Using Attribute Certificates with Mobile Policies in Electronic Commerce Applications[1]

Vinti Doshi, Amgad Fayad, Sushil Jajodia, Roswitha MacLean

The MITRE Corporation
1820 Dolley Madison Boulevard
McLean, Virginia 22102
{vdoshi, afayad, jajodia, rmaclean}@mitre.org

Abstract

Many electronic commerce applications including those developed for business-to-consumer (B2C) and business-to-business (B2B) uses, require operations in computing environments that are truly distributed. That is, users can request data access from multiple locations within a distributed computing system. To complicate this type of operation, however, data can be distributed and represented in multiple forms. As a result, system administrators are encountering increasing difficulty in developing and managing application-specific policies for users and data. A multi-tier (N-tier) architecture can provide a powerful solution for meeting the diverse needs of the electronic commerce applications. However, a drawback to multi-tier architectures is that they require that a user's credentials and the policy-to-data mapping context must be available in the middle tier of the system architecture.

This paper addresses the management of users and data by presenting a framework for combining attribute certificates with mobile policy for effective application-specific control specification and administration in a distributed computing environment. Attribute certificates provide mobility to credentials and also provide fine-grained information about security principals. Mobile policy allows application-specific policies to move along with the data to other elements of the distributed computing system. Herein we propose a high-level definition language to specify policies that are application-specific and mobile, and present an algorithm for enforcing attribute-based mobile policies.

1 Introduction

Many business-to-consumer (B2C) and business-to-business (B2B) applications require computing environments that are truly distributed. Users can request access to data from multiple locations, and data may be distributed and represented in multiple forms. Mechanisms are needed that can achieve consistent access control policy across elements of the distributed environment.

The N-tier architecture provides a powerful paradigm to accommodate the diverse needs of the B2C and B2B applications. In pure multi-tier architectures (see Figure 1), business logic is moved from the client and the database to a middle tier, which consists of a Web server and application code. The application code is hosted on a different platform from the database management system (DBMS). The application code may request data from many DBMSs and may implement functions much more complex than satisfying requests by clients to access data in the DBMS.

Since business logic moves from the client and the database to the middle tier, a down side of multi-tier architectures is that they require that a user's credentials and the policy-to-data mapping context be available in the middle tier. As shown in Figure 1, the user identity typically originates from the presentation tier, while the policy-to-data mapping resides in the data tier. Moreover, the middle-tier must be able to combine

[1] This work was funded by the MITRE technology program under project number 51MSR871.

policies from different sources within the system and resolve any conflicts among policies, whenever necessary.

B2C and B2B applications also require support for application-specific policies that address needs other than traditional access control policies [3,7]. Example application-specific policies include "requiring users to provide their credit card information before their orders can be processed" or "requiring that certain documents be appended with a copyright notice before they are given out to clients."

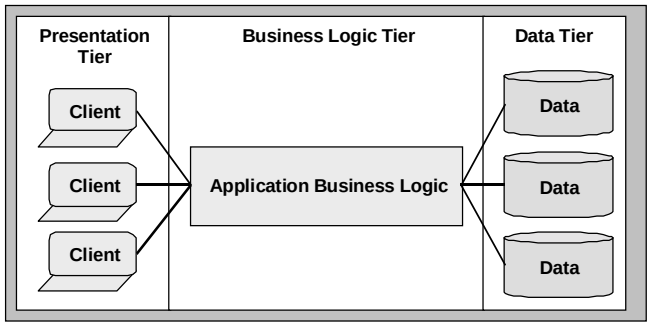

Figure 1. Multi-tier Architecture for Multiple Clients and Databases

This paper presents a framework for combining attribute certificates (ACs) [3] with mobile policy for effective access control specification and administration in the middle tier. Public key certificates, also known as identity certificates (ICs), allow the identities of the users to be mobile. A user can prove his/her identity, for access control purposes, using his/her IC. The advent of ACs extends ICs beyond identity-based authorizations. Attributes define fine-grained authorizations about security principals, allowing for role-based, rank-based, and group-based access control.

We use mobile policies to provide policy-to-data mapping context information to the middle tier. Mobile policies allow flexible policies that move with the data to which the policies apply, and they allow these policies to be enforced within any component, in particular, the middle-tier of the system.

