

Attribute Based Access Control for Grid Computing

Bo Lang,^{1,2} Ian Foster,^{1,3} Frank Siebenlist,^{1,3} Rachana Ananthakrishnan,¹ Tim Freeman^{1,3}
¹ Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL

² Beihang University, Beijing, China

³ University of Chicago, Chicago, IL

Email Addresses: langbo@buaa.edu.cn (Bo Lang, the corresponding author), foster@mcs.anl.gov (Ian Foster), franks@mcs.anl.gov (Frank Siebenlist), ranantha@mcs.anl.gov (Rachana Ananthakrishnan), tfreeman@mcs.anl.gov (Tim Freeman)

Abstract:

Grid systems, which are composed of autonomous domains, are open and dynamic. In such systems, there are usually a large number of users, the users are changeable, and different domains have their own policies. The traditional access control models that are identity based are closed and inflexible. The Attribute Based Access Control (ABAC) model, which makes decisions relying on attributes of requestors, resources, and environment, is scalable and flexible and thus is more suitable for distributed, open systems. But no ABAC model meets the special authorization requirements of Grid computing. This paper presents an Attribute Based Multipolicy Access Control (ABMAC) model based on the concept of ABAC and the authorization requirements of Grid systems. Also described is an authorization framework that was implemented in the Globus Toolkit release 4 and supports ABMAC. This attribute-based authorization framework supports several different policies and integrates third-party attribute-based authorization systems. It shows great advantages in supporting Grid application access control, which not only demonstrates the effectiveness of ABMAC model but also provides an open architecture for Grid authorization systems.

Keywords: Attribute-Based Access Control, Grid Computing, Globus Toolkit, GT4 Authorization Framework

1. Introduction

Access control as an important protection mechanism in computer security is evolving with the development of applications. Since the early 1970s, several access control models have appeared, including Discretionary Access Control (DAC), Mandatory Access Control (MAC), and Role Based Access Control (RBAC) [1][2][3]. These models are considered identity-based access control models, where subjects and objects are identified by unique names and access control is based on the identity of the subject, either directly or through roles assigned to the subject. DAC, MAC and RBAC are effective for closed and relatively unchangeable distributed systems that deal only with a set of known users who access a set of known services.

Recently, large-scale distributed open systems such as the Grid have been developing rapidly. A Grid system is a virtual organization comprising several independent autonomous domains [4]. In a Grid, the relationship between resources and users is more ad hoc and dynamic, resource providers and users are not in the same security domain, and users are usually identified by their characteristics or attributes rather than predefined identities. Therefore, the traditional identity-based access control models are not effective, and access decisions need to be made based on attributes. Also, in a Grid system,

autonomous domains have their own security policies, so the access control mechanism needs to be flexible to support different kind of policies.

Since the late 1990s, with the development of Internet-based distributed systems, a new access control model – the Attribute Based Access Control (ABAC) – has become increasingly important. In ABAC, access decisions are based on attributes of the requestor and resource, and users need not be known by the resource before sending a request. Current research and development efforts of ABAC usually focus on one kind of policy definition, however, and cannot support multiple policies. Hence, in order to establish an authorization mechanism suitable for Grid computing, further research is needed.

The paper is organized as follows. Section 2 surveys the research of attribute-based access control models. Section 3 gives a formal definition of the ABAC model, describes the special access control requirements of Grid computing, and presents our Attribute Based Multipolicy Access Control Model (ABMAC). Section 4 describes the design and implementation of the authorization framework, which is in accordance with the ABMAC model. Section 5 analyses the advantages of the framework for Grid Computing and summarizes our work.

2. Related Work

Since the early 1990s, with the development of Internet and Internet-based distributed application, public-key infrastructure (PKI) and X.509 certificates [5][6] have been widely used for authentication [7]. In 1996, the Simple Public Key Infrastructure SPKI was developed, which is a kind of PKI that emphasizes on authorization rather than authentication [8]. SPKI is one of the earliest attempts to define an authorization certificate [9]. In 1997, the Attribute Certificates (AC) was included in X.509 [10]. An AC may contain attributes that specify group membership, role, security clearance, or other authorization information associated with the AC holder [11]. Recent research and development efforts in attribute-based access control are based on the X.509 ACs.

