Policy Management in a Distributed Computing Environment

Russ Wakefield
Colorado State University

Abstract

Management of the security of resources owned by an organization has increasingly moved to policy-based systems. Using a common policy management mechanism allows the modification of the policies without rewriting the underlying management system. This is especially prevalent in large scale systems such as grid computing, peer-to-peer systems, cluster computing, and other forms of distributing computing.

Distributed computing has continued to evolve as the central site solutions become untenable. Web-based environments flourish where inexpensive processors are easily available and dominate the computing space. Scientific computing is progressing into a space where the hundreds or thousands of systems solve problems rather than huge and expensive supercomputers. As distributed computing evolves, security within the distributed environment becomes more complex leading to the need for using the commonality inherent in policy management.

Resources must be able to be managed and access to those resources controlled. Managing the access control policies for that environment becomes a key component for success. Many organizations have begun to look at this problem and present policy management systems as a solution.

To be successful within the distributed computing environment, these systems should be able to span multiple administrative domains, to support replication and redundancy to resolve scalability and fault tolerance issues. They should be able to dynamically update their policies as well as allow multiple entities to participate within the decision process.

This paper presents an analysis of four such access control policy management system, Ponder, XACML, Akenti, and dRBAC. Contrasting them using the criteria of the desired features for distributed computing environment allows us to identify strong contenders for use within a distributed or grid environment.

Keywords

Access control, distributed computing, policy languages, policy management, security, grid computing

1. Introduction

A security policy defines the activities that are acceptable and/or unacceptable within a defined framework, such as an organization or a physical locality. These policies apply to all phases of the framework, including physical (who gets keys to what doors), logical (who gets to see what data), and temporal (when can actions within the framework occur).

Security policies pertaining to information that define what actions specific people may or may not perform upon the information are called access control policies. Several access control mechanisms have been created over the years, including mandatory access control (MAC) [1], discretionary access control (DAC) [2], role-based access control (RBAC) [3], and usage-based access control (UCON) [4].

A policy is not a single instance of a command, rather the scripts or definitions that are applied on a consistent basis. Policies indicate the decision paths that are to be taken based on a set of conditions and obligations that exist at the time of the decision point.

Bringing commonality to the expression of access control policies is accomplished through the use of policy languages such as XACML [5], SPL [7], and Ponder [6]. These policy languages were originally created to solve specific issues – but like all languages, are becoming more generic as the next generation is being created.

Implementing access control in a distributed computing environment adds additional issues for policy management.

In a heterogeneous environment, different architectures, resources, and administrative organizations lead to policies that may have a common purpose, but have different expression or syntax. Issues as mundane as a lack of support for tool chains (compilers, parsers, etc.) prevent commonality from being achieved.

Distributed systems may be grouped in clusters, as in grid computing, each cluster may be managed autonomously. If we use the example of a virtual organization (VO) [8], these clusters may exist within different administrative domains. Localized policies may differ from those intended for use in the cluster environment. Reconciliation leading to a common base set policies is required for a security management system to work.

This paper presents four available options for policy management within a distributed computing environment – Ponder, XACML, Akenti, and dRBAC. Section 2 provides an overview of each of the policy management systems, a review of the concepts behind each of systems, and a description of the architectures. Section 3 contrasts the packages for desirability within a distributed computing environment. Section 4 contains the related works and Section 5 provides the conclusion.

2. Example implementations

2.1 Ponder

Ponder [6] is a policy specification language for distributed systems management, as well as a set of tools to support the language. It was created by the policy group at the Imperial College in London, currently headed by Prof. Morris Sloman [9]. Ponder is an object-oriented, declarative language that is derived from earlier work on policy specifications accomplished at Imperial College [10]. The tool chain includes a compiler, a policy editor, and a management kit for adding tools to the central console (such as a configuration management tool).