We propose a high-level definition language to specify policies that are application-specific and mobile. It extends the traditional Structured Query Language (SQL) access control commands to incorporate attribute certificate information as well as *provisions* [7] that specify required actions to be taken during policy enforcement.

The remainder of this paper is organized as follows. Section 2 describes some motivating examples of application-specific policies, Section 3 describes attribute certificates, Section 4 describes mobile policy and a language for expressing it, Section 5 describes the integrated architecture, Section 6 presents an algorithm for mobile policy enforcement using attribute certificates, Section 7 describes related work, and finally, Section 8 provides conclusions and addresses future work.

2 Motivating Examples

In this section, we give several motivating examples of application-specific policies that can benefit from the approach we propose in this paper. Throughout, we consider a three-tier system as shown in Figure 1. The first is the presentation tier, which is available to the user. An example of the presentation tier is a Web browser. The second tier contains the application logic. An example of an application logic tier is a Web server. Finally, the third tier is the data store or database.

Example 1: Consider an electronic library system that maintains a collection of articles published by an organization under various projects. The articles are stored in a document server. Users can request copies of articles through a Web browser interface. In order to authenticate to the web server in the middle tier of the system architecture, the user presents an IC. The Web server receives the user's request and connects to the document server for access to the requested article. Before the document can be released to the Web server, and ultimately to the user, the document distribution policy must be examined and enforced. A general distribution policy associated with each article may state that an article A that is published under project P can be made available to all members of P. The policy may also state that any user at the rank of manager or above can access A. However, if the user requesting the document is a sponsor who funded the project or the writing of A, then A can be released only after a proprietary notice is added. No one else should have access to the article.

Example 2: Consider a B2B application that allows two companies, C1 and C2, to exchange data. An intranet, accessible to C1 and C2, provides access to a Web browser interface for requesting data. Some of the data resides in C1's database and some in C2's database. When a user from C2 requests data, through the intranet, that resides in C1's database, a Web server receives the request and determines that it must connect to C1's database. Suppose the following policy applies to the requested data:

1. Every member of the accounting group in C2 can read data. The server should add the notice "Do not distribute outside the accounting group" to the data.
2. The accounting group manager in C2 can update the data.

To enforce such a policy, detailed information about the C2's users requesting the data must be available. Also, as in Example1, enforcement of this policy can take place either before the data leaves C1's enclave or after the data leaves C1's enclave at the Web server.

Example 3: We modify Example 1 and consider multimedia documents instead of text. We also assume that the Web server is capable of requesting network bandwidth for playing multimedia files requested by the user. Consider the following policy , which allocates different bandwidth for the multimedia file based on the user's credentials: If the user is a member of the accounting group, he can use 5 megabits per second (Mb/S) of bandwidth allocated. If the user is a member of the engineering group, he can use 3 Mb/S of bandwidth.

The following list provides variations on the policies we discussed in the previous examples:

- Every member of accounting group except group RD2 can read the document. The server should add the following copyright notice on printed copies: "For personnel use only."
- The document can be accessed by anyone if he/she provides his/her name and email address. The printed document should display a copyright notice.
- The document can be given to anyone who has a valid credit card number as part of his/her credentials.
- Access can be denied to anyone who requests the object; ask the requestor to contact the object owner directly and send the owner an email about it (with this policy, the owner retains the access control [10]).

3 Attribute Certificates

We begin with a definition of an attribute for a security principal. A security principal ***S*** could be a user, a server, or an application.

Definition 1 (Attribute): An attribute for a security principal ***S*** is an ordered pair (Tag, Value) where Tag is an identifier and Value is a text string.
Examples of attributes are the following:

- (group, Accounting_Group)
- (role, payroll_clerk)
- (rank, senior_staff)
- (credential, valid_credit_card_number)

As said earlier, in a Web application, where the number of potential users or integrated systems is unknown, the model of separating user identity verification and data access determination becomes virtually impossible to administer. ICs are commonly used today to provide identity and authorization information about the requestor. X.509 ICs [5, 6] bind an identity and a public key. The identity may be used to support identity-based access control decisions after the requestor is able to provide a private key that corresponds to the public key contained in the certificate.