A first attempt to provide a uniform framework for attribute based access control specification and enforcement was proposed by Damiani et al. [12]. They presented a uniform framework to logically formulate and reason about both service access and disclosure constraints based on related entity attributes [13]. Wang, Wijesekera, and Jajodia proposed a framework that models an attribute-based access control system using logic programming with set constraints of a computable set theory [14]. Most recently, Yuan and Tong described the attribute-based access control model in terms of its authorization architecture and policy formulation [15]. Although these models are all valuable to ABAC research, they are more general and not concerned about the special access control requirements of the Grid.

Attribute-based access control systems are an active area of research in Grid computing, and several systems have appeared, such as Akenti, PERMIS, Shibboleth, and VOMS. Akenti, developed by Lawrence Berkeley National Laboratory [9][16], represents the authorization policy for a resource as a set of certificates digitally signed by unrelated stakeholders from different domains. These certificates express the attributes a user must have in order to get specific rights to a resource. PERMIS, developed under the EC PERMIS project, is a role-based access control infrastructure based on X.509 PMI and using X.509 attribute certificates [17][18]. The project staff developed a policy-

driven decision engine and defined a policy language using XML. Shibboleth is an attribute authority service developed by the Internet2 community for cross-organization identity federation; it asserts attributes about a user and can make access decisions based on these attributes [19][20]. VOMS, developed by the European Data Grid and DataTAG projects, runs in a virtual organization, manages authorization information about its own members, and supplies this information as a kind of attribute certificate [21].

Akenti, PERMIS, and Shibboleth are kinds of ABAC systems and have been used in several Grid systems. However, these authorization systems support their own policies and cannot support multiple different policies. A more flexible and scalable attribute-based access control method is still needed to achieve more effective access control for the heterogeneous Grid computing environment. Also needed are a reasonable policy model that acts as a theory basis and an open architecture that supports the implementation of the model.

3. An Attribute-Based Multipolicy Access Control Model for Grid Computing

In a Grid system, each autonomous domain has its own security policy, such as the grid-mapfile, ACL (Access Control List), CAS, SAML authorization decision assertions, and XACML policy statements. Hence the authorization mechanism of the Grid system needs to be flexible to support these multiple policies. To this end, we built the Attributed-Based Multi-policy Access Control model.

In this section, the formal definition of ABMAC model is first given, followed by a scenario that uses the model to describe a policy.

3.1 Formal Definition of ABMAC

In ABMAC, access control decisions are made based on the attributes of the requestor, the resource, the action, and the environment. The formal definition of ABMAC is composed of four parts: access control related entities, attributes of entities, policy representation, and policy evaluation.

(1) Access control-related entities

Requestor

A requestor is the entity that sends requests to the Grid service and invokes actions on the service; it is represented as *Req*.

Service

A service is a Grid service that is a software agent with a network-addressable interface containing some well-defined operations; it can be invoked via standard protocols and data formats [22]. A service in ABMAC is represented as *Sev*.

Resource

A resource is a system entity that is acted upon by one or more Grid services. In Grid computing context, resource is always stateful; that is, has a specific set of state data expressible as an XML document and a well-defined lifecycle. Examples of stateful resources are files in a file system, rows in a relational database, and encapsulated objects such as Entity Enterprise Java beans [22]. A resource in ABMAC is represented as *Res*.

Action

An action is an operation provided by a Grid service that can be invoked by clients; it is represented as *Act*.

Environment

Environment is the context related to an invocation of a Grid service. It contains information that is not associated with any specific entity but might be useful in the decision process, such as the current date and time. The environment is represented as *Env*.

(2) Attributes of entities

Each entity has attributes that define the identity and characteristics of the corresponding entity. We define the attributes of the entities in ABMAC as follows:

The attributes of the requestor may contain the identifier, the name, the organization, and the other information of the requestor and are defined as follows:

$$Attr(Req) = \{ReqAttr_i \mid i \in [1, I]\}$$

The attributes of the service may include the service name and address and are defined as follows:

$$Attr(Sev) = \{SevAttr_j \mid j \in [1, J]\}$$

The attributes of the resource may include resource name, identifier, and other information and are defined as follows:

$$Attr(Res) = \{ResAttr_k \mid k \in [1, K]\}$$

The attributes of an action can be an action name, for example, and is defined as follows:

$$Attr(Act) = \{ActAttr_l \mid l \in [1, L]\}$$

The attributes of the environment *Env* may be the current date or time and are defined as follows:

$$Attr(Env) = \{EnvAttr_m \mid m \in [1, M]\}$$

The I, J, K, L, and M in these definitions are the maximum number of attributes of the corresponding entities and are integers.