The compiler supports the features of the language grammar and includes a syntax analyzer, a two-pass semantic analyzer, and a java code generator for obligation and refrain policies. The compiler is intended to be generic enough to allow additional code generators to be added dynamically to support additional back-ends. Taking in the policy specification, the compiler creates a set of code that is fed to the backend to create.

The policy editor is a cross-platform editor that understands the syntax of the Ponder language. It provides syntax highlighting and various library features to assist the user. It is open-source and GUI –based.

2.1.1 Policy definitions

Ponder defines a policy as a rule that can be used to change the behavior of a system. The definition of policy separate from the manager allows the policies to be changed without making extensive changes to the underlying manager. Several policy types include authorization policies, obligation policies, refrain policies, delegation policies, composite policies, constraints, and meta-policies.

Authorization policies define what actions a subject may or may not do regarding a set of objects. Obligation policies define the conditions a subject must meet to interact with

an object. The verification of these policies is event-driven; the policies interpreted by an agent of the manager. Refrain policies what a subject cannot do with respect to an object – similar to negative authorization. Delegation policies define which actions a subject can delegate to another subject. Composite policies group a set of policies within a common scope to simplify the management of large-scale systems. Constraints are specified to limit the applicability of policies based on temporal or value-based attributes. Meta-policies define allowed values for attributes within a policy and which policies may co-exist with each other in a system.

2.1.2 Domains

Domains are akin to the organizational structure to which the policies are being applied. They provide a mechanism for grouping objects according to some criteria such as geographic, object category, responsibility, authority, or a logical grouping convenient to the members of the domain.
Becoming a member of a domain requires an explicit definition within the language framework. The framework is not inclusive of the objects – it merely contains references to the objects it contains – similar to links with a file system. These links can point to any type of object, including another subject. Objects can be a part of multiple domains (much like the file link analogy) allowing domains to overlap. Domains may also be sub-domains of another domain. Policies may have inheritance and propagate to sub-domains.

Organizational positions define an organizational structure, outlining the hierarchical structure of the organization to which it is being applied. These positions or roles have tasks and responsibilities associated with them – similar to the concepts contained in RBAC [3]. The abstracts the person from the role and allows personnel movement without extensive recoding. The authorization policies define the responsibilities of the role, role relationships define policies about the interaction between roles. The hierarchical organization positions allow definition of the organizational unit called management structures.

2.1.3 Architecture

Ponder contains several components that make up it's management system. It uses JAVA RMI as a middleware communication mechanism during runtime. The management console talks to both the compiler and RMI, while the policy editor previously described understands the syntax and feeds into the compiler. A user-role management tool defines and stores the user to role relationships in the domain service.

The domain service is implemented using LDAP, Lightweight Directory Access Protocol, an attribute-based database system used primarily as a directory service. The attributes stored into the LDAP server are read to provide information to the, enforcement components, the policy service, and to the policy control objects.

The policy service creates a policy control objects for every policy defined by the language. These policy objects are stored in LDAP server and communicated throughout the system using the Java RMI communication middleware.

The domain browser is a user interface to show the management of all the objects stored in the domain service. Objects can be selected and properties opened and modified using the domain browser. Objects can be grouped into sub-domains, can be monitored as well as performing management operations.

2.2 XACML

eXtensible Access Control Markup Language (XACML) [5] is a XML standard created by OASIS (Organization for the Advancement of Structured Information Standards), a not-for-profit open standards consortium made up 5000 individuals in 600 countries according to their website [11]. Their focus is Web standards with additional work in security and e-business.

The purpose of the creation of XACML is "define a core XML schema for representing authorization and entitlement policies" [5].

Because so many of the access control languages were manager-specific, the tools necessary to manage the policies were not generic. In section 2, we detailed the problems arising from manger-specific systems in a distributed computing environment. The consortium behind XACML focuses on exactly that issue.

Version 1.0 of the OASIS standard for XACML was released in February, 2003 with the current version 2.0 released in 2005. Version 3.0 is currently under development.