Identity-based access control is effective if administrators have a pre-determined list of users and servers. However, in today's distributed applications, the number of potential users and servers is volatile. Managing access control based on user identity alone becomes very difficult. Rule-based, role-based, and rank-based access control decisions require additional information. Authorization information to support such access control decisions may be placed in an IC file extension or in a separate AC [4].

An AC, as described in [4], is a structure similar to an IC. The main difference is that an AC does not contain a public key. An AC must be linked to an X.509 IC and can be used only in conjunction with an X.509 certificate. This restriction is necessary, since an AC cannot provide any information about a user's identity. An AC is, therefore, a signed list of attributes applicable to the user.

ACs also provide the benefit of fine-grained access control based on user authorizations or attributes, such as group membership, rank, and organizational role. Access control specification and administration can be simplified considerably when user and server volatility is present if attribute-level access control is utilized. The two examples in Section 2 illustrate this point.

Examples 1 and 2 Continued: In the library distribution system (Example 1), it is important to know whether the request is coming from a sponsor. Instead of storing such information in a central location, ACs provide a flexible, mobile alternative. Similarly, Example 2 also requires information regarding the requestor's group membership and rank. Additional information like this can be specified in ACs with attributes such as (Organization_Name, …) or (Group_Name, …), for example.

Two benefits can be realized when specifying attributes in a certificate other than the identity certificate [4]. The first benefit is the lifetime of user attributes compared to the lifetime of ICs. As an

example, a user may change roles in an organization every few months, thereby requiring the revocation of the existing AC and issuance of an updated AC. The user's identity certificate, on the other hand, can remain the same.

The second benefit is certificate issuer authority. Public key infrastructures (PKIs) call for a certification authority (CA) as a trusted third party to issue certificates. It may be desirable to have separate CAs issuing ICs and ACs, depending on the organizational needs.

4 Mobile Policy

To provide access control, distributed computing systems currently use one of the following two methods to provide data access: identity delegation and application-dependent access control.

In identity delegation, a user U delegates his/her identity to a server A. Server A, in turn, attempts to access data from the data source, which we will call server D, using U's identity and, if successful, the data is returned to U. This approach, requires that U, A and D share a *security context*. That is, the semantics of the user identity must be consistent among U, A, and D [3].

In application-dependent access control, server A retrieves data from server D using A's identity. Subsequently, application-dependent access control logic determines if user U will be granted access. This approach relies on each application to interpret and implement access control correctly.

Mobile policy provides a third alternative for access control in a distributed computing system [3]. With mobile policy, data and its authorization information travel from server D to server A. Server A enforces the authorization information without having to implement application-dependent policy enforcement logic.

Policy enforcement can take place on any machine that is trusted to perform it. Chapin et al. [3] proposed an application framework for making policy mobile. In addition, it extends policy management in multi-tier applications. In this framework, policy management is decomposed into three functions: defining policies, associating policies with data, and applying policies. Almost any policy can be supported, limited only by the ability of developers to implement the policy in software.

The primary goal of the framework is to support separation of duty among application component developers by moving policy from the database to the application along with the data while minimizing the knowledge that must be shared between data tier developers and business logic tier developers. Secondary goals are to minimize effort on the part of application developers, support assurance that the system works as intended, work harmoniously alongside existing policy mechanisms, support multiple application and DBMS platforms, and minimize the impact on performance.

Sections 4.1 and 4.2 expand the mobile policy description by describing related terminology and a mobile policy definition language.

4.1 Mobile Policy Terminology

We return to the B2B application described in Example 2. According to the policy, data can be read by any member of the accounting group in C2 with the provision that the server must add the notice "Do not distribute outside the accounting group" to the data. Moreover, an accounting group manager in C2 can update the data.

We will call the required elements of a mobile policy, such as "accounting group" and "accounting group manager," the M*obile Policy Attributes* (MPAs). MPAs represent the attributes that the policy requires in the user's AC.

For example, suppose that a user has an AC with the following attribute: (role, accounting chief). Suppose further that the role accounting chief is the new title for the accounting group manager role. A mapping between the original title (which appears as an MPA) and the new title (which appears as an attribute of the AC) becomes necessary. In Section 5, we will show how MPAs can be mapped to attributes from an AC.

We will call actions such as "Add the notice: 'Do not distribute outside the accounting group' to the data" *provisions*.

