also requires Ant [Ant], for building/deploying AASUR implementation to the application hosting environment, and the necessary Java MySQL connector libraries.

It must be mentioned that the set of software components which have been selected to implement this system are in no way, shape, or form, the only tools which can accomplish our goals. Other existing languages, tools, and components can achieve the same end. The set which has been chosen, in our opinion, is simply the best choice from many points of view including rapid development, security, ease of use, performance, cost, etc. Should AASUR become a viable option for authorization, its continued

development would likely see specific components exchanged to meet the needs of a long term authorization system which has different goals.

6.3 Installation and configuration

Note that the specifics of individual software packages selected for installation on the various test bed machines will be discussed in more detail in following sections. Installation and configuration began with the configuration of several machines that would be used for the required test bed. We will outline the installation procedures in this section, but for full details see **Appendix A.**

A total of five machines were used. Four machines were configured with Fedora Core 5 (a flavour of Linux) and the last was configured with Windows XP. All machines were placed on the same subnet and file sharing was enabled for ease of file transfers via Samba and Windows file sharing. The following steps involved installing Java and Apache Ant. Then JBoss was installed and configured, including the necessary configurations for secure HTTP (HTTPS). Once JBoss was working properly, MySQL was installed, followed by EJBCA. Aside from the basic previously mentioned installation/configuration steps, some modifications to the inner workings of EJBCA were necessary to prepare it to handle what would be required for the AASUR implementation. Those modifications are discussed in 6.3.4.3.

6.3.1 JBoss

JBoss is a Java based application server, produced by Red Hat, for use with J2EE and the associated Enterprise Java Beans. It is similar in functionality to other popular application

servers such as BEA Web Logic and IBM Web Sphere [BEA, WEB]. It is completely free, Open Source, interoperable, and portable to any platform that has a JRE written for it. This makes it a very desirable product. Of course, the fact that it is free does not mean it lacks quality. In fact, JBoss is the most used Java application server [jboss].

6.3.2 Java

Java, developed by Sun Microsystems, was our programming language of choice for this project. There are several factors that make Java good choice. First, Java is interpreted via a Runtime Environment (RTE). This means that and Java code is portable to any platform which has the accompanying RTE. This platform independence means is it very portable. Second, Java was designed to be a very secure web enabled language. For an architecture that will be used for Web Services, this type of language is very beneficial. Third, it is Object Oriented (OO), which makes it very good for modeling various existing objects, as well as modeling custom objects for the system. The benefits of OO can easily be the subject of another paper, but to summarize, OO languages provide: abstraction, polymorphism, encapsulation, inheritance, and modularity. This flexibility is very useful when developing new prototypes. Fourth, the syntax is very straightforward and the error message capabilities are advanced which allows for rapid and efficient development of new systems. Fifth, automatic garbage collection means that explicit memory management is a thing of the past. No more will memory leaks be a major issue in new, large scale, systems. Sixth, Java is completely free to use. Seventh, Java has a very large number of packages available to accomplish many ends. These existing classes are very useful for quick development of new system. In other words, you don't have to reinvent the wheel.

Of course, no programming language is perfect. There are always downsides, and Java is no exception. The most prominent negative aspect of Java is its performance. Though vastly improved from its original days, it is generally accepted that Java does not perform quite as fast as C/C++. Various tests have been performed to compare the two, and it has been found that the difference is only marginal between the two [JavaWorld]. This is due to the RTE and how it acts as an interpreted, the garbage collection and security model [JavaWorld]. Since C/C++ is compiled to native hardware instructions, it will run faster. However, this means that C/C++ is nowhere near as portable as Java. Nevertheless, it is a tradeoff, and one which must be carefully considered when selecting an implementation language. Second, some consider the highly secure environment to be somewhat restrictive, but perhaps that is due to improper understanding of the language.

6.3.3 Linux

Linux is an Open Source operating system. It has been around since roughly 1990, and ever since has been gaining a larger market share in the OS race. It is predominantly a server operating system which boasts better security, stability, and extensibility than other more commonly known operating systems. We chose it because it is very capable, secure, and of course its cost.

6.3.4 EJBCA

EJBCA is an Enterprise Java Bean (EJB) based Certificate Authority (CA). It is a fully functional CA. There are so many features that listing them here would take up too much space, but suffice to say it meets and exceeds every requirement we have. In this section,

we will be discussing the pros and cons of EJBCA, as well as the necessary modifications that took place.

6.3.4.1 Benefits

It has a very flexible and feature rich web-based (and CLI) interface for administrative purposes. It supports key sizes up to 4096 bits, smart cards, multiple hashing algorithms, and multiple Hardware Security Modules (HSMs), all for maximum security [EJBCA]. It can integrate with any Lightweight Directory Access Protocol (LDAP) compliant repository including Active Directory [EJBCA]. This product is also an Open Source piece of software. The fact that it was created in Java makes integration with our system a much more palatable proposition. The only issue with the selection of EJBCA was that (at the time of selection) it did not operate in the context of web services. That being said, we decided to use it anyhow, and make the necessary modifications ourselves.

6.3.4.2 Downside

There is really only one issue with EJBCA, and this could be attributed to the overall design of ASSUR rather than EJBCA. In general, a PKC is generated once. This is due to the intention to keep a name bound to a public key for a long period of time. When a new certificate is generated (once for each complete job) the old version of the certificate is stored for historical reasons. These old versions are also used for Certificate Revocation List (CRL) generation. Due to the fact that a user could complete hundreds, if not thousands, of jobs a day, the number of historical certificates on file could grow very large. This may affect the overall scalability of this system. It is recommended that, in the future, only the latest certificate is maintained for a user. However, at this point in time,

the historical feature has been left in, and has also been determined to be acceptable from a performance point of view for this proof of concept architecture.

6.3.4.3 Necessary Modifications

As previously mentioned, the Web Service capability was not available in EJBCA when it was researched. We believe that the newest version does incorporate this type of exposure, but we have long since made the necessary modifications to handle them ourselves. Making the necessary functions, which include certificate lookup and retrieval, was not very difficult. The main steps behind this type of addition involved the following:

1. select the necessary functionality to expose as web services
2. determine how theses services can be accessed from the local CA server through EJB sessions
3. create new classes which will be publicly exposed as services that will instantiate a local session of a particular bean (eg. RSASignSessionBean for certificate signing) which will serve as an interceptor.

For example, consider the signing of a new certificate from a remote location. We have already exposed a public service which mirrors the available functionality in RSASignSessionBean (referred to as RSAMirror). When a request arrives for a certificate to be signed, it comes in to RSAMirror, which has already established a local session with RSASignSessionBean. The certificate to be signed is passed through RSAMirror to RSASignSessionBean and then EJBCA performs its required action.

At this point, perhaps it is better to say that we did not modify EJBCA; rather, we augmented it with respect to the Web Service exposure requirements. There were some things within EJBCA that did need modification, however. These are discussed in the next paragraph.