(3) Policy representation

The authorization systems of Grid computing need to support multiple security policies, each of which may have its own policy description method. To ensure the integration of different policies and to make ABMAC more scalable, we encapsulated each policy as an independent policy unit and defined the policy that ABMAC supports as a superset of these policies:

$$ABMAC_Policy = \{P_i \mid i \in [1, m], P_i \text{ is a policy.}\}$$

(4) Policy evaluation

Policy evaluation is the process of making an access decision based on the security policy. The decision is made by the Access Control Decision Function (ADF) [23], which applies access control policy rules to an access request. In ABMAC, we defined a function named *adf()* to implement ADF; it takes the attributes of the requestor, the service, the resource, the action, and the environment as parameters.

The evaluation function of policy P_i , called $P_i_adf()$, is defined as follows:

$$P_i_adf(Attr(Req), Attr(Sev), Attr(Res), Attr(Act), Attr(Env))$$

$$= \begin{cases} \text{permit} \\ \text{deny} \end{cases}$$

The request evaluation function of ABMAC, called $ABMAC_adf()$, is defined as follows:

$$\begin{aligned} & ABMAC_adf(Attr(Req), Attr(Sev), Attr(Res), Attr(Act), Attr(Env)) \\ &= combine_f(p_1_adf(Attr(Req), Attr(Sev), Attr(Res), Attr(Act), Attr(Env)), \\ & \quad p_2_adf(Attr(Req), Attr(Sev), Attr(Res), Attr(Act), Attr(Env)), \\ & \quad \dots \dots, \\ & \quad p_m_adf(Attr(Req), Attr(Sev), Attr(Res), Attr(Act), Attr(Env))) \\ &= \begin{cases} \text{permit} \\ \text{deny} \end{cases} \end{aligned}$$

The access control decision function of ABMAC, called $ABMAC_adf()$, implements a combining algorithm in $combine_f()$, which combines the decision results returned by the access control decision function of each policy P_i and makes a final access decision.

3.2 Using ABMAC to Describe a Policy

In a Grid system, an authorization system is always established to guard one Grid service. In the following scenario, the attributes of the requestor, the resource, the action, and the environment will be used to make an access control decision. We use a data structure named Attribute, which contains the attribute name and attribute value to describe an attribute.

(1) Entities and attributes definition

The requestor

X.509 End Entity Certificates are widely used in Grid computing. If the X.509 End Entity Certificate is used for the authentication of a requestor, then the requestor's Distinguished Name and the public key can be the following two attributes:

```
ReqAttr1 = Attribute( name="x509SubjectDN", value="CN=requestor1")
ReqAttr2 = Attribute( name="publicKey", value="#$%$^&$&requestor1&*&#")
theRequestor = Set(ReqAttr1, ReqAttr2)
```

The resource

According to the WSRF specification [24], a WS-Resource is composed of a Web service and several stateful resources. A resource is associated with one or more WSDL portTypes, by which the resource can be operated by the Web services. We define the resource as follows:

```
ResAttr1 = Attribute( name="portType", value="fileTransferPortType")
ResAttr2 = Attribute( name="resourceId", value="#%#@resrcId$$%#")
theResource = Set(ResAttr1, ResAttr2)
```

The action

The action is the operation defined in a service porttype:

```
ActAttr1 = Attribute( name="operation", value="readFile")
theAction = Set(ActAttr1)
```

The environment

The environment contains the current time as the attribute:

```
EnvAttr1 = Attribute( name = "currentTime", value="21:57:05.09")
```

```
theEnvironment = set(EnvAttr1 )
```