In Section 4.2, we define a mobile policy definition language. Once the policy is specified, it can be easily implemented as mobile code in Microsoft's ActiveX or Sun Microsystems' Java, for example. Implementations of mobile policy using mobile code are called *policy modules* [3].

4.2 Mobile Policy Definition Language

In this section, we introduce our mobile policy definition language with attributes. It has several features that are found in the authorization specification language proposed by Jajodia, et al. [2, 7-9], extended to allow for attributes. We chose a syntax for our policy definition language that is consistent with the familiar SQL syntax.

The language has three types of statements: **Grant**, **DoNotGrant**, and **MustGrant**. **Grant** provides the ability to give access to data. **DoNotGrant** provides the ability to deny access to data. Finally, **MustGrant** is a

strong authorization; it is used to ensure that authorizations will be obeyed by the system, and to resolve conflicting **Grant** and **DoNotGrant** rules on a particular access.

Definition 2 (Authorization Rule): An authorization rule is a rule of the following form:

Grant <*Access Type*> **on** <*Object*>
to <*Security Principal*>
with provision [<*Provisions*>]
where [<*Security Principal*> **has attribute** <(tag, value), …> | <predicate>]

Here <predicate> is any condition that must be satisfied during the evaluation of the access request.

Authorization rules allow access to security principals. Access to an object is allowed after the conditions in the **where** clause have been validated and the provisions have been applied.

Example 4: Consider the following authorization rule:

Grant update **on** Balance_Sheet
to user1
with provision Add notice "For Accounting Group Only"
where user1 **has attribute** (group, accounting group) **and** (rank, manager of accounting group)

This rule states than *user1* has *update* access to file *Balance_Sheet* if user1 is a member of the accounting group and has the rank of an accounting group manager.

Example 5: Consider the following authorization rule:

Grant read **on** Balance_Sheet
to user1
with provision: add copyright notice
where user1 **has attribute** (group, organization O)
and O has status competitor

This rules states that if *user1* is a member of organization O, and O has the status of a competitor, then *user1* has *read* access to *Balance_Sheet*, after the copyright notice is added.

Definition 3 (Negative Authorization Rule): A negative authorization rule is a rule of the following form:

DoNotGrant <*Access Type*> **on** <*Object*>
to <*Subject*>
with provision [<*Provisions*>]
where [<*Security Principal*> **has attribute** <(tag, value), …> | <predicate>]

Negative authorization rules explicitly deny access to an object by certain security principals. Some applications require explicit negative authorizations. Example 6 shows separation of duty as an explicit negative authorization.

Example 6: Consider the following authorization rule:

DoNotGrant write **on** Payroll_Employee_Check
to user1
with provision Notify Payroll_Supervisor
where user1 **has attribute** (group, payroll)

This rules states that *user1* does not have access to Payroll_Employee_Check if *user1* is a member of the group payroll. If such access is attempted, the Payroll_Supervisor must be notified.

Definition 4 (Strong Authorization Rule): A strong authorization rule is a rule of the following form:

MustGrant <*Access Type*> **on** <*Object*>
to <*Subject*>
with provision [<*Provisions*>]
where [<*Security Principal*> **has attribute** <(tag, value), …> | <predicate>]

Strong authorization rules facilitate the resolution of conflicting authorizations by superceding decisions from other authorization rules. We illustrate this by an example.

Example 7: Consider the following authorization rules:

Grant read **on** file1
to user1
with provision Add copyright notice
where user1 **has attribute** (group, accounts payable)

DoNotGrant read **on** file1
to user1
with provision Notify sysadmin
where user1 **has attribute** (rank, supervisor)

Suppose further that Alice, who is both a member of the accounts payable group and a supervisor, wants to access file1. The two authorization rules above result in a conflict, which can be resolved in one of the following ways:

1. We can have a default policy that gives precedence to **DoNotGrant** over **Grant** whenever such conflicts arise. In this case, Alice does not get access to file1.
2. We can insert a strong authorization as follows:

> **MustGrant** read **on** file1
> **to** user1
> **with provision** Notify VP
> **where** user1 **has attribute** (member, accounts payable group) **and** (rank, supervisor)

In this case, Alice will get access to file1, but a notification will be sent to the VP.

5 Integrated Architecture Description

Figure 2 shows the components of an integrated architecture, which combine the mobile user credentials presented by an AC with the mobile policy. First, the client presents the IC and AC containing the user identification and authorization information to the application layer.

