CHAPTER 6

Distributed Policies for Data Management – Making Policies Mobile[1]

Susan Chapin, Don Faatz, and Sushil Jajodia

Abstract: This paper presents the challenges facing developers of multi-tier information systems in providing effective consistent data policy enforcement, such as access control in these architectures. It introduces "Mobile Policy" (MoP) as a potential solution and presents a framework for using mobile policy in the business logic tier of multi-tier information systems.

Key words: Security, access control, mobile policy, n-tier architecture

1. INTRODUCTION

In typical multi-tier architectures, the client is reduced to no more than a Web browser and the database management system (DBMS) is returned to its primary function of storing data. Business logic is moved from the client and the database to a middle tier, hosted on a different platform from the DBMS as shown in Figure 1.

Multi-tier architectures require changes in the way security and other policies are managed. Mechanisms are needed that can achieve consistent policy across elements of a distributed environment and support flexible policies other than access control. The need for consistent policy management across distributed components is analogous to the need for consistent transaction management across distributed components. The need for flexible policies arises from the complex functionality of many multi-tier applications. While control over access to data remains a very important policy, support for other types of policies, such as requiring certain files to

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

have a copyright notice or be sanitized in some way before being returned to certain clients, is also needed.

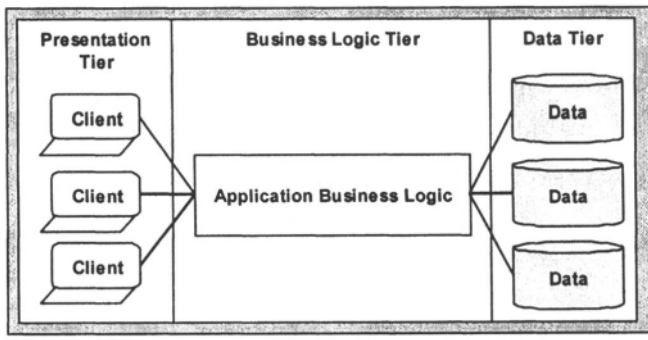

Figure 1. Multi-tier Architectures for Multiple Clients and Databases

Specific requirements for making policies consistent across different components include making policies mobile, so that they may travel with the data from one component to another rather than being applied before the data are released from the DBMS, and making security contexts mobile, so that references to users and roles have the same meaning to all components. Making policies flexible requires enabling those who manage data to define the kinds of policies supported, rather than relying on DBMS vendors.

Traditional data tier policy management does not well support these needs. Supported policies, defined by DBMS vendors, are limited to access control policies; access control is applied by the DBMS at the time access to the data is requested, requiring that the source of the request be known to the DBMS; and traditional data tier users, roles, and policies are all locally defined within the DBMS based on local security context.

We propose an application framework that extends the capabilities for policy management in multi-tier applications without interfering with existing access control policy mechanisms. Policy management is decomposed into three functions, defining policies, associating policies with data, and applying policies. Security context management is enhanced by including third-party mechanisms, such as digital certificates provided by a public key infrastructure (PKI), that can be referred to by all components; the description of the context management can be found in [2]. Almost any policy can be supported, limited only by the ability of developers to implement the policy in software. The framework allows policies to be applied in any tier of the application, determined by the application developers in conjunction with the database administrators. As of this writing, we have developed our proposed framework design in sufficient detail to support a proof-of-concept prototype.

The rest of this paper is organized as follows. Section 2 describes the needs for policy management in multi-tier architectures. Section 3 describes the limitations of existing mechanisms for dealing with policy management in multi-tier architectures. Section 4 presents an overview of our proposed framework and describes how security contexts, as well as policies, can be shared among application components. Section 5 summarizes what we have achieved and what we want to achieve in the future.

2. MULTI-TIER ARCHITECTURES

The downside of multi-tier application architectures is that they can be quite complex. The presentation tier can consist of any one of a number of browser products on any one of a number of platforms. The data tier can consist of multiple databases in different DBMSs on different platforms. The business logic tier can consist of multiple components on multiple different platforms.

The problem is that all these different components need to be composed together into a single application. Where security is involved, the composition must be seamless and reliable. Composing policies in multi-tier applications can be an issue, because the developers of the different components of the business tier and the different databases have different responsibilities and knowledge about the policies that should apply. We address policy composition in multi-tier architectures by providing a framework that allows the developers of each component to concentrate on policy matters that are properly their responsibilities.