(2) Policy evaluation

After all the entities and their attributes are defined, the policy evaluation function can be called to make an access control decision. We do so by passing all the attributes of the entities to the *ABMAC_adf()* function, which in turn calls the decision function of the supported policies and combine the returned decision results by calling the *combine_f()* function:

```
ABMAC_decision  
= ABMAC_adf( theRequester, theResource, theAction, theEnvironment )  
= combine_f( p1_adf( theRequester, theResource, theAction, theEnvironment ),  
            P2_adf( theRequester, theResource, theAction, theEnvironment ),  
            ... ..  
            p1_adf( theRequester, theResource, theAction, theEnvironment ) ).
```

3.3 Characteristics of ABMAC

Policy representation and evaluation are the most important parts in attribute-based access control models. ABMAC defines a hierarchical policy structure basing on the abstraction and encapsulation concepts. The policy of ABMAC is a policy set composed of different kinds of policies that need to be supported. The policies are encapsulated; that is, they use their own definitions and decision-making algorithms. Compared with other ABAC models, ABMAC does not use a unified method to describe each policy. A unified description method would force the policies to change their descriptions, a situation that is difficult to achieve and is impractical in a heterogeneous real system. ABMAC can support each policy without any change.

Hence ABMAC is a policy framework. The encapsulation of the heterogeneous policies enables ABMAC to support multiple policies effectively and makes the model more flexible and scalable.

4. GT4 Attribute-Based Authorization Framework

In this section we present the authorization framework in the Globus Toolkit.

4.1 Design Concepts of the GT4 Authorization Mechanism

As a fundamental enabling platform for Grid, the authorization mechanism of Globus should have the following properties:

- Supporting attribute based access control.

- Being flexible and scalable for supporting multiple policies.

- Being open for integrating existing authorization systems.

For building such an authorization system, we used the same ideas of ABMAC model, namely, policy abstraction and encapsulation, and used the same policy description. Based on this policy model, we designed the authorization mechanism architecture using Web services specifications.

The Globus Toolkit release 4 (GT4) implements the WSRF specification, which is a convergence of Grid and Web services technology. Several Web services standards are introduced into Grid computing. XACML (eXtensible Access control Markup Language) and SAML (Security Assertions Markup Language) are access control-related Web services standards that support attribute-based access control [25][26]. XACML provides a basic authorization architecture; and SAML, which defines a framework for exchanging security information such as authentication and authorization decisions in XML format, is the technology for integrating existing authorization systems. Thus, we built the GT4 authorization framework based on the multipolicy model, XACML, and SAML.

4.2 XACML Authorization Architecture

XACML defines a policy language using the attributes of requestors, resources, and environment. The authorization architecture supports attribute-based access control [25], as shown in Fig. 1.

The access control framework mainly contains PEP (Policy Enforcement Point), PDP (Policy Decision Point), PIP (Policy Information Point), and PAP (Policy Administration Point). The PEP intercepts the access requests from users and sends the requests to the PDP. The PDP makes access decisions according to the security policy or policy set written by PAP and using attributes of the subjects, the resource, and the environment obtained by querying the PIP. The access decision given by the PDP is sent to the PEP. The PEP fulfills the obligations and either permits or denies the access request according to the decision of PDP.

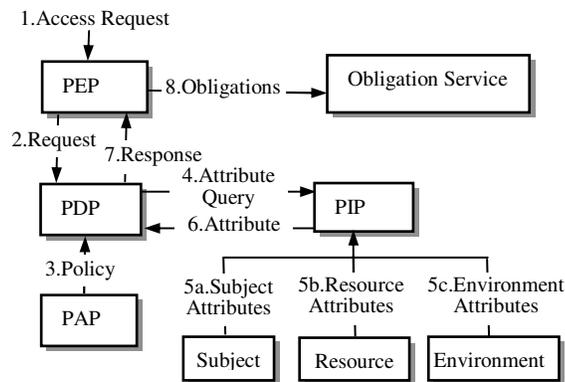


Fig. 1: XACML Data Authorization Framework

4.3 Multipolicy Framework in GT4

The PDP is the entity that implements the policy evaluation function of a policy model. It is the core of the authorization framework that makes access control decisions. In XACML, PDP is supposed to implement only one kind of policy, and the multipolicy supporting methods are not provided. Based on the concepts described in ABMAC, we define a scalable multipolicy framework, shown in Fig. 2.