There are six profiles released with 2.0, SAML 2.0, XML Digital Signature, Privacy Policy, Hierarchical Resource, Multiple Resource, and Core and Hierarchical Role Based Access Control (RBAC). SAML is the Security Assertion Markup Language used for authorization decision querying.

2.2.1 Policies

XACML defines the syntax of a policy language and the semantics for enforcing those policies. The protocol of a request/response query is defined, as well as the semantics for checking the appropriateness of the request with respect to the policy. Included in the standard is the use of arbitrary attributes as a part of the request/response system.

XACML uses a hierarchical approach with each policy consisting of an arbitrary set of sub-policies. Each of the trees defines the actions for a target with the leaves consisting of a set of rules about the target. The query system checks the values of the attributes as a part of the Policy Decision Point (PDP) passing information to the Policy Enforcement Point (PEP). [12] The components of a request include the subject attributes, the id of the object, the action and the environment. The reply is one of four decisions, accept, deny, not applicable, or error.

A policy is established through the use of pre-defined data types and interfaces to functions defined by the standard. These pre-defined attributes and functions implement most access control policies. In addition, XACML has the ability to define extensions to handle access control functions and data types not defined by the standard.

Policies can also reference other policies. This allows the rules to be created that are not specific to a policy and used by many policies. Part of the language includes the ability to combine results from several operation/queries and combine them into a single result. This includes a mechanism to define new algorithms.

XACML also has the ability to add functionality at the PDP. This allows the PDP to function with additional criteria, such as an existing access control system. This is done by identifying attributes that are unknown to the attribute system and allowing them to be fulfilled using an external or legacy system.

2.2.2 Architecture

XACML uses the concept of a Policy Enforcement Point (PEP) as a gateway through which a user must request access to a resource. This PEP then makes a request to the Policy Decision Point (PDP) and receives one of the responses described in 2.2.1 – accept, deny, N/A, or error. The PDP can also specify obligations that the PEP must enforce on behalf of the resource. These obligations are obtained from the policy definitions. Based on that response, the user is then permitted or denied access to the resource.

The PDP can pull from multiple sources to provide the decision. It can fetch attributes from using one of two mechanisms that are a part of the system. These mechanisms are AttributeDesignators, which reference values by metadata, and AttributeSelectors, which use XPath expressions to find values). XPath is a language for finding parts of an XML document, in this case the policy definitions.

If the attributes are not satisfied by either of these mechanisms, the PDP is free to look elsewhere. This would allow attributes to be retrieved from a legacy system or from another modern access control system.

Like Ponder, XACML focuses primarily on the language definition. Sun has implemented an open source toolkit utilizing the XACML standard in Java. According to the Sun website, "There is full support for parsing both policy and request/response documents, determining applicability of policies, and evaluating requests against policies. All of the standard attribute types, functions, and combining algorithms are supported, and there are APIs for adding new functionality as needed. There are also APIs for writing new retrieval mechanisms used for finding things like policies and attributes." [18]

2.3 Akenti

Akenti is an access control system for grid computing environments created at the Distributed Systems Division of the NERSC Division at Lawrence Berkeley National Laboratory [15]. Grid computing is defined as coordinated resource sharing and problem solving within a dynamic, multi-institutional, virtual organization [8]. Grid computing theory indicates that resources that are unused within an environment could be added to the greater good. The promise of massively parallel systems has been with us for years – but the implementation of this promise is just beginning to evolve within the grid computing environment.

The goals for Akenti include that control over objects should be as decentralized as possible to reflect the multi-organization structure of distributed environments, delegation of authority to control access in a restricted and hierarchical fashion, inheritance of restrictions from parents, and protecting the content of the data not the structure.

Using X.509 certificates in a grid environment provides a virtual organization-wide user authentication system while allowing the owners of the resources (called stakeholders) to control access within their domain [14]. This is ideal for grid computing where control over the grid resources is desired by the resource owners.