The X.509 standard allows for non-mandatory extensions within a certificate. These extensions can be named and accessed within a certificate when needed. Although EJBCA supports certificate extensions, the way in which it supports them required some modifications to the code. Prior to any modifications, the addition of a particular extension to a certificate (created/signed by EJBCA) required the creation of a user profile. Within this profile, the extension id and the value of the extension were specified. Then, any new certificate which was created under the umbrella of this profile would contain the new extension. The problem with this is that the extension is fixed. In the case of AASUR, the extension value will be changing each time a certificate is resigned. For this reason, EJBCA had to be augmented. Once this modification was complete, EJBCA supports this ability via the public certificate signing web service. When a certificate is to be created the arbitrary extension value is passed into the web service with the other required information such as primary key, etc. EJBCA will add this to the certificate dynamically, sign it, and return it to the user, or store it.

6.3.5 MySQL

MySQL is an enterprise class, multithreaded/multi-user, completely free and Open Source database management system (DMBS) [MySql]. It is very robust and has a large feature set. We chose this database because of several reasons:

1. cost
2. open source
3. scalable (supports clustering services)
4. large scale data support
5. user friendly graphical monitoring/administration tools
6. interoperability (works with a long list of programming languages)

Statistics show that MySQL has an installation base of over 10 million machines [MySql].

7 Testing and Verification

The testing and verification of this system was performed for two major reasons. The first was to ensure that all code was sound, meaning everything worked as it was designed to and did what it was expected each and every time. The second reason for testing was to determine what kind of performance this system was likely to provide, as fast processing is very important in distributed computing.

7.1 Code Correctness Verification

In order to verify the correctness of the code we adopted the following approach. First, several users were created within the CA. Each user was given a reputation which would put them into a specific User Class with a specific RF. Then, one at a time, all users made 10 job requests (each containing 100 actions, with various good/bad percentages, etc. 90/10, 75/25, 50/50, 25/75) to a resource site. Each request, and all associated logging code, was examined thoroughly for errors. Upon being satisfied that all results were correct, the next step was to determine what exact metrics would be used for the performance analysis.

7.2 Metrics

The metrics selected to measure the overall performance of a system are very important. The selection of the wrong metrics can make a system appear worse/better than it is. The comparisons for AASUR were considered very carefully over a period of time. The information which has been obtained is very promising. The following metrics have been selected to analyze the overall performance of the AASUR system:

1. certificate retrieval time (before reputation)

2. certificate retrieval time (after reputation) for a new user (no reputation)
3. certificate retrieval time (after reputation) for a user with up to 100,000 actions in their certificate
4. relationship between the number of actions in a user's reputation and the size of that user's certificate
5. access decision time for a user with up to 100,000 actions in the certificate

7.3 Procedures and Results

7.3.1 Test A: Certificate Retrieval Time (NO REPUTATION)

For this test we created a new user in the CA, which did not have the new X.509 extension representing the reputation XML. We then timed 1000 certificate retrievals. The minimum retrieval time was 0.156 seconds, while the maximum retrieval time was 1.813 seconds. If we ignore the top five times, our maximum retrieval time is 0.531. The extraordinarily high values (top 5) are likely caused by external factors such as other processes consuming system resources. The mean retrieval time was 0.216, while the median was 0.187 seconds. The certificate size with no added reputation information is 892 bytes. For a detailed look at the data set in a graph, see Figure 29 below.

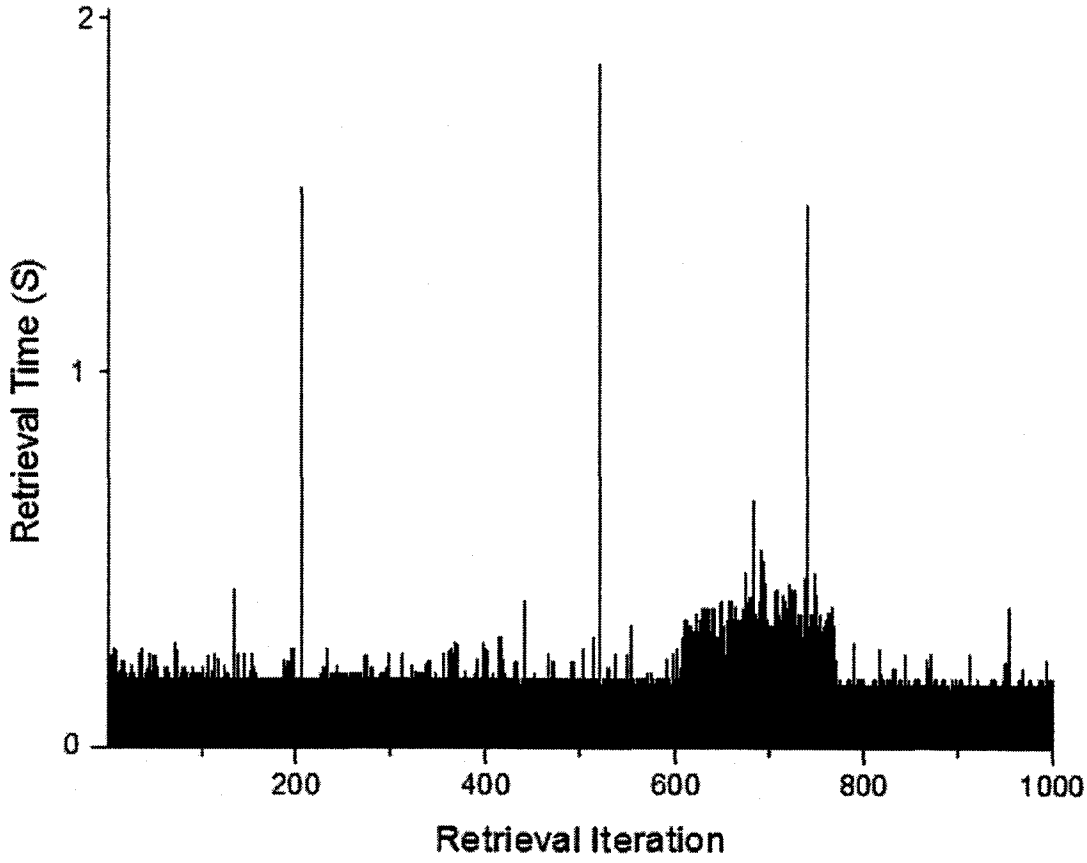


Figure 29: TEST A: 1000 Certificate Retrievals. No reputation in certificate.

7.3.2 Test B: Certificate Retrieval Time (EMPTY REPUTATION)

Next, we tested how the retrieval times would be affected if the certificate had the reputation xml in place, but it was empty. For this test, we created a new user in the CA. Note that the tasks involved with retrieving the certificate still do not involve any reputation extraction/processing. They simply involve the retrieval of a larger certificate. In this case, the minimum retrieval time was again 0.156 seconds, while the maximum retrieval time was 1.672 seconds. This maximum is actually less than with no reputation. If we ignore the top 5 values in this (empty reputation) case (1.516, 1.531, 1.562, 1.562, and 1.672), the maximum is 0.625 (the no reputation max. was 0.531). The median retrieval time was 0.172 seconds, and the mean, or average, retrieval time for a certificate with the empty reputation holding XML extension (averaged over 1000 retrievals) was 0.197 seconds. This involved a certificate size of 1248 bytes. Notice that this retrieval time is only slightly different from the empty certificate.