Each policy, such as Grid map file and access control list, is the policy P_i in ABMAC. These policies essentially need their own decision functions that understand the intrinsic semantics of the policy expressions. Hence we encapsulate each policy in an independent PDP. At the same time, we define an abstract PDP that has the common characteristic of

the policies. In accordance with the policy evaluation function `ABMAC_adf()` in ABMAC, the PDP abstraction (the PDP class in Fig. 2) defines a common evaluation interface `canAccess()` that can be used to interact with the PEP or with other PDPs. This common interface presents the decision request as a collection of attribute values for the subject, resource, and action. Each specific policy is a subclass of the PDP abstraction, which implements the common interface `canAccess()` inherited from PDP with its own policy and evaluation mechanism.

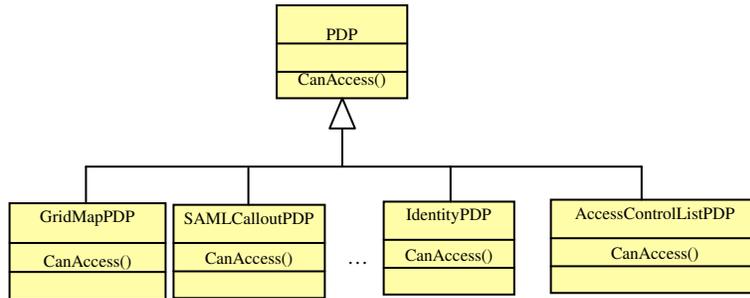


Fig. 2: GT4 Authorization Policy Framework

In the implementation of the framework, a separate Master PDP is created that implements the `ABMAC_Policy` in ABMAC. The Master PDP first collects information about the request and calls the PDPs, then combines the decisions from all the different PDP instances, and renders a single decision reflecting the overall evaluated policies.

Since the policy framework is object oriented, it is scalable and flexible, which means that new policies can be added to the framework just by inheriting the PDP class and that the existing policies can be removed and modified at any time. Also, since PDP instances are queried through the same interface and since the mechanism-specific details of the PDPs are all hidden behind the public interface, a change to the policy framework has no effect on the Master PDP: it can cooperate with any specific PDPs designated by the security configuration files. This multipolicy framework thus provides users of GT4 with an authorization mechanism that is flexible and scalable and can support multiple different policies.

4.4 Architecture of GT4 Authorization Framework

Based on the GT4 authorization policy framework and on the XACML and SAML standards, we built the GT4 authorization framework, which is shown in Fig. 3. The framework is composed of a PEP, PDPs, and PIPs.

Grid systems have several frequently used simple authorization policies or mechanisms. We provided PDPs, such as `AccessControlListPDP` and `GridMapAuthorizaionPDP`, which implement these policies. Four types of decision may be returned by each PDP: permit, deny, not applicable, and indeterminate. In order to integrate the authorization systems developed by others into the authorization framework, such as Shibboleth, VOMS and PERMIS, a `SAMLAuthorizationCalloutPDP` is established that integrates these systems through the SAML assertions.