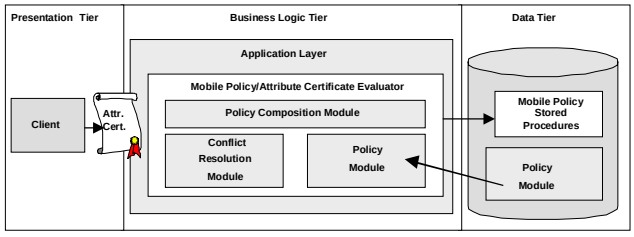

Figure 2. Attribute Certificate/Mobile Policy Architecture

The *Mobile Policy/Attribute Certificate Evaluator* (MoPACE) maps the AC attributes to the MPAs. The mapping itself can either reside in every mobile policy module or in a central repository. This mapping is required to determine what attributes are relevant to the policy when it is time to enforce the policy.

An advantage to using a central repository for the AC attributes to MPA mapping is that policy module implementers would not have to maintain correct mappings in each policy module. A disadvantage is that the system policy would become dependent on such a mapping.

Due to the dynamic nature of policy modules, each policy module supports an interface that enumerates the input parameters that it expects. The scope of a policy module may vary. It may be developed and used within a single application, department, or enterprise, implementing rules specific to the developing organizational unit, or, eventually, well-known policy module components may exist and be widely available.

The *policy composition module* deals with issues such as the order in which policies are to be applied. We do not discuss policy composition further in this paper.

If two policies, with conflicting rules, apply to the same data, the *conflict resolution module* can be used to determine which policy to enforce. The conflict resolution module arbitrates conflicts by enforcing policies designed to address data policy conflicts. We use **MustGrant** rules to specify how conflicts are to be resolved.

In support of a policy for data association, two *MoP stored procedures* are proposed. First, a stored procedure accepts a query for data as input and returns an identifier for applicable mobile policy. Second, a stored procedure accepts a mobile policy identifier as input and returns the actual policy module.

We illustrate the entire process in the next two examples.

Example 1 Continued: Now we return to the motivating Example 1 in Section 2. The following list outlines a typical scenario for the library distribution system example:

1. In the presentation layer, a user Trudy from organization O presents her IC and AC to the business logic tier (machine M1). Her AC will have an attribute (Organization_Name, Acme).[2]
2. Once authenticated, Trudy submits a request for article A to M1.

[2] Currently, the Secure Sockets Layer (SSL) protocol allows clients and servers to exchange IC information for mutual authentication. As of this writing, the new version of SSL proposes to allow AC information to be exchanged between clients and servers as well.

3. If article A, and its associated policy *P*, are already available in M1, we skip to step 5. Otherwise, M1 submits a query for article A to the document server or data tier (machine M2).
4. When M2 receives the request, it returns article A, locates the associated policy identifier using the first stored procedure, locates the policy module using the second stored procedure, and, finally, returns the associated policy *P* to M1.
5. The associated policy *P* with respect to article A states that it is for unlimited distribution among project members, but if the requestor is a sponsor then attach a proprietary notice.
6. When M1 receives article A and policy *P*, it determines that user Trudy is a member of organization "Acme". Next, M1 verifies that organization "Acme" is a sponsor. Finally it allows access to A after attaching a proprietary notice.

Example 2 Continued: Consider the B2B application of Example 2:

1. Suppose a user Bob, from corporation C2, has an AC with the following two attributes: (group, Accounting_Group) and (rank, manager).
2. Once authenticated, Bob submits a request to machine M2 to update data *D*.
3. M2 submits the query for data *D* to the data tier in corporation C1 (machine M1).
4. When M1 receives the request, it returns data *D*, locates the associated policy identifier using the first stored procedure, locates the policy module using the second stored procedure, and, finally, returns the associated policies *P1* and *P2* to M2.
5. Policy *P1* states that every member of the accounting group has read access to *D* with the provision to add the notice: "Do not distribute outside the accounting group".
6. Policy *P2* states that users with rank manager can update *D*.
7. When M2 receives data and policies *P1* and *P2*, the MoPACE maps Bob's AC with *P1*'s and *P2*'s MPAs. Using *P1* and attribute (group, Accounting_Group), it allows read access after the distribution notice is added. Using *P2* and attribute (rank, manager), it allows write access to D.