In this case, the performance of empty reputation is slightly better, than the performance of retrieval with no reputation. This is likely caused by external system factors such as other processes consuming other resources. We believe that the essence of these results indicate that the addition of empty reputation does not significantly increase the retrieval times.

For a detailed view of the results for Test B, see Figure 30 below.

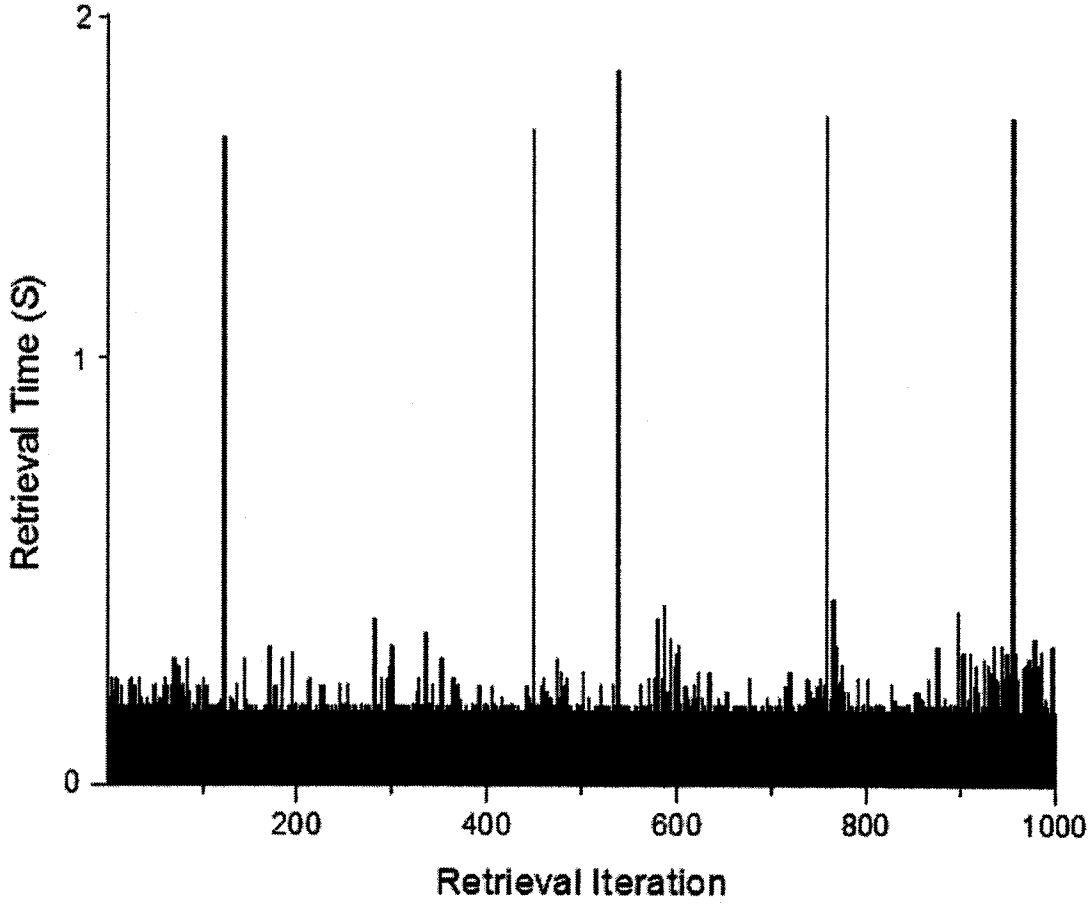


Figure 30: TEST B: 1000 Certificate Retrievals. Empty reputation in certificate.

7.3.3 Test C: Certificate Retrieval Time (0-100,000 ACTIONS)

As the number of actions increase within a user's reputation, the certificate size would be expected to increase (thereby increasing retrieval time) until the certificate was to reach the theoretical maximum size. In this section we illustrate the retrieval times for 1000 retrievals for a certificate with 100,000 actions contained as reputation. This is intended to represent the maximum (or very close to the maximum) size. This certificate has been used for 1000 previous jobs, each containing 100 actions. This means that there are 999 historical certificates for this user in EJBCA.

In the case of certificate retrievals, the minimum retrieval time was 4.141 seconds, while the maximum retrieval time was 7.5 seconds. If we ignore the top 5 values, the maximum is 6.296 seconds. The median retrieval time was 4.25 seconds, the mean, or average, retrieval time (over 1000 retrievals) was 4.536 seconds. This involved a certificate size of 1358 bytes. The retrieval times in this case are significantly higher than in Test A and Test B. This rapid increase in certificate retrieval time is not primarily caused by the slight increase in certificate size. EJBCA maintains all previous certificate versions, meaning each time the reputation is modified, a new certificate is created and the overall number of certificates stored increases by one (there are 999 old certificates in this case). This causes an increase in the overall retrieval time. If EJBCA did not process each historical certificate until it reaches the current certificate, it is our belief that the retrieval times would align with the times illustrated in Test A and Test B. For further details regarding Test C, see Figure 31 below.

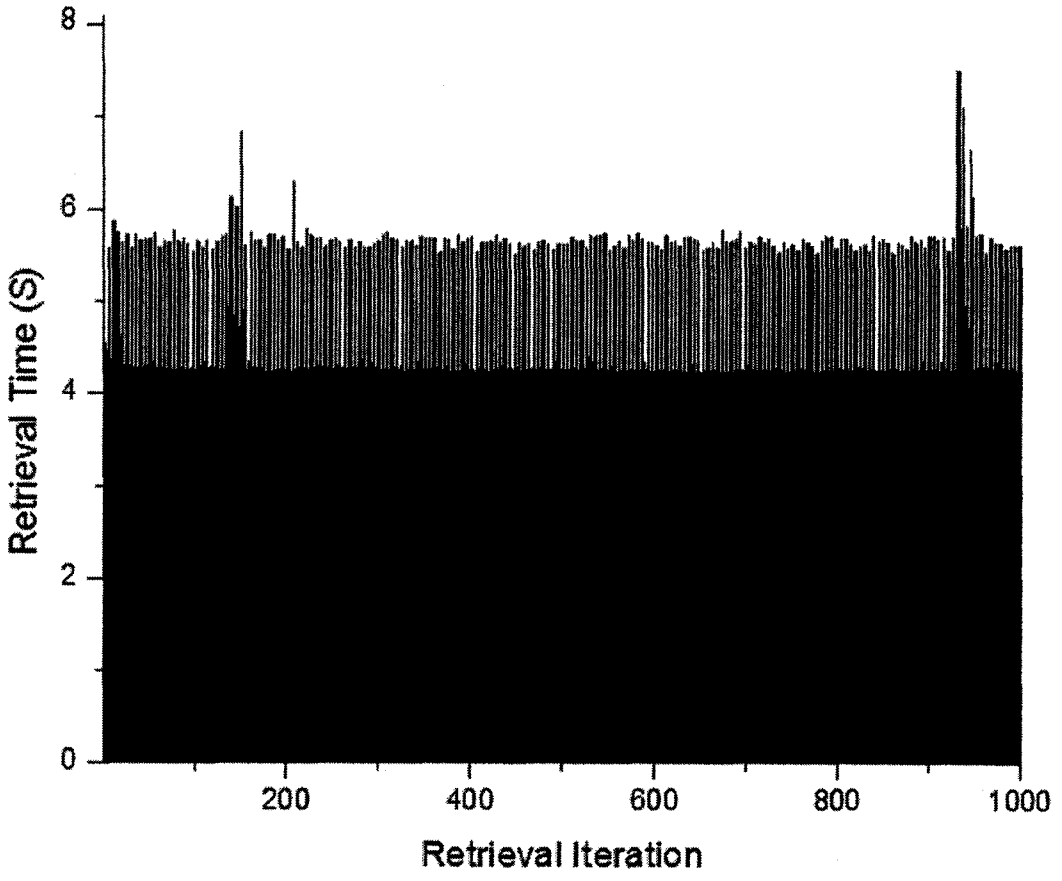


Figure 31: Test C. 1000 certificate retrievals. 100,000 actions in reputation.