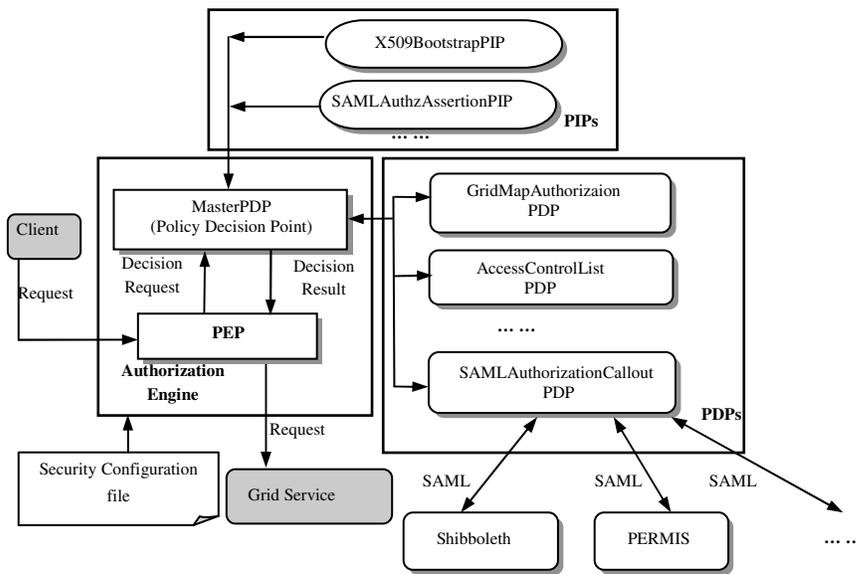


Fig. 3: GT4 Authorization Framework

The Master PDP uses a combining algorithm, such as Deny override, Permit override, and First applicable, to combine the decisions returned by each PDP. The algorithm can be configured in various ways.

The PEP intercepts the user’s request and executes the authorization decision received from the Master PDP. The Master PDP and the PEP are collectively called the authorization engine of the framework.

The PIPs are information collection points that collect attributes about various entities related to the authorization evaluation. The attributes-collecting process is shown in Fig. 4.

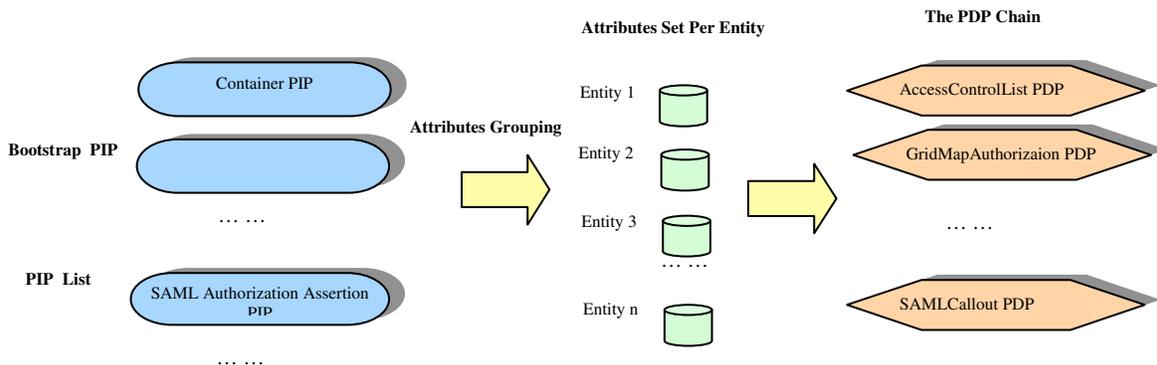


Fig. 4: Attributes Collection in GT4

When collecting the authorization information, the Container PIP is first invoked to collect attributes inherent to the framework, such as the service name and the operation

name. Next, the Bootstrap PIPs are invoked to collect information about the request; usually the X509 Bootstrap PIP is invoked before any other Bootstrap PIP configured. Then, other PIPs are invoked in the configured order. Each PIP might return a normalized representation of the collected attributes. The attributes then are grouped as a single set of attributes per entity and are stored, so that the PDPs in the PDP chain can access them when evaluating their policies.

In the authorization framework, the collection of Bootstrap PIPs, PIPs, and PDPs is referred to as an authorization chain. An authorization chain can be configured through the security configuration file (shown in Fig. 3) or programmatically at the resource, service, and container level.

4.5 GT4 Authorization Framework Decision-Making Algorithm

When a request of the Grid resource comes, it is intercepted by the PEP in the authorization engine and sent to the Master PDP to begin evaluation. The evaluation algorithm implemented by the authorization framework is expressed by the following pseudo codes:

```

CALL master_PDP_decision() with the request RETURNING decision
IF decision = 'permit' THEN
    Forward the request to the Grid resource
ELSE
    Deny the request
END IF

Master_PDP_decision( request )
    Input PIP subchain, PDP subchain
    CALL BootstrapPIP_attribute_collection() RETURNING attributes of the requestor
    Add the attributes to the request-attribute-storage
    FOR each PIPi in PIP subchain
        CALL PIPi_attribute_collection() RETURNING attributesi
        Add the attributesi to request-attribute-storage
    ENDFOR
    Group the attributes in request-attribute-storage into entities
    Associate the entities with resource, action, environment.
    FOR each PDPj in PDP subchain
        CALL PDPj_decision() with the requestor, the resource, the action, the environment
        RETURNING decisionj
        CALL master_PDP_combinator_algorithm() with the decisionj
        RETURNING decision
        IF decision = 'permit' or decision = 'deny' THEN
            Break
        ENDIF
    ENDFOR
    RETURN with decision