2.1 Aligning the Authority for Policy with the Responsibility

Reserving policy application to the DBMS requires extensive communication among various subgroups within the enterprise. Managing a policy has three components, and each component is, ultimately, the responsibility of different enterprise subgroups. *Policy specification* is properly the responsibility of enterprise management. *Policy association* is the process of associating enterprise policy with data. It is properly the responsibility of the data owners, usually represented by the database administrator (DBA). *Policy application* is the process of applying the policy to data at the appropriate time. It is properly the responsibility of the business logic developer or whoever is in charge of the point(s) at which the data are used. Applying the policy at time of use is an ongoing activity; the

data may be considered to be "used" when they are accessed within the DBMS, but they are also "used" within the application, whether the application code is located within the DBMS or in a separate middle tier component, and whether the application uses the data immediately or holds on to it for several days before use.

Policy enforcement is only complete when all three elements, definition, association, and application, work harmoniously together. The problem is that building coordinated support for policies can require close cooperation among those responsible for each component. It is not that any of the policy management problems are inherently impossible to solve. The problem is that they require cooperative design decisions affecting application code in both the middle tier and the DBMS, and these two portions of the application may be developed by different groups of people on different time schedules. The result is a greater risk of miscommunication and decreased assurance in the resulting product.

A mechanism is needed that decouples the development of software that implements the policy, the process of associating the policy with data, and the development of software that applies the policy at time of use.

2.2 Making Policies General

"Policy" is often taken to mean "security policy," and "security" is often taken to mean "access control," and all access control is assumed to be handled by the same mechanisms. Although security policies are important policies, and access control policies are important to an enterprise's overall security, these are not the only policies that an enterprise may want to enforce. Furthermore, not all policies, access control or other types, are equally important.

A substantial part of most middle-tier application development involves implementing various kinds of policies. Some policies are enterprise policies that are specified by enterprise management as rules that must be followed. Examples of enterprise policies include requirements for ensuring files have the proper copyright notice before they are released outside the enterprise, degrading the resolution of certain images before they are released to specified classes of clients, and scanning files for viruses before they are used.

Other rules are local to the application but span both the business logic tier and the data tier. It is a bit of a stretch to call these application rules "policies," but it is convenient for our discussion because they share many of the characteristics of enterprise policies. In particular, they may be as critically important and as much in need of assurance that they are working

correctly as enterprise policies, and can equally well be handled by our proposed framework.

An example of one of these other "policies" is a rule that defines the confidence that the middle tier application can have in the accuracy of a data item retrieved from a DBMS. Imagine an application that controls airplane takeoffs for various destinations. One of the data items it needs is the amount of fuel in the plane's tank. The rule might be that the confidence level of this type of data is a function of metadata, such as the time since the data were last updated, rather than something that can be derived from the data themselves. The application as a whole, including both the middle tier and the DBMS, needs a mechanism to calculate the confidence factor and get that information to the middle tier before the middle tier releases the plane for takeoff, or some considerable unpleasantness might ensue.

Support is needed for any policies that may be applicable to data, using the same techniques for any policy, without requiring that policy types be predefined by DBMS vendors.

A characteristic of this expanded definition of policies is that not all policies are equally critical. Some types of policies may be less critical than others in an enterprise; for example, the need to check files for copyright notice may be less critical than protecting write access to the salary file. Even within access control, some data may need to be protected more carefully than others. For example, the author of a document may wish it to be restricted to only a small group of people while it is under development, but the accidental release of the partially written document to other employees would not have as severe consequences as the accidental release of the company product's source code to the general public.

Therefore, a mechanism that is not deemed sufficiently secure for one policy may still be acceptable, and very valuable, for other policies. The requirement is that the mechanisms must not interfere with each other.

3. RELATED WORK

Several research efforts are currently under way to centralize the administration of policy. The Open Group's Adage project [7, 9] is a typical example of this research. The notion of mobile policy is not particularly new [1,4,5-7,10]. Several approaches to sharing policy information have been developed. However, none is as general as the approach proposed here.

One problem common to all attempts to centralized policy definition and storage is the need for a semantically rich policy specification language capable of representing all policies that may apply within the multi-tier system. Such a language is very difficult to define and has so far eluded

researchers. Mobile policy tries to avoid this problem by encapsulating policy in an executable module. These modules can be coded using any programming or policy definition language that the policy administrator chooses. Instead of defining an all-powerful magic policy language, the problem is transformed into defining a shared vocabulary of inputs to and outputs from policy modules. These vocabularies should be more tractable than a general-purpose policy language.