7.3.4 Test D: Certificate Size Vs. Actions in Reputation

Test D is intended to illustrate the relationship between the number of actions within a user's reputation and the overall size of the certificate. This test involved the execution of 1000 jobs, each containing 100 actions. These jobs also included the authorization decision times, and certificate retrieval time, but only the individual certificate sizes and number of jobs will be discussed here. The authorization timing will be discussed in a later section.

The graph shows that the maximum certificate size (after 100,000 actions) is quite minimal. It also shows that this maximum size is reached with a very young reputation in the certificate, and stays that way after addition of many more actions. This is a proves the overall scalability of the design of the reputation format. It is not to say that this is the absolute maximum, but any changes in size will be minimal. Changes in size will occur, for example, when a category goes from 9,999 illegal read actions to 10,000 illegal reads, simply due to the requirement to store the additional digit. The previously discussed cert maximum of approximately 3.5K still applies, though it will take a significant number of actions to reach.

More details regarding the results from this test are available below in Figure 32.

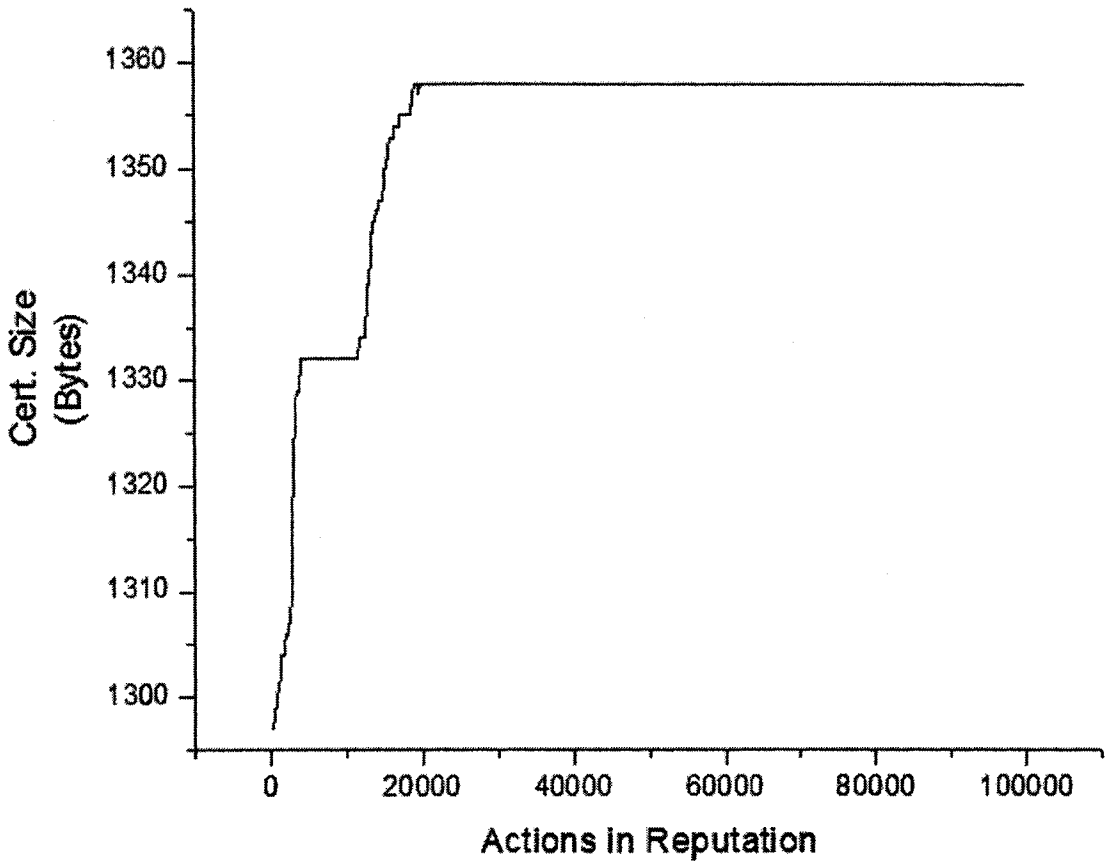


Figure 32: Test D. Certificate size increase with 100,000 actions in reputation.

7.3.5 Test E: Access Decision Time (0-100,000 ACTIONS)

This test is arguably the most important test of all. The AASUR system is intended to perform dynamic access decisions based on reputation, so it is fitting that we test the act of the making the decisions and record the associated timing information. For this test a new user was created. This user went through 1000 simulated jobs, each involving 100 actions (90/10 split). In the end (with a total of 100,000 actions in the user's certificate) the results are as follows. The minimum access decision time was 0.796 seconds, while the maximum access decision time was 3.828 seconds. If we ignore the top 5 values in this case, the maximum access decision time is 2.297 seconds. The median access decision time was 0.928 seconds, the mean, or average, retrieval time (over 1000 retrievals) was 0.852 seconds. This involved a maximum certificate size of 1358 bytes.

This test confirms our beliefs regarding the overall authorization decision performance as well as the overall scalability of this system. The required time to make an access decision remains fairly constant (linear complexity) regardless of reputation size. It also shows that the amount of time required to make an access decision is reasonable and is not going prevent this system from becoming a feasible architecture for distributed authorization. Again, the spikes in authorization decision time are likely due to external operating system factors, rather than a worst case scenario.

Further details are available regarding Test E in Figure 33 below.