```

When the Master PDP received the evaluation request from PEP, it first collects information needed by calling the Bootstrap PIPs and other PIPs and then invokes the corresponding PDPs with the request and the information collected. The PIPs and the PDPs used are all specified in the security configuration file. When the Master PDP receives the decisions returned by each PDP, it combines the decisions, using a policy

combination algorithm to render a final decision, and returns the decision to the PEP. The PEP then executes the decision, either denying or permitting the request.

4.6 Integration of Third-Party Authorization Systems

As shown in Fig. 3, the GT 4 authorization framework uses a SAMLAuthorizationCallout PDP to integrate a third-party authorization system, such as Shibboleth and PERMIS. This PDP uses SAML to interact with these authorization systems.

The SAML specification defines a number of elements for making assertions and queries regarding authentication, authorization decisions, and attributes [26]. As for authorization, SAML defines messages exchanged between a PEP and a PDP: the AuthorizationDecisionQuery element is used to send request to the PDP, and an Assertion returned from the PDP contains some number of AuthorizationDecisionStatements.

Both the SAMLAuthorizationCallout PDP in the GT4 authorization framework and the third-party authorization systems need to support SAML standard. The SAMLAuthorizationCallout PDP sends a SAML AuthorizationDecisionQuery to an outside authorization service. The service evaluates the request against its policy and returns a response encoded as a SAML Assertion, which includes one or more AuthorizationDecisionStatements.

The AuthorizationDecisionQuery contains a Subject, Resource, and any number of Action elements, while the Assertion returned by an outside authorization service includes the following elements [27]:

- A *Conditions* element specifying the conditions for use of the assertion
- An *Advice* element specifying advice for use of the element
- Any number of *AuthorizationDecisionsStatements* specifying capabilities, which contains the same elements as the AuthorizationDecisionQuery
- An optional *Signature* element allowing the Assertion to be verified

The SAMLAuthorizationCallout PDP will analyze the assertion returned. If the assertion contains a positive decision, the request will be permitted; otherwise it will be denied.

5. Conclusion

Attribute-based access control, making access decisions based on the attributes of requestors, resources, and the environment, provides the flexibility and scalability that are essential to large-scale distributed systems such as the Grid. To support the special authorization requirements of Grid computing, we developed an attribute-based multipolicy access control model ABMAC and described the GT4 authorization framework that supports the model. The authorization framework provides the needed features for Grid computing, which makes decisions based on attributes of related entities, supports multiple policies, and can integrate third-party attribute-based authorization systems. Our results show that the ABMAC model and the architecture for implementing multipolicy attribute-based access control provided in the GT4 authorization framework are effective.

Acknowledgments

Work on GT4 GSI has been funded in part by NSF, by IBM, and by the U.S. Department of Energy under Contract W-31-109-Eng-38.