The next section describes our framework for use of mobile policy in multi-tier information systems.

4. THE PROPOSED FRAMEWORK

We call our proposed framework *MoP,* for mobile policies. With MoP, when data move from one component or tier to another, any associated policies travel along with the data until the policies are applied. The movement of policies is shown in Figure 2.

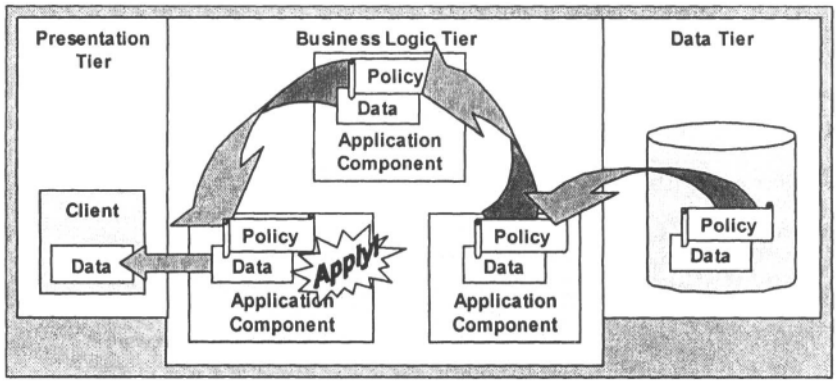

Figure 2. Mobile Policy

MoP is designed 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. The framework consists of code component types and vocabulary standards that represent the minimal knowledge that must be shared among the developers of systems that use MoP. The component types are shown in Figure 3. The vocabulary standard is the glue that makes the system work. The next sections describe each of these elements.

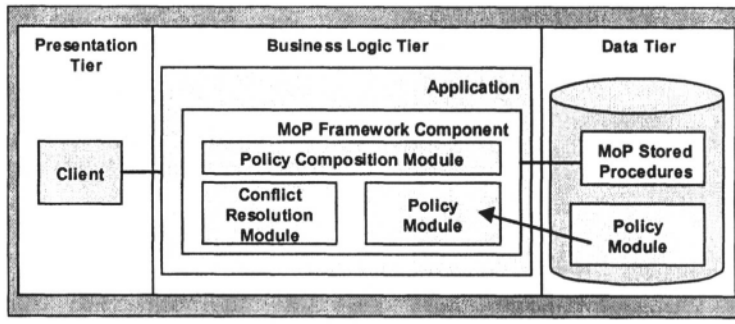

Figure 3. MoP Component Types

4.1 Policy Module

Policy modules implement policy rules. Each policy module is an executable code module, written for the platform of choice of the application, that implements one specific policy rule. For example, a policy module may determine whether a requested access is granted based on user identity, or whether access is granted based on the type of connection between the client and the application, or it may add the correct copyright notice to a file. Thus, each policy module is a self-contained package with a limited, specific function, which has the nice benefit that it simplifies validation of correct behavior.

Policy modules are classified into types by the end function they perform, not by the rule that governs how they perform it. The three examples above include only two policy types: determine whether access is granted and add a copyright notice. The two access granting rules, one of which looks at user identity and the other of which looks at the client connection, would be implemented in two separate policy modules, both of which are of type "access grant."

All policy modules of the same function type return the same output parameters with the same syntax and semantics. An application programmer needs to know what the module does in order to determine whether the module is applicable to the planned use of the data, and what output parameters the module returns and what they mean in order to code an appropriate response, but the application programmer does not need to know the policy rule the module implements.

In contrast, not all policy modules of the same type require the same input parameters. All policy modules implement a method that returns a list of input parameters. The application must be able to accept a list of parameters and return a value for each.

To summarize, a policy module is an executable code module that implements a single rule, has a well-known type and set of output parameters, and produces a list of required input parameters.

4.2 Policy Composition Module

Policy composition modules deal with issues such as the order in which policies are to be applied. We have not yet designed or prototyped this capability, and do not discuss it further.

4.3 Conflict Resolution Module

Conflict resolution modules resolve conflicts among policy modules. Multiple policy modules may be associated with the same data set. If it should happen that more than one policy module of the same type is associated with the same dataset, then any conflicts must be resolved before the correct single output parameter set is defined. This conflict resolution is performed by a *conflict resolution module.*

We assume that different policy module types are independent of each other. Any interactions between, say, a copyright notice rule and an access grant rule we consider to be idiosyncratic, complex, and outside the scope of the MoP framework. MoP, of course, does not prevent the application developer from writing code to resolve any such conflicts.

