Security Policy Coordination for Heterogeneous Information Systems

John Hale * Pablo Galiasso Mauricio Papa Sujeet Shenoi †
Department of Computer Science, University of Tulsa, Tulsa, OK 74104

Abstract

Coordinating security policies in information enclaves is challenging due to their heterogeneity and autonomy. Administrators must reconcile the semantic diversity of data and security models before negotiating secure interoperation. This paper proposes an architecture that uses mediators and a primitive ticket-based authorization model to manage disparate policies in information enclaves. The formal foundation of the architecture facilitates static and dynamic analysis of global consistency and policy enforcement.

1. Introduction

Security managers have an arsenal of tools at their disposal for protecting information systems. However, they are not well-equipped to coordinate security policies between interconnected enterprises. Security in open heterogeneous environments is too often a patchwork of conflicting policies implemented in an *ad hoc* manner. Nevertheless, security managers of mission-critical information enclaves must guarantee the coherence of their policies within global, often hostile, environments.

Consider the information enclave shown in Figure 1. In this example, a toxin specialist is rendering medical advice to a M.A.S.H. unit treating a patient exposed to an unknown toxin. The specialist must integrate information from a variety of sources. Specifically, the specialist needs to know patient symptoms, toxins local to the region, and patient allergies.

Each site contains a piece of the puzzle. The M.A.S.H. unit maintains data regarding status and location of patients: $R_1(Patient, Symptom, Region)$. Military headquarters (HQ) maintains an object-oriented database of toxins detected in regions of battle, expressed as $R_2(Toxin, Region)$. The Poison Control Center (POISON CTL) maintains a list of poisons associated with symptoms and antidotes: $R_3(Toxin, Symptom, Antidote)$. Finally, the patient's physician maintains a database of patient allergies: $R_4(Patient, Allergy)$. The specialist must integrate this information to suggest a treatment for the patient at the M.A.S.H. unit.

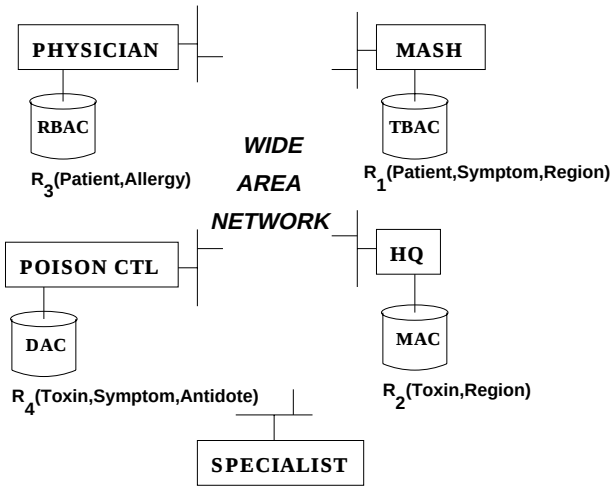

Figure 1. Heterogeneous information system.

Each site manages its own security policy. The M.A.S.H. unit employs Task-Based Access Control (TBAC) to permit fine-grained control in the execution of mission-critical tasks. HQ uses Mandatory Access Control (MAC) to provide access based on clearances. POISON CTL engages Discretionary Access Control (DAC) while the physician's database provides role-based access control (RBAC). The prevailing security policies may change at any time. E.g., an officer at HQ may decide to declassify "expired" data, or the administrator of the physician's database may elect to adjust the permissions of a given role.

Treating the specialist as a regular user in each system has disadvantages. Assuming that the relevant HQ data is classified <*SECRET,COMBAT*>, access to it implies clearing the specialist at this level. Subsequently, the specialist would have access to other sensitive data, circumventing the prevailing security policy. Implementing a flexible au-

*To whom correspondence should be addressed (email: *johnhale@utulsa.edu*).

†Research supported by MPO Contracts MDA904-96-1-0114, MDA904-96-1-0115 and MDA904-98-C-A900.

thorization model that delivers least privilege in the face of policy model heterogeneity remains an inherent challenge for information enclaves.