2.3.1 Certificates

Akenti makes use of digitally signed certificates that embed user identity authentication, resource usage requirements, user attribute authorizations, and delegated authorizations. Authorization decisions are accomplished by both on-line and off-line entities [13]. Certificates are issued by a Certificate Authority (CA) – a trusted authority run on a secure machine that issues the signed certificates.

An **identity certificate** is a document signed by a trusted Certificate Authority that contains the Distinguished Name and the public key of a user. Akenti uses the CN (common name), OU (organizational unit), O (organization), and/or C (country). An attribute certificate verifies that a particular user (or list of users) identified by the Distinguished Name possesses a value for a given attribute. It may also contain an optional condition controlling the attribute validity.

A **use certificate** specifies the conditions that must be satisfied for an operation to occur on a named resource. All of the use-conditions define the group of entities that are permitted to access an object, i.e. an ACL. Use Condition certificates are created and signed by the resource stakeholders and contain the resource name, attribute names and values that a user must satisfy, actions allowed by satisfying the condition, CAs to be trusted to verify users, issuers to be trusted for attribute certificates.

A **policy certificate** is a self-signed certificate normally stored with the protected resources. It lists the trusted Certificate Authorities and the stakeholders and where their Use Condition certificates for the resource are stored. Each certificate contains a list of trusted CA's and their public key, places to search for Attribute Certificates, acceptable UCC issuers, URLs for Use Condition Certificates, and the maximum certificate cache lifetime.

2.3.2 Architecture

The Akenti system uses a Policy Enforcement Point that is referenced through a resource gateway to access resources. The **stakeholders,** those that control the resources, identify the constraints as a set of certificates that can be stored both on the gateway and on a number of remote machines. Contained in these certificates are the values of the attributes the user must have to access the resource. This requires the use of a trusted resource to provide authentication information about the user. This trusted resource, referred to as an **Akenti server**, implements the PDP by gathering the appropriate certificates and verifying that the requesting user has the necessary permissions and obligations to acquire the resource.

Policies are communicated using XML stored in the certificates. If a stakeholder wants to impose conditions on the use of a resource, that stakeholder issues a use-certificate describing those conditions. A user is identified by an identity certificate, verifying that the user is who they say they are. Finally, the policy certificate identifies the appropriate certificate authorities for a given resource.

2.4 dRBAC

Distributed Role-based Access Control (dRBAC) was designed to be "a scalable, decentralized trust-management and access control mechanism for systems that span administrative domains" [16]. dRBAC uses PKI identities to define trust domains, traditional RBAC roles to for access control, and role delegation across domains to map permissions to these activities.

dRBAC was created to focus on access control with a "coalition environment". These environments are akin to VOs [8] in their makeup – corporations that form partnerships, multi-nation government projects. The primary point of identity is the unwillingness of the participants to rely on a third party to administer the trust relationships. This unwillingness forces cooperation between the entities to share the resources while still denying access to private resources. This is akin to the issues raised in grid computing or in network-based services – an occurrence that is becoming more common as the technology becomes widely used.

Traditional RBAC systems rely on a centralized trust system administered by a central authority. This doesn't work in a distributed environment, because in many situations this central authority does not exist. Instead, each organization manages their own resources based on a common, agreed upon set of rules.

Like RBAC, dRBAC uses permissions assigned to roles to control actions. Roles are defined within the namespace of the organization managing the resources – these roles can then be delegated to other roles within the same namespace or within other organizational namespaces. PKI certificates are used to identify all the entities associated with the transactions. Associating roles to namespaces that are defined by keys allows the policies to be enforces within the namespace.

2.4.1 Delegations

dRBAC uses delegations to accomplish much of the functionality necessary within the distributed environment. Third-party delegations provide a second level of delegation, allowing an organization to be granted the ability to delegate access controls. This relationship, known as **speaks for**, authorizes a principal to act as a grantor in a given role.