Conflict resolution module development is closely linked to policy module development. Conflict resolution modules implement the resolution of conflicts among policy rules, and therefore conflict resolution rules are policy rules.

4.4 DBMS Stored Procedures

When data are accessed, the MoP application component needs to retrieve the policy modules associated with the data. Two DBMS stored procedures provide this capability. One receives a SQL request and returns identifiers associated with relevant policy modules, the other receives a policy module identifier and returns the specified policy module.

MoP therefore requires three or more database queries instead of one for each request: access the data, request relevant policy module identifiers, and request each needed policy module. The MoP application component makes

all three (or more) requests within the same transaction, thereby eliminating potential synchronization difficulties.

The separation of function not only supports flexibility but also decreases performance overhead by allowing the application to make only those requests it actually needs and to make them in any order. For example, for READ requests the policy may be run before the data are retrieved, because the result of the policy may make retrieving the data unnecessary, or the application may first retrieve data and review it to determine which of the associated policies are relevant to its intended use of the data before requesting the policy modules.

Separating the request for policy module identifiers from the request for specific policy modules allows the application to cache policy modules and to request only those policy modules it actually needs, a potentially significant performance enhancement.

4.5 Application Framework Component (MoP)

The MoP application framework component encapsulates MoP implementation details that are not application dependent. The MoP component exposes methods that support accessing data, identifying relevant policy modules, retrieving relevant policy modules, and running selected policy types.

As of this writing, the application is responsible for setting up pointers to permanent objects (in the current version, the permanent objects are caches and connections to databases), providing an object that actualizes parameters, and calling the MoP retrieve time and MoP apply time methods.

4.6 MoP Shared Vocabularies

MoP shared vocabularies are the heart of our solution for sharing policy among developers responsible for different application tiers while minimizing the knowledge they must share with each other. Encapsulating policy rules into components allows us to reduce the semantics that must be shared from understanding policy logic, which requires a “magic language” and is very difficult, to understanding a small vocabulary of shared terms, which is a relatively easy and familiar technology. We define three vocabularies: a policy module types vocabulary, an output parameters vocabulary, and an input parameters vocabulary.

In the *policy module types* vocabulary, each term specifies what the policy module does, such as add a copyright notice or determine whether access is granted and implies a set of output parameters.

In the *output parameters* vocabulary, each term specifies both syntax and meaning of a parameter returned from a policy module. Output parameters are the same for all policy modules of the same type. The application uses the output parameters to apply the policy. The output parameter vocabulary is important because for many policy types, such as access control, the application must be prepared to take action based on returned output parameters.

In the *input parameters* vocabulary, each term specifies an input parameter needed by the policy module. The application provides a method to be called by the MoP component that accepts a list of input parameters and returns a list of matching values. The input parameter vocabulary is important because two modules with the same function may have different input parameters.

4.7 Allocation of Responsibilities

Supporting separation of duty by allocating specific policy management responsibilities to different development groups is MoP's prime benefit. With MoP, each group of developers needs to understand only the subset of policy management that falls properly within the group's purview.

MoP allocates responsibilities to policy-makers, database administrators, DBMS developers, and application developers. MoP does not impose any requirements on the client tier.

Policy-makers specify the policy rules that are implemented by MoP policy modules. They also have the ultimate responsibility for locating or creating policy modules that implement the rules and conflict resolution modules that implement the resolution of conflicts among policy rules.

DBMS developers create stored procedures that implement the two DBMS functions required by MoP, returning identifiers for the policy modules associated with a data access request and returning a policy module on request.

Database administrators create and install the MoP stored procedures into the DBMS, insert policy modules identified by the policy-makers, and associate policy modules with data. Mechanisms for these functions will vary from DBMS to DBMS. This process is out of the scope of MoP.

Application developers call the MoP application component, pass it required parameters, and use policy module outputs to apply policy.

4.8 The Implementation

We are using a prototype implementation of the MoP components to validate our framework design as we develop it. We do not consider any portion of our design complete until it has been included in the prototype.

The current prototype is an all-COM solution built using Microsoft Visual Basic Enterprise 6.0 and Microsoft Access 8.0. Early work has focused on building the MoP application component, using stubs for database support and policy modules, and a demonstration application that exercises each feature of the MoP application component.