References

- [1] Butler W. Lampson. Protection. In Proc. 5th Princeton Conference on Information Sciences and Systems, Princeton, 1971, pp. 437-443.
- [2] D. E. Bell, L. LaPadula, Secure Computer Systems: A Mathematical Model. Mitre Corporation, Bedford, Mass., January 1973.
- [3] R. S. Sandhu and P. Samaratiy. Access Control: Principles and Practice. IEEE Communications, 32(9):40-48, 1994.
- [4] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International J. Supercomputer Applications, 15(3):200-222, 2001.
- [5] ITU-T Recommendation X.509. Information technology - Open systems Interconnection - The Directory: Authentication Framework, ISO/IEC 9594-8, 1993.
- [6] R. Housley, W. Ford, W. Polk, and D. Solo. Internet X.509 Public Key Infrastructure Certificate and CRL Profile, September 1998.
- [7] J. S. Park and R. Sandhu. RBAC on the Web by Smart Certificates. In Proc. 4th ACM Workshop on Role-Based Access Control. ACM, Fairfax, VA, October 28-29 1999.
- [8] R. L. Rivest and B. Lampson. SDSI - A Simple Distributed Security Infrastructure. Presented at CRYPTO '96 Rumpsession, April 1996.
- [9] M. Thompson, W. Johnston, S. Mudumbai, G. Hoo, K. Jackson, and A. Essiani. Certificate-based Access Control for Widely Distributed Resources. In Proc. Usenix Security Symposium, Aug. 1999.
- [10] Joon S. Park and R. Sandhu. Smart Certificates: Extending X.509 for Secure Attribute Service on the Web, NISSC 1999.
- [11] S. Farrell and R. Housley. An Internet Attribute Certificate Profile for Authorization, IETF — RFC 3281, 2002.
- [12] E. Damiani, S. De Capitani di Vimercati, and P. Samarati. New Paradigms for Access Control in Open Environments. In Proc. 5th IEEE International Symposium on Signal Processing and Information, Athens, Greece, December 18-21, 2005.
- [13] P. Bonatti and P. Samarati. A Unified Framework for Regulating Access and Information Release on the Web. J. Computer Security, 10(3):241-272, 2002.
- [14] L. Wang, D. Wijesekera, and S. Jajodia. A Logic-based Framework for Attribute based Access Control. In Proc. 2004 ACM Workshop on Formal Methods in Security Engineering, Washington, D.C., October 2004.
- [15] E. Yuan and J. Tong. Attributed Based Access Control (ABAC) for Web Services. In Proc. IEEE International Conference on Web Services (ICW'05), 2005.7.
- [16] M. Thompson, A. Essiari, and S. Mudumbai. Certificate-based Authorization Policy in a PKI Environment. ACM Transactions on Information and System Security (TISSEC), 6(4):566-588, November 2003.
- [17] D. Chadwick. Authorization in Grid Computing. Information Security Technical Report, 10(1):33-40, 2005.

- [18] D. Chadwick and A. Otenko. The PERMIS X.509 Role based Privilege Management Infrastructure. *Future Generation Computer Systems*, 19(2):277-289, February 2003.
- [19] V. Welch, T. Barton, K. Keahey, and F. Siebenlist. Attributes, Anonymity, and Access: Shibboleth and Globus Integration to Facilitate Grid Collaboration. In *4th Annual PKI R&D Workshop*, April 2005.
- [20] T. Barton, J. Basney, T. Freeman, T. Scavo, F. Siebenlist, V. Welch, R. Ananthakrishnan, B. Baker, M. Goode, and K. Keahey. Identity Federation and Attribute-based Authorization through the Globus Toolkit, Shibboleth, Gridshib, and MyProxy. In *5th Annual PKI R&D Workshop*, April 2006.
- [21] R. Alfteri, R. Cecchini, V. Ciaschini, L. Dellagnello, A. Frohner, A. Gianoli, K. Lorente, and F. Spataro VOMS, An Authorization System for Virtual Organizations, In *1st European Across Grids Conference*, Santiago de Compostela, February 13-14, 2003.
- [22] I. Foster, J. Frey, S. Graham, S. Tuecke, K. Czajkowski, D. Ferguson, F. Leymann, M. Nally, T. Storey, and S. Weerawaranna. Modeling Stateful Resources with Web Services. *Globus Alliance*, 2004.
- [23] ISO/IEC 10181-3:1996, Information Technology - Open Systems Interconnection - Security Frameworks for Open Systems: Access Control Framework.
- [24] K. Czajkowski, D. F Ferguson, I. Foster, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke, and W. Vambenepe. The WS-Resource Framework, Version 1.0, March 5, 2004
- [25] OASIS, Extensible Access Control Markup Language (XACML), V2.0, February 2005.
- [26] OASIS, Security Assertion Markup Language (SAML), V2.0, March 2005.
- [27] V. Welch, R. Ananthakrishnan, S. Meder, L. Pearlman, and F. Siebenlist. Use of SAML for OGSA Authorization (work in progress), *Global Grid Forum*, May 14, 2004.

The submitted manuscript has been created by the University of Chicago as Operator of Argonne National Laboratory (“Argonne”) under Contract No. W-31-109-ENG-38 with the U.S. Department of Energy. The U.S. Government retains for itself, and others acting on its behalf, a paid-up, nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government. This government license should not be published with the paper.