Permissions can also be delegated transitively. If an organization is granted a set of permissions, it may also be given the ability to further grant those permissions to others, depending on how those permissions were granted. A sequence of delegations from a subject to an object is called a "delegation chain".

Proofs are graphs of delegations from a subject to an object. **Proof monitors** are used to monitor the validity of delegations within a prolonged trust relationship. These monitors

verify that the proofs exist to show that a subject has the permission to access an object.

2.4.2 Architecture

dRBAC uses the concept of **wallets** to store delegations. Wallets are objects and methods used to implement the **delegation chains**. These wallets support three basic operations to accomplish this:

- **Publications of delegations.** When a new delegation is issued, a corresponding entry is made in the wallet. This includes support proofs from third-party delegations – the wallet does not have to do searches to establish the delegation chain.

- **Authorization queries**. Resources can query the wallet object for information to support the PDP. These queries include direct queries (can a subject access an object given a set of constraints), object queries (give me the proofs applicable to an object and a set of constraints), and subject queries (give me a set of proofs applicable to a subject and a set of constraints).

- **Proof Monitoring.** As described in 2.4.1, proof monitoring continues to validate the proof after it has been returned through the use of a callback mechanism.

Discovery tags are used to facilitate searches through multiple wallets. These tags include the subject's home walled, the required role, a TTL field, and two discovery search flags – one for subjects, one for objects. Cache invalidation is accomplished through the use of a push mechanism to inform subscribers of invalidated delegations. The invalidation is done through the use of the previously described callback mechanism; the notified subscriber can then alter the trust relationship, or discontinue the relationship.

3. Contrasting implementations

Distributing computing offers many additional challenges that are not seen when working within the centralized computing environment. With a centralized system, there is a single, common authoritative PDP, often implemented in the operating system kernel which can be referenced for access control. Following are some of the issues that are applicable within a distributed computing environment:

- Resources may or may not be owned by a common organization. When there are multiple owners, there may be multiple policies. Reconciliation would be required to resolve this.
- There may be many administrative domains that have policies must be reconciled when determining access control. In addition, these domains may want to participate at the PDP; the policy management systems should be able to support that.
- It is often not possible to perform a restart of the entire system within a distributed computing environment; modifications to the policies must be able to be accomplished dynamically.
- Scalability within a distributed system must be addressed. If hundreds of systems are being utilized, having a single authorization or authentication server will often cause bottlenecks. Features such as support for redundancy and replication are required to resolve these issues.

3.1 Multiple domains

Multiple owners and/or multiple administrative domains lead to the issue of how to reconcile conflicting policies within a specific environment. In general, this is function of the PDP. If the PDP is executed locally, the control of reconciliation is also local. If the PDP is executed centrally, the decision for reconciliation is done by a central server.

Ponder forces the reconciliation to occur at the time of compilation – effectively forcing a local PDP. Whatever entity is using the compiler would reconcile the discrepancies and resolve

them locally. The Sun implementation of XACML has the ability to have the resolution occur either locally or as a network service. This is a strength allowing the application developer to chose based on his requirements. Akenti implements the PDP within the Akenti servers – resulting in a centralized decision. There may be multiple instances of the servers, but they are evaluating the same data. dRBAC evaluates the certificates locally, resulting in a local decision.

3.2 Dynamic update

Distributed systems can rarely be simultaneously updated, the ability to update their policies dynamically is necessary for reliable service. In the case of our packages, being able to dynamically update is an ambiguous term. The access control system may be offered as a service. Restarting the service would have little effect on running applications – so a static update of a service based access control system would probably not force a restart of the running applications. Let's examine how the four systems work.

Ponder is probably the most static of the systems, because the result of the policy language being compiled is a set of scripts, etc. that implement the back-end. Each backend can be specific to architecture or to a site, depending on how the back-end is implemented would have some effect – but the general concept is still static.