6 Mobile Policy Enforcement

The purpose of policies is to provide rules for enforcement. In this section, we present a technique that can be used to *derive* new policies from existing ones. We accomplish this objective by allowing system administrators to establish *hierarchies* and *partial orders* for derived authorizations. Then we present an attribute-based mobile policy enforcement algorithm for applying existing and derived policies.

6.1 Hierarchies and Partial Orders for Derived Authorizations

In our model, it is possible to organize groups and roles into hierarchies, as shown in Figures 3 and 4. Notice that we draw the role hierarchy upside down so that in group hierarchies as well as role hierarchies, the child node always inherits the privileges of the parent node. Group and role hierarchies can be used to derive implicit authorizations from explicit authorizations. Thus, referring to the group hierarchy in Figure 3, any authorization given to the accounting group can be derived for all of its descendants.

Although it is possible to organize ranks into hierarchies, rank hierarchies do not imply derived authorizations in our model. This is because it is not reasonable to assume, for example, that if an employee has access to certain objects, so does his/her manager.

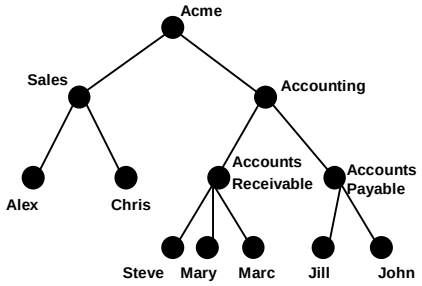

Figure 3. Group Hierarchy Example

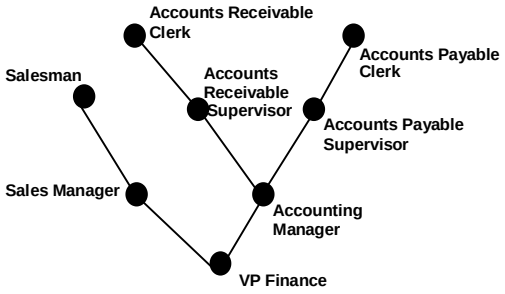

Figure 4. Role Hierarchy Example

Provisions, on the other hand, may have a partial order < associated with them. Although this partial order is not used to derive additional implicit authorizations, it is useful for enforcement purposes, as we will see below. Figure 5 shows an example of a provision partial order. In this example, "VP Approval" is the strongest provision, and **{}** denotes the weakest (no action required) provision.

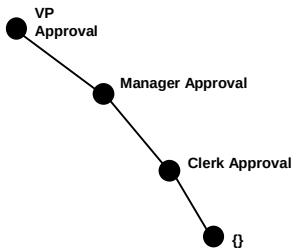

Figure 5. Provision Partial Order Example

Example 8: Consider the following two authorizations:

1. User1 can read file Balance_Sheet provided that provision ***p1*** 'User1 provides his/her name and address' is satisfied.
2. User1 can read file Balance_Sheet provided that provision ***p2*** 'User1 pays $50 via credit card' is satisfied.

If provisions p1 and p2 are ordered as ***p1*** < ***p2***, then we can accept ***p1*** instead of applying ***p2***. The algorithm given below will look for the weakest possible provision to apply.

Example 9: Consider the following policy example: every member of the accounting group can read A document; only the accounting group manager (or anyone superior to this manager) can update the document. The only provision is that clerk's approval is required.

Under this policy, since all members of the accounting group are allowed access to the document, based on the hierarchy in Figure 3, Steve, Mary, Marc, Jill, and John have access. If there is a role hierarchy such as the one shown in Figure 4, then we can derive the policy that the VP can also update the document. If there is provision partial order, as shown in Figure 5, then we can derive the policy that the document can be updated with a manager's approval if clerk approval cannot be obtained.

6.2 Attribute-Based Mobile Policy Enforcement Algorithm

We give an algorithm that determines access control and provisions based on given authorization rules with attributes and provisions. The algorithm uses group and role hierarchies to grant access based on group and role memberships. It is easy to adapt this attribute-based mobile policy enforcement algorithm to other application-specific hierarchies.