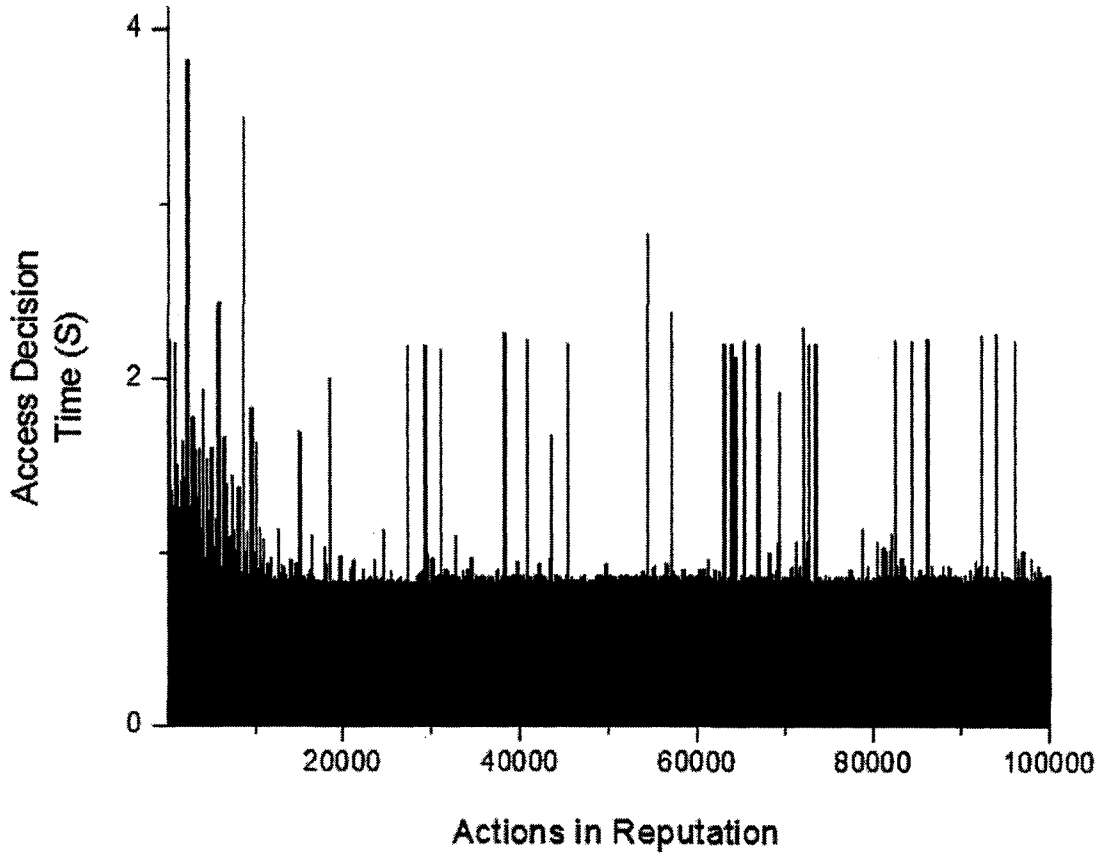


Figure 33: Test E. Authorization decision time (0-100,000 actions in reputation).

7.4 Performance Comparisons

Though the Tera binaries were available, the source codes were not. This made it impossible to insert any timing code into the application to determine any kind of results suitable for a performance comparison. We contacted the authors of Tera, but were told that no numeric results were available. [Private communication – May 16, 2007]

We did obtain results for two other authorization systems which were discussed in Chapter 3. These systems are Permis and Akenti. Results from these systems were compared in [Otenko2003]. There was no detailed raw data available however, their results were summarized. Those results are presented in following sections. For both systems we compared the time it takes to make an access decision. Further, Permis includes timing information which relates to credential retrieval so we compared that to AASUR timings as well.

It should be noted that the timing for an AASUR authorization decision includes the time required to send the certificate from the user site to the resource site, as well as all parsing, followed by the decision making process, ending with the return of the access decision. Further, the retrieval times for AASUR involve no caching. They all require a connection to a CA, as well as lookup/retrieval of the appropriate certificate.

7.4.1 Akenti

The Akenti timing information includes the access decision time as well as its standard deviation for two scenarios. The first scenario involves credentials that must be retrieved,

and the second scenario involves locally cached credentials. All of the following results obtained from [Otenko2003].

For the scenario where the credentials must be retrieved, the average authorization time of Akenti is 368.2ms, with a standard deviation of 40.1ms. On the other hand, if the credentials are already cached on the Akenti server, the average access time is 17.2ms with a standard deviation of 51ms.

7.4.2 Permis

The Permis timing information groups the access decision and the credential retrieval time into one. All of the following results obtained from [Otenko2003]. The average credential retrieval time for Permis is 142.217ms with a standard deviation of 289.842ms.

7.4.3 AASUR / Permis / Akenti Comparison Summary

The results obtained are promising within the right context. Although the average credential retrieval time for AASUR is higher than the times for both Akenti and Permis, this is due in part to the CA implementation in Java, as well as the previously mentioned historical certificates. It is our opinion that this retrieval time could be significantly lowered if these changes were implemented.

The access decision times for AASUR are slightly longer than both Akenti and Permis as well; however, the lack of requirement for nearly any administrative overhead, makes up for this.

Another promising aspect of these results is that both of these systems which involved multiple credential retrievals are just typical results. If a given situation required more credentials than a specific case (as tested) then the retrieval times would be greater. For AASUR, the access decision time essentially remains constant.

8 Conclusion

In this thesis we have introduced relevant tools and technologies as well as existing systems which try to solve the distributed authorization issue. We also describe the approach to authorization of unknown entities in the context of grid computing that we have chosen. This approach is implemented as a system called AASUR and entails new approaches which have not been used prior to this thesis.

The "new" in this thesis is the technique of defining and implementing a schema which defines a user's reputation. This schema makes use of a spectrum of metrics that essentially define what actions a user has performed in the past. Further, the notion of storing these metrics and counts in an identity certificate is also novel. Typically these are stored in a third party repository within an Attribute Certificate (AC) such as an Attribute Authority [Opplinger2000]. This introduces an increased number of messages, which consequently affects the overall performance of the system. However, the act of making an authorization decision based on information about a user is not new. In fact nearly all authorization is based on some specific information about a user. Although the way in which AASUR uses the information that it collects is in fact new, it is not based on any known statistical or other methodologies, so its use may be limited to proof of concept. We chose to use an arbitrary decision algorithm (RF, UC, ADE) simply to illustrate the usability of the information we collect for authorization decisions without requiring an in depth survey into a specific authorization method. Further, this system could have been implemented using one of many different authorization algorithms. In the future it is likely that such a system as ours could employ several pluggable modules, each defining

a specific authorization technique, enabling a significant degree of flexibility for the resource site.

We believe that not only has our architecture accomplished the necessary goals to overcome the issues stated within, but also, that it does so in such a way that the overhead required is acceptable. Although the results indicate that the retrieval times, as well as, the decision times, are generally slightly higher than those of Akenti and Permis, we believe that the consequent drop in administrative overhead is significant and valuable. We also believe that in the context of this problem, the maximum bounds of our various tests are still within an acceptable range, and do not preclude AASUR from being a feasible distributed authorization architecture.

8.1 Future Work

In this work a unique architecture has been presented which provides a distributed authorization capability which allows arbitrary entities to participate in the Grid while greatly improving scalability due to lower administrative overhead. Full completion of this system will require additional work. Currently, point actions are generated from a standalone application for testing purposes. In the future, we must develop a piece of middleware, which will run at each resource site, for monitoring the actions of the entity on the grid.

Failure and break-pointing of grid jobs must also to be taken into account. Current middleware has the mechanisms in place to resume processing on physically separate machines. Thus, the system must include the ability to maintain the actions performed

prior to failure on the initial node and to migrate seamlessly, to the new location of processing.

References

Abghour2001 – N. Abghour, Y. Deswartes, V. Nicomette, and D. Powell. An Internet Authorization Scheme Using Smart-Card-Based Security Kernels. ACM Lecture Notes In Computer Science; Vol. 2140. 2001.