Our target databases are Oracle and SQL Server. Access does not provide stored procedures or sophisticated policy management mechanisms, but its functionality is adequate to support work on the MoP application component.

5. CONCLUSIONS AND FUTURE WORK

This paper proposes the use of mobile policy in multi-tier information systems. Specifically, it separates policy administration from policy enforcement. Policy is specified and administered at the element of a distributed system where the data being controlled by policy is defined. That policy is then shared with consumers of the data so that they can enforce the appropriate policy when using the data.

Performing authentication and access control in the database is suitable where the data are extremely sensitive or the number of potential clients is limited. However, it can be restrictively expensive where the application resides in a middle tier instead of the database or the number of potential clients is large. Therefore, where company policy permits, a solution such as MoP can enhance overall security by expanding practical policy enforcement beyond access control within the DBMS.

Although we have not completed work on the basic MoP framework, we have identified a number of enhancements that we would like to add once the basic framework is complete: dynamically-generated policy modules, dynamic determination of conflict resolution metadata, a policy composition module that manages relationships among different policy modules, and support for associating policy modules with subsets of retrieved data.

Dynamically generated policy modules are interesting because they would eliminate parallel implementations of the same policy. DBMS systems already have a mechanism that associates access control policies with data. We would like to develop a mechanism that extracts the access control information relevant to an SQL query and packages it as a MoP

policy module. In addition to convenience value, automatic generation of MoP policy modules potentially could enhance assurance because the information would not have to be associated with the data twice, once as a policy module and once as DBMS access control lists.

Dynamic determination of conflict resolution metadata is interesting because it would simplify the task of policy module developers. As it stands today, MoP requires linked code development in policy modules on one type and their associated conflict resolution modules. We think it would be desirable to provide a cleaner interface so that policy module and conflict resolution module development can be more independent.

Support for associating policy modules with subsets of retrieved data is interesting because it would support applications, such as data warehouses, where a large block of data is retrieved all at once and stored internally in database table format. Later, when the data are to be used, the application extracts subsets of the data for each specific use. MoP as currently designed does not support this kind of application.

Before our framework can be shown to be useful in production environments, a number of issues need to be addressed: performance, multi-platform application support, and assurance.

Performance is an issue, because a prime reason for using multi-tier architectures is to gain enhanced scalability and efficiency. If making policies mobile slows processing down any appreciable amount, any benefits will not be worth the cost.

Multi-platform support is an issue because another prime reason for using multi-tier architectures is to gain application development flexibility. If the MoP application component can be called only by COM applications, and not by EJB or CORBA applications, its usefulness will be limited.

Assurance is an issue because many MoP policies are security policies. A mechanism for implementing security policies that cannot itself be shown to meet enterprise requirements for security will not be very useful.

REFERENCES

1. Black, D. L., D. B. Golub, D. P. Julin, R. F. Rashid, R., P. Draves, R. W. Dean, A. Forin, I. Barrera, H. Tokuda, G. Malan, and, D. Bohman, "Microkernel Operating System Architecture and Mach," Journal of Information Processing, Volume 14, Number 4, 1995.
2. Doshi. Vinti, Amgad Fayad, and Sushil Jajodia, "Using Attribute Certificates with Mobile Policies in Electronic Commerce Applications," Proc. 16th Annual Computer Security Applications Conf., New Orleans, LA, December 2000.
3. Minear, Spencer E., "Providing Policy Control Over Object Operations in a Mach Based System," Secure Computing Corporation, Roseville, MN, April 1995.
4. Minear, Spencer E., "Controlling Mach Operations for use in Secure and Safety-Critical Systems," Secure Computing Corporation, Roseville, MN, June 1994.

5. Object Management Group (OMG), Resource Access Decision (RAD), OMG document corbamed/99-03-02, March 1999.
6. Object Management Group (OMG), Transaction Service Specification.
7. Simon, Richard and Mary Ellen Zurko, "Separation of duty in role-based environments," Proceedings of the 10th Computer Security Foundations Workshop, June 1997.
8. U.S. Department of Commerce/National Institute for Standards and Technology, Standard Security Label for Information Transfer, FIPS PUB 188, September 1994.
9. Zurko, Mary Ellen, Rich Simon, Tom Sanfilippo, "A user-centered, modular authorization service built on an RBAC foundation," Proc. IEEE Symp. on Security and Privacy, May 1999.