This paper describes a primitive authorization model as a common foundation embedded within a mediator architecture to facilitate policy coordination between enterprises. The next section presents a ticket-based access control scheme and uses it to model a variety of security policies. Section 3 proposes an architecture for integrating security policy mediators within an information enclave. It also considers the potential for security analysis supported by formal authorization semantics accompanying the ticket-based access control scheme. The final sections compare this approach with related work in the field and make some concluding remarks.

2. Modeling Security Policies

Our scheme employs a flexible ticket-based access control scheme to permit the implementation of a variety of authorization models for distributed objects [9]. A design priority is to provide developers of heterogeneous information systems with opportunities for secure interoperability by offering common access control mechanisms that underlie their authorization models. The model can be used for ubiquitous access control or in more practical, lightweight authorization service implementations.

2.1. Tickets

Tickets are tokens linked with permissions held by subjects. Ticket distribution and revocation are thorny issues in authorization models [8]. A naive distribution policy would prohibit all but the ticket "owner" from distributing tickets. Another policy might permit ticket holders to distribute tickets. This is more flexible, but requires constraining the circumstances under which holders pass tickets.

Revocation is more complicated for tickets with multiple privileges. Revocation must be partial, selective and transitive [8]. Partial revocation is the ability to remove a subset of privileges bestowed by a ticket. Selective revocation is the specification of a subset of ticket holders for revocation. Transitive revocation stipulates that revocation propagates through all principals receiving privileges. Clearly, a trade-off exists between the flexibility of a ticket-based authorization model and the effectiveness of its revocation scheme.

Our model incorporates mechanisms for adding and removing tickets from access control lists (ACLs). However, it does not have built-in facilities to ensure that the critical properties of revocation and distribution are respected. System developers build the necessary constraints when creating authorization schemes to enforce specific policies.

ACCESS TYPES

```
priv ::= ALL
       | priv'

priv' ::= KEY
        | LOCK
        | G . priv'
        | R . priv'
```

ACCESS CONTROL PREDICATES

```
EVAL:  cmd -> state -> state -> bool

TRANS: state -> state -> bool
```

ACCESS CONTROL COMMANDS

```
cmd ::= ADD priv token object object
      | REMOVE priv token object object
      | cmd ; cmd
```

Figure 2. Access control definitions.

2.2. Authorization Model

An authorization model resolves (s, o, a) tuples as true or false for subjects (s), objects (o) and access types (a). This information defines the global authorization state. Our ticket-based scheme defines an authorization state by a function $State : Object \rightarrow Privilege \rightarrow Token \rightarrow Bool$. Note that $< Object, Privilege, Token >$ denotes an authorization statement, where $Token$ identifies a label in a ticket that serves as a representative for a subject. The state $< o, p, t >$ means that object o associates privilege p with token t.

Privileges are either keys held by subjects or locks held by objects. E.g., if $Bill$ is a subject and $Door$ is an object, then $< Bill, \texttt{KEY}, sesame >$ implies that $Bill$ can use $sesame$ as a key in its access requests. On the other hand, $< Door, \texttt{LOCK}, sesame >$ means that $Door$ has a lock that can be opened by a subject holding $sesame$ as a key. Thus, the two tuples together imply that $Bill$ can access $Door$.

The recursive definitions in Figure 2 specify a general model of $grant$ and $revoke$ privileges, with higher order access types such as `G.LOCK` and `G.R.KEY`. `G.LOCK` (grant lock) permits adding a `LOCK` privilege to an object. `G.R.KEY` (grant revoke key) permits adding a `R.KEY` to an object. Note that every type other than `KEY` behaves as a lock. In general, a higher order privilege can always be decomposed into the form $< X.priv >$ where X identifies the permitted action (grant or revoke) and $priv$ identifies the privilege type.

The `ALL` privilege confers all privileges to a subject. A subject holding a key matching a lock held by an object

with access type ALL has complete access to that component. The authorization state of an object can be modified by adding or removing ticket-privilege associations maintained in its access control list.