Sun's implementation of XACML uses a attribute query system that allows the underlying attributes to be modified dynamically. Because this can be implemented as a service, you would have to restart to service to force it to re-establish the configuration file safely. If it is implemented as a local PDP, the application would have to be restarted.

Akenti implements the PDP within the Akenti servers, updating the policies within the server can be done dynamically with respect to the application. dRBAC has a cache invalidation component for its certificates; it also could be updated dynamically.

3.3 Scalability & Replication

Scalability in distributed systems is primarily related to the systems ability to replicate itself. With replication, additional applications can query the PDP without impacting the performance of the overall system.

Once again, how the back-end of Ponder is implemented resolves whether the PDP could be replicated. Since the decision is localized – significant effort in designing how the back-end is implemented would be required – albeit not terribly difficult.

Because XACML is nothing but templates and descriptions within XML, XACML could easily be replicated. Simply pass copies of the document to multiple PDPs. Sun's implementation as a network service is an example of exactly that.

Much the same is true with the Akenti servers. The use of the existing certificate implementations allows revoking and invalidation – these servers can easily be replicated. dRBAC's local evaluation of the certificates would make replication of its model difficult.

4. Related Work

In *RBAC Administration in Distributed Systems* [19], Dekker, et al presented a model for administering RBAC in a distributed system. Their model focused on the fact that RBAC policies for large and distributed systems can become quite large involving hundreds of roles. Addressing safety and availability were key criteria, addressing the situation of many datasets with many subsystems, each with different RBAC policies.

The model was a distributed system model with a centralized administrative system. Mappings provide identification that subsets of policy changes only affect subsets of subsystems – and only those subsystems being affected needed to be updated. The administrative procedure presented focused on handling only the relevant

policy changes while preserving safety and availability. Finally, they extended to original model to be able to handle multiple administrative subsystems.

While focused on RBAC similar to dRBAC, this work is in its early stages making it difficult to draw any comparisons between the two. The model presented by Dekker et al did not address the issues arising from an environment like grid computing where multiple owners control the resources – instead it was focused on a business or entity with multiple datasets and multiple subsystems as in a hospital environment. dRBAC presented a more complete solution within that environment.

In *Distributed Access Control, a Privacy-conscious Approach* [20], Cautis examines the issues associated with privacy where the information that needs to be private may be contained in the policy decision itself. Outsourcing the PDP at that point could cause a violation of the privacy constraints places upon the system.

The key distributed system construct pertained to being able to determine where an incoming query came from. His model proposed rewriting the query to maintain the privacy goals, being able to identify the originator (i.e. had the query been rewritten) was essential to the decision-making process.

While this issue dealt peripherally with managing policies in a distributed computing environment, the focus of the paper was on maintaining the privacy concerns of the originating query and not on system details such as performance and robustness.

In *Towards Effective Security Policy Management for Heterogeneous Network Environments* [21], Teo and Ahn present a system-driven policy framework called Chameleos-x. Chameleos-x was designed to address the specific issue of the management of consistent security policies in heterogeneous environments. This is one of the key problems that exist in cluster or grid computing.

Chameleos-x differs from Ponder in that Ponder is strictly a policy specification language and Chameleos-x is involved in both the policy specification and the enforcement. This is done through the use of pluggable policy sets supporting multiple system entities. These entities represent the systems within the heterogeneous environment. To assist within the diverse environment, Chameleos-x assists in the configuration stage, performs evaluation of the policies (similar to running it through a syntax checker), and implements a response mode to enforce the policy.

Chameleos-x was still in experimental phase at the time of the paper (2007), with little actual data to evaluate. This approach will be worth further examination at when the tool chain has been created and data is available for use.

5. Conclusion

We've described how access control within a distributed system raises its own unique problems. For each of four existing access control implementations, Ponder, XACML, Akenti, and dRBAC, we looked at the underlying technologies, basic design concepts, and the architectures of their implementations. We then looked at feature sets required to be successful within a distributed computing environment, including reconciliation of conflicting policies, ability to dynamically update policies, and replication. Finally, we showed three other related technologies.