Alfieri2003 - R. Alfieri, R. Cecchini, V. Ciaschini, L. dell'Agnello, Á. Frohner, A. Gianoli, K. Lõrentey and F. Spataro, "VOMS, an Authorization System for Virtual Organizations," European Across Grids Conference. 2003.

Alfieri2005 – R, Alfieri, R. Cecchini, V, Ciaschini, L. dell'Agnello, Á. Frohner, K. Lörentey, F. Spataro. From gridmap-file to VOMS: managing authorization in a Grid environment. Future Generation Comp. Syst. 21(4). 2005.

Ant - http://ant.apache.org/

Bacon2001 - J. Bacon, M.L., K. Moody. Translating Role-Based Access Control Policy within Context. Policies for Distributed Systems and Networks. 2001.

BEA – http://www.bea.com

Bhargava2002 - B. Bhargava and Y. Zhong, "Authorization Based on Evidence and Trust", in Proceeding of International Conference on Data Warehousing and Knowledge Discovery (DaWaK), Sept. 2002.

Bhargava2004 – B. Bhargava and L. Lilien. Formalizing Evidence and Trust for User Authorization. http://www.cs.purdue.edu/homes/bb/2004BostonIDM.html. 2004

Blaze99 - M. Blaze, J. Feigenbaum, and A.D. Keromytis. KeyNote: Trust management for public-key infrastructures. In Security Protocols---6th Int'l Workshop, Lecture Notes in Computer Science 1550. Springer, 1999.

Butler2000 - Butler, R., et al., A National-Scale Authentication Infrastructure. IEEE Computer. **33**(12). 2000.

Cecchini2004 - R. Alfieri, R. Cecchini, V. Ciaschini, L. dell'Agnello, A´ . Frohner, A. Gianoli, K. L"orentey, and F. Spataro. VOMS, an Authorization System for Virtual Organizations. 2004

Chadwick2003 - D.W.Chadwick, A. Otenko, and E.Ball. Implementing role based access controls using X.509 attribute certificates. IEEE Internet Computing. March 2003.

Curbera2001 – F. Curbera, W. Nagy, and S. Weerawarana. Web services: Why and how. In Workshop on Object Orientation and Web Services OOWS2001, 2001.

Dai2001 - J. Dai and J. Alves-Foss. Certificate Based Authorization Simulation System. In Proc. 25th Annual Int. Computer Software and Applications Conference. 2001.

Damianou2001 – N. Damianou, N. Dulay, E. Lupu, M. Sloman. "The Ponder Policy Specification Language". Proc. International Workshop of Policies for Distributed Systems and Networks. 2001.

Ejbca – The Enterprise Java Bean Certificate Authority: http://ejbca.sourceforge.net/features.html

Farrell2002 – S. Farrell, R. Housley. RFC 3281. An Internet Attribute Certificate Profile for Authorization. 2002.

Fiadeiro1995 - J.L. Fiadeiro and T. Maibaum. Verifying for reuse: foundations of object-oriented system verification. In I. Makie C. Hankin and R. Nagarajan, editors, Theory and Formal Methods. World Scientific Publishing Company, 1995.

Foster1998 - I. Foster, et al. A Security Architecture for Computational Grids. 5th ACM Conference on Computer and Communications Security. 1998.

Foster2001 - I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International Journal of High Performance Computing Applications, Vol. 15(3). 2001.

Foster2002 - I. Foster, "What is the Grid? A Three Point Checklist", Grid Today, Vol. 1, No. 6, 22 July 2002.

Foster2005 - I. Foster, H. Kishimoto, A. Savva, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, J. Treadwell, J. Von Reich. The Open Grid Services Architecture, Version 1.0. January 2005.

Globus2005 - Globus Security Team, Globus Toolkit Version 4 Grid Security Infrastructure: A Standards Perspective. Version 4, September 12th, 2005.

Herzberg2000 - A. Herzberg, Mihaeli, Y. Mass, D. Naor, and Y. Ravid, "Access Control Meets Public Key Infrastructure, Or Assigning Roles to Strangers," IEEE Symposium on Security and Privacy, Oakland, CA, May 2000.

Ibm2004 – IBM Alpha Works. Trust Establishment. 2004. http://www.alphaworks.ibm.com/tech/trustestablishment

Jajodia1997 - S. Jajodia, P.S., V.S. Subrahmanian. A Logical Language for Expressing Authorizations. IEEE Symposium on Security and Privacy. 1997.

JavaWorld – Performance Tests Show Java as Fast as C++. http://www.javaworld.com/javaworld/jw-02-1998/jw-02-jperf.html

Jax2006 – JAX-RPC. http://jax-rpc.dev.java.net.

JaxTypes – Types Supported by JAX-RPC. http://java.sun.com/j2ee/1.4/docs/tutorial-update2/doc/JAXRPC4.html

Jboss – Jboss Application Server: http://labs.jboss.com/jbossas/

Keahey2002 - K. Keahey, and V. Welch, "Fine-Grain Authorization for Resource Management in the Grid Environment", In proceedings, Grid Computing 2002.

Kesselman2002 - C. Kesselman, I. Foster, J. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Open Grid Service Infrastructure WG, Global Grid Forum. June 22, 2002.

Kirschner2004 - Beth A. Kirschner¤, Thomas J. Hacker¤, William A. Adamsony , Brian D. Athey. Walden: A Scalable Solution for Grid Account Management. 5th IEEE/ACM International Workshop on Grid Computing (Grid 2004), 5 July 2004.

Koehler2003 - Koehler, J., Srivastava, B. Web service composition: Current solutions and open problems. In: ICAPS 2003 Workshop on Planning for Web Services. 2003.

Kohl1991 – J. Kohl, B. Neumann. The Kerberos Network Authentication Service. 1991.

Lampson2000 - B. W. Lampson. Computer security in the real world. In Proc. Annual Computer Security Applications Conference (ACSAC), 2000.

Lepro2003 - Lepro, R., Cardea: Dynamic Access Control in Distributed Systems, NASA A dvanced Supercomputing (NAS) Division, 2003.

Lorch2004 – M. Lorch. "Privilege Management and Authorization in Grid Computing Environments". PhD Thesis, Virginia Polytechnic Institute and State University. 2004.

Mont2003 – M. Mont, A. Baldwin, J. Pato. Secure Hardware-based Distributed Authorization Underpinning a Web Service Framework. HP Laboratories Bristol. 2003.

Morgan2004 – R. L. Morgan, S. Cantor, S. Carmody, W. Hoehn, K. Klingenstein. Federated Security: The Shibboleth Approach. Educause Quarterly. November 2004.

Mostéfaoui2003 – G. K. Mostéfaoui, P. Brézillon. A Generic Framework for Context-Based Distributed Authorizations. Modeling and Using Context (CONTEXT-03), Lecture Notes in Artificial Intelligence, Vol 2680. 2003.

MySql – The MySql Database: http://www.mysql.com/

Opplinger2000 - R. Oppliger, G. Pernul, and C. Strauss. Using attribute certificates to implement role-based authorization and access controls. In S. T. K. Bauknecht, editor, Sicherheit in Informationssystemen (SIS 2000).

Orth2002 – Orth, G. The Web Services Framework: A Survey of WSDL, SOAP, and UDDI. Information Systems Institute. 2002.