The command set in Figure 2 provides for dynamic and explicit authorization state modification. Commands can be embedded in messages as authorization requests. Subjects can add or remove ticket-privilege associations in objects for the tokens they hold as keys.

$$
\begin{aligned}
Rule\ 1: &\quad \forall p : priv,\ o : obj,\ t : token,\ s : state;\\
&\quad s\ o\ \texttt{ALL}\ t\ \Rightarrow\ s\ o\ p\ t\\
Rule\ 2: &\quad \forall s_1, s_2. \exists c : comm;\\
&\quad \texttt{EVAL}\ c\ s_1\ s_2\ \Rightarrow\ \texttt{TRANS}\ s_1\ s_2\\
Rule\ 3: &\quad \forall p, s_1, s_2, o_1, o_2, t;\\
&\quad s_1\ o_1\ \texttt{KEY}\ t\ \wedge\ s_1\ o_2\ \texttt{G.p}\ t\ \Rightarrow\\
&\quad (s_2\ o_2\ p\ t\ \wedge\ (\forall o', p', t'.\ o' \neq o_2\ \vee\ p' \neq p\\
&\quad \vee\ t' \neq t\ \Rightarrow\ s_1\ o'\ p'\ t' = s_2\ o'\ p'\ t')\\
&\quad \Rightarrow\ \texttt{EVAL}\ (\texttt{ADD}\ p\ t\ o_2\ o_1)\ s_1\ s_2)\\
Rule\ 4: &\quad \forall p, s_1, s_2, o_1, o_2, t;\\
&\quad s_1\ o_1\ \texttt{KEY}\ t\ \wedge\ s_1\ o_2\ \texttt{R.p}\ t\ \Rightarrow\\
&\quad \neg(s_2\ o_2\ p\ t\ \wedge\ (\forall o', p', t'.\ o' \neq o_2\ \vee\ p' \neq p\\
&\quad \vee\ t' \neq t\ \Rightarrow\ s_1\ o'\ p'\ t' = s_2\ o'\ p'\ t')\\
&\quad \Rightarrow\ \texttt{EVAL}\ (\texttt{REMOVE}\ p\ t\ o_2\ o_1)\ s_1\ s_2)\\
Rule\ 5: &\quad \texttt{EVAL}\ (c_1)\ s_1\ s_2\ \wedge\ \texttt{EVAL}\ (c_2)\ s_2\ s_3\\
&\quad \Rightarrow\ \texttt{EVAL}\ (c_1; c_2)\ s_1\ s_3
\end{aligned}
$$

Figure 3. Authorization semantics.

Figure 3 shows rules specifying the authorization model semantics. Rule 1 expresses the semantics for the ALL access type. Rule 2 formalizes the relationship between the predicates EVAL and TRANS used to specify the authorization state transition semantics. The predicate EVAL returns true when a command will take one state to another; TRANS returns true if a transition between states is possible.

Rule 3 defines the semantics of the ADD command. A subject must have $grant$ privilege over an access type in an object to add a token of that type to the object. It also stipulates that subjects can only add tokens held by them as keys. For example, authorization tuples $<$ o, G.R.LOCK, a $>$, $<$ s, KEY, a $>$ and $<$ s, KEY, b $>$ allow the command ADD R.LOCK b o s.

Rule 4 provides the semantics of the REMOVE command. It specifies when it is legal for a subject to remove authorization tuples. Using the previous example, an additional authorization tuple $<$ o, R.R.LOCK, b $>$ would let s remove R.LOCK permissions from o. Once again, s must hold the token as a key. Rule 5 introduces command sequences to the system, formalizing the transitive nature of commands on authorization states.

2.3. Specification Language

The authorization scheme employs an object-based specification language to model protection schemes for a variety of information systems. The specification of DAC, RBAC, TBAC and MAC policies for relational and object-oriented databases is simplified by the language whose abstract syntax is given in Figure 4.

```
obj -> OBJECT ID(template)
          : METADATA mdinit
          : ACL := { aclinit }
          { (meth | subobj)* }

meth -> METHOD ID(return) ID(name) prms
prms -> (ID(type) ID(name)
           (, ID(type) ID(name))* )
          | ( )

subobj-> SUBOBJ ID(name) : ID(template)

mdinit -> := ID(template)

aclinit -> acle ( , acle)*
acle -> [ ID(name) , priv , ID(token) ]

priv -> ALL | priv2
priv2 -> KEY | LOCK | GRANT . priv2
        | REVOKE . priv2
--------------------------------------
OBJECT R3(Patient,Allergy)
        : METADATA = empty
        : ACL := {[read,lock,Pat],
                  [read,lock,Phys],
                  [write,lock,Phys]}
        { METHOD Tuple read()
          METHOD Void write(Tuple t))}
```

Figure 4. Specification language and example.