The most successful implementations had two major design decisions that led to their success. The first was abstracting the policy definitions from the management package; the second was the use of an existing widespread technology, such as X509 certificates or XML as their underlying policy representation. Because support for these widespread technologies already exists in most systems, adapting them to use in a distributed environment was greatly simplified and provided more easily adaptable solutions within a heterogeneous environment.

References

[1] D.E. Bell and L.J. La Padula. *Secure Computer System: Unified Exposition & Multics Interpretation*. Technical report, Technical Report MTIS AD-A023588, MITRE Corporation, 1975.

[2] J. P. Anderson. *Computer Security Technology Planning Study*, ESD-TR-73-51, vol. I, ESD/AFSC, Hanscom AFB, Bedford, Mass., October 1972 (NTIS AD-758 206)

[3] David Ferralolo and Richard Kuhn. Role-based access Controls. In *Proceedings of the 15th NIST-NCSC National Computer Security Conference*, pages 554-563. Baltimore, MD. 1992

[4] J. Park and R. Sandhu, *The UCON_ABC Usage Model*, ACM Transactions on Information and System Security, 7(1): 128-174, 2004.

[5] Moses, T., ed., *eXtensible Access Control Markup Language(XACML), Version 2.0*; OASIS Standard, February 1, 2005; http://www.oasisopen.org/committees/tc_home.php?wg_abbrev=xacml.

[6] http://www-dse.doc.ic.ac.uk/Research/policies/ponder.shtml. *Ponder: A Policy Language for Distributed Systems Management.*

[7] Carlos Ribeiro, Andre Zuquete, Paulo Ferreira and Paulo Guedes. *SPL: An access control language for security policies with complex constraints.* In Proceedings of the Network and Distributed System Security Symposium. 2001.

[8] I. Foster, C. Kesselman, and S. Tuecke. *The Anatomy of the Grid: Enabling Scalable Virtual Organizations*. International Journal of High Performance Supercomputing Applications, 15(3), 2001.

[9] http://www-dse.doc.ic.ac.uk/cgi-bin/moin.cgi/mss. *Prof. Morris Sloman*

[10] http://www-dse.doc.ic.ac.uk/Research/policies/ponder/PonderSpec.pdf. *Ponder: A Language for Specifying Security and Management Policies for Distributed Systems. October, 2000.*

[11] http://www.oasis-open.org/who/

[12] M. Lorch, S. Proctor, R. Lepro, D. Kafura, S. Shah. *First Experiences using XACML for Access Control in Distributed Systems.* ACM Workshop on XML Security. October, 2003.

[13] http://acs.lbl.gov/Akenti/docs/specs.html.

[14] M.Thompson, A. Essiari, S. Mudumbai. *ACM Transactions on Information and System Security*, (TISSEC), Volume 6, Issue 4 (November 2003) pp: 566-588

[15] http://acs.lbl.gov/security/DistSecArch.html.

[16] E. Freudenthal, T. Pesin, L. Port, E. Keenan, and V. Karamcheti. *dRBAC: Distributed role-based access control for dynamic coalition environments*. In Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS 2002), Jul 2002.

[17] http://www.w3.org/TR/xpath. *XPath path language*.

[18] http://sunxacml.sourceforge.net/. *Sun's XACML Implementation*.

[19] M. Dekker, J. Crampton, S. Etalle. *RBAC Administration in Distributed Systems.* In Proceedings of the 13th Symposium on Access Control Models and Technologies. (SACMAT). 2008.

[20] B. Cautis. *Distributed Access Control: A Privacy-conscious Approach.* In Proceedings of the 12th Symposium on Access Control Models and Technologies. (SACMAT). 2007.

[21] L. Teo and G. Ahn. *Towards Effective Security Policy Management for Heterogenous Network Environments.* 8th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'07). 2007.