Algorithm: Given a user *U* with attributes *A* and data *D* with policy *P*, we initialize the variable GrantAccess as undefined. The algorithm searches for strong authorization rules that apply directly to user *U* and attributes *A*. If a rule is located, we set GrantAccess to True and save the required provision. If, on the other hand, no strong authorization rule is found, the algorithm then searches for relevant strong authorization rules that apply to attributes located higher in the defined hierarchies. In particular, the algorithm looks for group and role hierarchies and traverses those hierarchies starting from the user's attributes. If a rule is located, GrantAccess is set to True and the required provision is saved. Otherwise, the algorithm is repeated for weak negative and weak positive authorization rules[3]. When a weak negative authorization rule is found, GrantAccess is set to False. If a weak positive authorization rule is found, GrantAccess is set to True, and, again, the provision is saved. If no relevant rule is found, GrantAccess is set to False. Once the algorithm

[3] We refer to authorizations that are not strong (i.e., **Grant** and **DoNotGrant** authorizations) as weak authorizations [2].

completes its calculations, it searches for the weakest enforceable stored provision to apply based on the defined provision partial order. Figure 6 shows the algorithm in pseudocode.

AUTHO(U, A, D, P)

1. **GrantAccess=Undefined.**
2. **Find strong authorization rule that applies to attributes A and user U.**
3. **If found, GrantAccess=True; store provision.**
4. **Else, find strong authorization rule that applies to groups higher than group in A.**
5. **If found, GrantAccess=True; store provision.**
6. **Else, find strong authorization rule that applies to roles higher than role in A.**
7. **If found, GrantAccess=True; store provision.**
8. **Else, repeat steps 2 through 7 to identify negative authorization rules.**
9. **If a negative authorization rule was found, GrantAccess=False; store provision.**
10. **Else, repeat steps 2 through 7 to identify positive authorization rules.**
11. **If a positive authorization rule was found, GrantAccess=True; store provision.**
12. **Else, GrantAccess=False.**
13. **Find weakest enforceable stored provision and apply it.**

Figure 6. Policy Enforcement Algorithm

Note that explicit conflict resolution is not required in this algorithm, since strong authorizations are checked first, followed by negative authorizations. The following example shows how the algorithm is applied.

Example 2 Continued: Coming back to our B2B application example, suppose there is a file *F* in the database of corporation *C1*. Also suppose that the access policy attached to F states the following:

> **Grant** read **on** F
> **to** user1
> **with provision** clerk approval
> **where** user1 **has attribute** (group, accounting)
> **and** (role, accounting supervisor) **and**
> (corporation, Acme)

This policy states that user1 is allowed read access to F, if he/she belongs to the accounting group in the Acme corporation and if he/she can assume the role of an accounting supervisor provided there is clerk approval.

Suppose now user Mary comes in with an attribute certificate with the following attributes (corporation, Acme), (group, accounts receivable), (role, VP) and submits a request to read document F. The MoPACE verifies that Mary is from Acme; that she is a VP, which is higher than a supervisor (as shown in the role hierarchy diagram Figure 4); and that her group "accounts receivable" is under the group accounting as required by the policy attached to F.

Therefore, Mary is given access to F but first we must fulfill the provision of getting clerk approval. Suppose there is no clerk available in Acme that day, a stronger provision can be accepted instead. As shown in Figure 5, manager approval is a stronger provision than clerk approval. Therefore, Mary can now access F with the provision that a manager approves.

7 Related Work

Although several relevant research efforts [1,2,7-13] are currently under way in the policy area, the researchers assume that a single authority has administrative control of all elements of the multi-tier distributed system. Thus, they do not address how a user's credentials and the policy-to-data mapping context can be made available in the middle tier. Moreover, the policies under consideration are limited to access control policies.

Chapin et al. [3] present a framework for using mobile policy in the business logic tier of multi-tier information systems. Mobile policy allows policy to be defined by an authority close to the system element that is responsible for the information being controlled. A binding exists between the data and the underlying policy; as the data moves to other elements that use the data, so does the policy.