Otenko2003 - Otenko S., Chadwick D. A comparison of the Akenti and Permis authorization infrastructures. Proceedings of the ITI First International Conference on Information and Communications Technology (ICICT 2003). Cairo University. 2003.

Pearlman2003 - L. Pearlman, C. Kesselman, V. Welch, I. Foster, and S. Tuecke. The community authorization service: Status and future. In Proceedings of the Conference for Computing in High Energy and Nuclear Physics, La Jolla, California, USA, Mar. 2003.

Rivington2004 - J. Rivington, R. Kent, A. Aggarwal, P. Preney. AASUR: A Distributed System for Authorization of Unfamiliar Entities in a Service Oriented Architecture. WSEAS TRANSACTIONS on COMPUTERS, Issue 9, Volume 4, September 2005.

SAML2005 – Oasis. Assertions and Protocols for the OASIS Security Associations Markup Language (SAML) Version 2.0. 2005. http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf

ShaikhAli2003 - A. ShaikhAli, O. Rana, R. Al-Ali, and D. Walker, "UDDIe: An extended registry for web services," in Proceedings of Workshop on Service Oriented Computing: Models, Architectures and Applications at SAINT 2003.

Tang2004 – Definition – Pervasive Computing: http://searchnetworking.techtarget.com/sDefinition/0,,sid7_gci759337,00.html

Thompson1999 - M. Thompson, W. Johnston, S. Mudumbai, G. Hoo, et al., "Certificate-based Access Control for Widely Distributed Resources," Proceedings of the Usenix Security Symposium, Aug. 99.

Thompson2003- M.Thompson, A. Essiari, S. Mudumbai , "Certificate-based Authorization Policy in a PKI Environment", ACM Transactions on Infomation and System Security (TISSEC), Volume 6, Issue 4. November 2003.

Welch2002 - Welch, V., Pearlman, L., Foster, I., Kesselman, C., Tuecke, S., "A Community Authorization Service for Group Collaboration", 2002 IEEE Workshop on Policies for Distributed Systems and Networks.

Welch2003 - V. Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Czajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman, and S. Tuecke. Security for grid services. In HPDC, 2003.

Welch2005 - Von Welch, Tom Barton, Kate Keahey, and Frank Siebenlist. Attributes, Anonymity, and Access: Shibboleth and Globus Integration to Facilitate Grid Collaboration. Proceedings of the 4th Annual PKI R&D Workshop, April 2005.

W3C1 – World Wide Web Consortium. Extensible Markup Language (XML) 1.0, Fourth Edition. 2006. http://www.w3.org/TR/REC-xml/

W3C2 – World Wide Web Consortium. Web Services Description Language (WSDL) 1.1. 2001. http://www.w3.org/TR/wsdl

W3C3 – World Wide Web Consortium. Simple Object Access Protocol (SOAP). Version 1.2. 2003. http://www.w3.org/TR/soap/

WEB - http://www-306.ibm.com/software/websphere/

WSS2004 – Web Services Security: SOAP Message Security 1.1 (WS-Security 20004). OASIS Standard Specification. February 2006. http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf

Yao2003 – W. Yao, "Trust Management for Widely Distributed Systems", PhD Thesis, University of Cambridge. 2003.

Appendix A: Environment Configuration Instructions

In this appendix we provide details of how to install and configure the AASUR system. In order we discuss Linux, JBOSS, MySQL and EJBCA installation and configuration and subsequent preparation and deployment of the complete system. All steps are enumerated, from beginning to end, in order to clarify the entire procedure.

A.1 LINUX INSTALLATION

1. place bootable dvd image of Fedora Core 6 in dvd rom drive
2. when prompted hit enter for default installation
3. be sure to test media prior to beginning installation to ensure all necessary are intact
4. when selecting packages to install, include mysql, ant, and samba
5. apply a applicable updates to system
6. configure a samba share via for installation file transfers to machine
7. configure mysqld and smb to start automatically on boot
   - chkconfig --level 3 mysqld on
   - chkconfig --level 5 mysqld on
   - chkconfig --level 3 smb on
   - chkconfig --level 5 smb on
8. verify configuration via chkconfig --list and manually start services this time or reboot
9. update Java to 1.5.0
   - import JPacakge keys: rpm --import http://www.jpackage.org/jpackage.asc
   - download (from sun) jdk-1_5_0_09-linux-i586-rpm.bin
   - cd /etc/yum.repos.d
   - wget http://jpackage.org/jpackage.repo
   - edit (vi) the /etc/yum.repos.d/jpackage.repo file and changing line(s) enabled=1

to enabled=0

- go to file share where jdk-1_5_0_09-linux-i586-rpm.bin is
- set executable permissions on java bin: chmod 711 jdk-1_5_0_09-linux-i586-rpm.bin
- execute java bin: ./jdk-1_5_0_09-linux-i586-rpm.bin
- download: java-1.5.0-sun-compat-1.5.0.09-1jpp.noarch.rpm
- install java-1.5.0-sun-compat-1.5.0.09-1jpp.noarch: rpm -ivh java-1.5.0-sun-compat-1.5.0.09-1jpp.noarch.rpm
- alternatives --config java and choose 1.5
- verify java version with: java -version (should be 1.5.0)

A.2 JBOSS INSTALLATION

10. copy jboss-4.0.3SP1.tar.gz to /home/jboss
11. go to /home/jboss and unzip jboss-4.0.3SP1.tar.gz: gunzip jboss-4.0.3SP1.tar.gz
12. unpack jboss-4.0.3SP1.tar: tar -xvf jboss-4.0.3SP1.tar
13. now we have a /home/jboss/jboss-4.0.3SP1
14. change directory to jboss-4.0.3SP1: cd /home/jboss/jboss-4.0.3SP1
15. add jboss group: groupadd jboss
16. add jboss user: useradd -g jboss -p jro1979 jboss
17. change directory to /home
18. change ownership of all jboss dirs/files: chown -R jboss jboss*
19. change group of all jboss dirs/files: chgrp -R jboss jboss*
20. change associated jboss file/dir permissions: chmod -R 771 jboss*
21. set appropriate anvironment variables in /root/.bashrc (see included file)

22. copied startup script into /etc/init.d (from included file) to /etc/init.d/jboss

23. added necessary symbolic links to jboss startup script in init.d (to various run levels)

- ln -s /etc/rc.d/init.d/jboss /etc/rc3.d/S84jboss
- ln -s /etc/rc.d/init.d/jboss /etc/rc5.d/S84jboss
- ln -s /etc/rc.d/init.d/jboss /etc/rc4.d/S84jboss
- ln -s /etc/rc.d/init.d/jboss /etc/rc6.d/K15jboss
- ln -s /etc/rc.d/init.d/jboss /etc/rc0.d/K15jboss
- ln -s /etc/rc.d/init.d/jboss /etc/rc1.d/K15jboss
- ln -s /etc/rc.d/init.d/jboss /etc/rc2.d/K15jboss

24. turn on jboss automatically as a service on boot

- chkconfig --level 3 jboss on
- chkconfig --level 5 jboss on

25. reboot and verify jboss functionality by going to http://localhost:8080

A.3 MYSQL CONFIGURATION

It is assumed that MySQL was selected to be installed during Linux install. This section deals only MySQL configurations.