This paper extends the framework described in [3] in several ways. First, our framework incorporates attribute certificates. With attribute certificates, policy rules can specify detailed requirements for access purposes. Second, mobile policies in [3] are encapsulated in *executable* modules that can be coded using any programming or policy definition language that the policy administrator chooses. These modules are shared by different system elements by defining a shared vocabulary of input to and output from policy modules. Third, we define a mobile policy definition language that has several benefits: (a) It is a semantically rich policy

specification language capable of representing many policies that may apply to multi-tier systems. (b) It uses familiar syntax that is similar to SQL access control predicates. We hope that this familiarity will reduce the learning curve for using the language. (c) Converting the access rules (in this language) to mobile code (such as Java) is an easy process. Finally, we present an algorithm for enforcing attribute-based mobile policies.

8 Conclusions and Future Work

We presented a framework for combining attribute certificates with mobile policy for effective access control specification and administration in a distributed (n-tier) environment. We also proposed a high-level specification language to specify mobile policies and incorporate attributes. Finally, we presented an algorithm that illustrates how to apply policy rules with a set of attributes to a given request from a user.

Currently, we are designing a prototype that will implement mobile policies with attributes using Java code. The prototype will translate mobile policy constructs, like **Grant**, to Java. Thus automating the process from policy specification, using the syntax we defined in this paper, to corresponding Java code that implements it. Provisions remain an area that will probably require manual intervention by a Java programmer.

We are also looking into applying the mobile policy language we defined to address intelligence community requirements, such as Originator Control (ORCON) and No Foreign (NOFORN).

In the future, we would like to investigate the use of mobile policy in multi-level security (MLS) environments. In particular, a study area would address the use of boundary controllers for a classification level in MLS to apply and enforce mobile policy. This capability would convert boundary controllers into policy enforcement engines without needing to store or retrieve policies that apply to the data being checked for releasability.

References

1. E. Bertino, P. Samarati, and S. Jajodia, "An extended authorization model for relational databases," *IEEE Trans. on Knowledge and Data Engineering*, Vol. 9, No. 1, 1997, pages 85-101.
2. E. Bertino, S. Jajodia, and P. Samarati, "Supporting Multiple Access Control Policies in Database Systems'," *Proc. IEEE Symp. on Research in Security and Privacy*, Oakland, Calif., May 1996.
3. S. Chapin, S, Jajodia, and D. Faatz, "Distributed Policies for Data Management Making Policies Mobile," *Proc. 14th IFIP 11.3 Working Conference on Database Security*, Schoorl, Netherlands, August 2000.
4. S. Farrell and R. Housley, "An Internet Attribute Certificate Profile for Authorization," PKIX Working Group, Internet-Draft, March 2000.
5. R. Housley, W. Ford, T. Polk, and D. Solo, "Internet Public Key Infrastructure - X.509 Certificate and CRL profile," RFC 2459, January 1999.
6. ITU-T Recommendation X.509 (1197 E): *Information Technology - Open Systems Interconnection - The Directory: Authentication Framework*, June 1997.
7. S. Jajodia, M. Kudo, and V. S. Subrahmanian, "Provisional authorizations," *Proc. 1st Workshop on Security and Privacy in E-Commerce*, Athens, Greece, November 2000.
8. S. Jajodia, P. Samarati, V. S. Subrahmanian, and E. Bertino, "A Unified Framework for Enforcing Multiple Access Control Policies," *Proc. ACM SIGMOD Int'l. Conference on Management of Data*, May 1997, pages 474-485.
9. S. Jajodia, P. Samarati, and V. S. Subrahmanian, "A logical language for expressing authorizations," *Proc. IEEE Symp. on Research in Security and Privacy*, Oakland, Calif., May 1997, pages 31—42.
10. C. J. McCollum, J. R. Messing, and L. Notargiacomo, "Beyond the Pale of MAC and DAC – Defining new forms of access control," *Proc. IEEE Symp. on Security and Privacy*, 1990, pages 190-200.
11. P. Samarati, P. Ammann, and S. Jajodia, "Maintaining Replicated Authorizations in Distributed Database Systems," *Data & Knowledge Engineering*, Vol. 18, No. 1, February 1996, pages 55-84.
12. R. Simon and M. E. Zurko, "Separation of Duty in Role-Based Environments," *Proc. 10th Computer Security Foundations Workshop*, June 1997.
13. M. E. Zurko, R. Simon, and T. Sanfilippo, "A User-Centered, Modular Authorization Service Built on an RBAC Foundation," *Proc. IEEE Symp. on Security and Privacy*, May 1999.