26. create new database: create database ejbca (at mysql prompt)

27. secure the DB

- UPDATE user SET Password=PASSWORD('new_password') WHERE user='root';
- FLUSH PRIVILEGES;
- DELETE FROM user WHERE User = '';

- FLUSH PRIVILEGES;
- GRANT USAGE ON ejbca.* TO 'ejbca'@'127.0.0.1' IDENTIFIED BY 'jro1979';
- FLUSH PRIVILEGES;
- DROP DATABASE test;
- DELETE FROM db WHERE (db.Db LIKE 'test%');
- DELETE FROM db WHERE (db.Host = "%");
- DELETE FROM db WHERE (db.User = "");
- DELETE FROM user WHERE ((user.Host = "%") OR (user.User = ""));
- commit;
- FLUSH PRIVILEGES;
- commit;
- GRANT ALL PRIVILEGES on ejbca.* to ejbca@localhost IDENTIFIED BY 'jro1979' WITH GRANT OPTION;

28. put mysql-connector-java-3.1.12-bin.jar into:
    - $JBOSS_HOME/server/default/lib
    - $JAVA_HOME/jre/lib

A.4 EJBCA INSTALLATION

29. copied ejbca source to /root/ejbca (all modified sources included)
30. modified ejbca.properties to suite this install (see included file)
31. go to /root/ejbca:
    - ant bootstrap
    - ant install

32. stopped JBoss: service jboss stop

33. go to /root/ejbca:

    - ant deploy

34. started JBoss: service jboss start

35. go to /root/ejbca: ./bin/ejbca.sh ca getrootcert SHEMP_AASUR_CA ca.crt -der

36. keytool -import -trustcacerts -alias SHEMP_AASUR_CA -keystore \

    $JAVA_HOME/jre/lib/security/cacerts -storepass changeit -file ca.crt

37. restarted JBoss: service jboss restart

38. in /home/ejbca: ./bin/ejbca.sh setup setbaseurl server_ip ejbca

A.5 PREPARATION AND DEPLOYMENT

39. added bouncy castle jar to classpath: bcprov-jdk15.jar

    (located in /home/jboss/jboss-4.0.3SP1/server/default/lib)

40. compiled aasur-common package src and created a jar with internal dir structure of

    /org/aasur/common

41. added aasur-common.jar to classpath

    (located in /home/jboss/jboss-4.0.3SP1/server/default/lib/aasur-common.jar)

42. refreshed classpath with bash

43. compiled aasur-reputation package src and created a jar with internal dir structure of

    /org/aasur/reputation

44. added aasur-reputation.jar to classpath

    (located in /home/jboss/jboss-4.0.3SP1/server/default/lib/aasur-reputation.jar)

45. refreshed classpath with bash

46. compiled aasur-simulation package src and created a jar with internal dir structure of

/org/aasur/simulation

47. added aasur-simulation.jar to classpath

(located in /home/jboss/jboss-4.0.3SP1/server/default/lib/aasur-simulation.jar)

48. refreshed classpath with bash

49. added ejbca-ejb.jar to classpath

(located in /home/jboss/jboss-4.0.3SP1/server/default/lib/ejbca-ejb.jar)

50. refreshed classpath with bash

51. added commons-fileupload-1.0.jar to classpath

(located in /home/jboss/jboss-4.0.3SP1/server/default/lib/commons-fileupload-1.0.jar)

52. refreshed classpath with bash

53. restarted server: service jboss restart

54. installed Java Web Services Developer Pack 2.0

- downloaded jwsdp-2_0-unix.sh from Sun (included)
- not good for linux, so required some modifications
  - tail -n +368 jswdp-2_0.sh > jwsdp.jar
  - extraced all files from jwsdp.jar to a folder called JWSDP (temp folder)
  - added . to classpath
  - ran jwsdp.class using: java jwsdp
  - procede normally (typical installation in /usr/java/jwsdp-2.0)
  - moved all files in /usr/java/jwsdp-2.0/jaxp/lib and /usr/java/jwsdp-2.0/jaxp/lib/endorsed into /usr/java/jdk1.5.0_09/jre/lib/endorsed
  - added /usr/java/jwsdp-2.0/jaxrpc/bin to PATH
  - refreshed environment with bash

55. added jboss-jaxrpc.jar to classpath (located in in /home/jboss/jboss-4.0.3SP1/server/default/lib/ejbcaAasurService-ejb.jar)
56. refreshed classpath with bash
57. added jaas.jar to classpath (/home/jboss/jboss-4.0.3SP1/server/default/lib/jaas.jar)
58. refreshed classpath with bash
59. copied all code/build files for AasurCaSignService to /root/ca
60. compiled all ca source (from within /root/ca) javac *.java
61. moved all classes to /root/ca/org/aasur/ejbca/ca
62. built jar file (from /root/ca: jar -cvf ejbcaAasurService-ejb.jar ./org)and then added to classpath (in /home/jboss/jboss-4.0.3SP1/server/default/lib/ejbcaAasurService-ejb.jar)
63. refreshed classpath with bash
64. restarted server: service jboss restart
65. generated wsdl and mapping files for web service using config.xml (specific to machine ip) in /root/ca command: wscompile.sh -define -mapping mapping.xml -d . –nd . -classpath /home/jboss/jboss-4.0.3SP1/server/default/lib/ejbcaAasurService-ejb.jar config.xml
66. moved wsdl and mapping files into WEB-INF folder, the wsdl into wsdl folder (all within /root/ca)
67. built/deployed aasur ca service: ant all (from within /root/ca)
68. restarted server: service jboss restart
69. copied all code/build files for resource site service listener to /root/resource
70. compiled all resource site source (from within /root/resource) javac *.java
71. moved all classes to /root/resource/org/aasur/resource

72. built jar file (from /root/resource: jar -cvf resourceSiteServiceListener.jar ./org)

73. added resourceSiteServiceListener.jar to classpath

    (located in in /home/jboss/jboss-4.0.3SP1/server/default/lib/resourceSiteServiceListener.jar)

74. refreshed classpath with bash

75. restarted server: service jboss restart

76. generated wsdl and mapping files for web service using config.xml (specific to machine ip) in /root/resource

    command: wscompile.sh -define -mapping mapping.xml -d . -nd . -classpath /home/jboss/jboss-4.0.3SP1/server/default/lib/resourceSiteServiceListener.jar config.xml

77. moved wsdl and mapping files into WEB-INF folder, the wsdl into wsdl folder (all within /root/resource)

78. built/deployed aasur resource site service: ant all (from within /root/resource)

79. restarted server: service jboss restart

Vita Auctoris

Jordan Rivington was born in Windsor, Ontario, Canada in 1979. In 1997 he graduated from St. Thomas of Villanova in LaSalle, Ontario. From there, he pursued a post secondary education at the University of Windsor where he remained for almost 11 years. He first obtained a Bachelor of Science from the School of Computer Science with Honours. This was followed by a Masters of Science, also at the University of Windsor, dealing specifically with Grid Computing and Security. Currently he is an instructor for the ITT Technical Institute School of Computer Networking Technologies, as well as the School of Software Application Programming.