The language employs a simple object model to facilitate the expression of abstract structural descriptions of information systems. The model forms a hierarchical system of objects that can be used to represent information protection units and system boundaries. For instance, an object can represent a table or tuple in a relational database, or it can represent a host or subnet on an enterprise network. The metadata facility in the model can be used to capture class and interface behavior in object-oriented databases. Methods, generalized for object-oriented systems, can be special-

ized to `read` and `write` as accessors for relational data.

Figure 4 also shows a low-level specification for the $R_3(Patient, Allergy)$ table in Figure 1. The specification names the protected object, describes initial metadata and access control configurations, and identifies the set of legal actions on the object. Note that the first element in each `ACL` triple refers to an action on the table. The second element confers a privilege to principals possessing tokens matching those held as the third element.

2.4. Mappings

Sophisticated authorization policies can be mapped into the primitive access control framework. This subsection illustrates the mapping process for DAC, RBAC, TBAC and MAC schemes in by expanding the example in Figure 1.

Discretionary access control (DAC) bases authorizations on subject identity. DAC is mapped to the ticket-based authorization model by associating tokens with subject identities.

Figure 5 illustrates enterprise structure and permission in the poison control center (POISON CTL in Figure 1). In it, `Al`, `Sue`, `Jim` and `Cy` are registered as physicians, while `Jan`, `Jim` and `Cy` have administrative privileges. Registered physicians have access to poison control data. For instance, `Al` (because he is registered as a physician) can issue a query on $R(Toxin, Symptom, Antidote)$. Queries from `Al` carry a ticket with the key that identifies him. $R(Toxin, Symptom, Antidote)$ must have a lock matching `Al`'s key.

This example also demonstrates a concept of ownership. `Cy` owns $R(Toxin, Symptom, Antidote)$; its ACL indicates that `Cy` can grant subject access (via `G.LOCK`) to the table. Thus, physicians must register with `Cy` for access to the toxin data. Moreover, `Cy` can write and grant write permission to the table.

Figure 5 shows a high-level specification expressing DAC configurations. The specification maps to the underlying authorization model and associates each protection unit (table) with a list of possible actions. Actions are associated with a set of subjects permitted to execute them.

The ability of a subject to assume multiple roles makes role-based access control (RBAC) an attractive scheme in many enterprises. Roles may be defined by groups of transactions and/or method invocations in the ticket-based access control scheme. Subjects typically assume a role at login, endowing method groups with "role" tickets. Corresponding locks must be distributed to the appropriate resources.

In the example enterprise, `Joe` plays the roles of accountant and patient. As an accountant, `Joe` can bill patients; as a patient `Joe` can look at his medical record (part of which lies in $R_4(Patient, Allergy)$). These actions are modeled in the object system by methods, `bill()` and `look()`, respectively. Invocations of `bill()`, associated with the accountant role, are endowed with an `Acct` key. Invocations of `look()`, associated with the patient role, are given a `Pat` key.

Figure 6 presents the enterprise with its RBAC authorization scheme. It also shows a high-level specification that maps RBAC configurations to the primitive authorization model. As with the DAC example, each protection unit (table) is associated with a list of permissible actions. Note that each action is associated with a role set.

Task-Based Access Control (TBAC) is motivated by the need for temporary trust while executing transactions. It regards individual transactions as trust units. Individual methods, procedures and functions are the natural manifestations of transactions in computer systems.

Figure 7 illustrates the M.A.S.H. enterprise employing a task-based access control model. Medics may only execute particular functions to complete specific tasks. For example, `Jane` can perform an examination – a process that accesses medical records, but only for the purpose of diagnosis. Furthermore, `Jane` can only access the stockroom database $R(Med, Qty)$ when treating patients.

A specification of the TBAC scheme adopted by the M.A.S.H. enterprise is shown in Figure 7. It associates abstract tasks with methods. However, trust in TBAC can be more fleeting than in RBAC; the same task that was authorized at one point in time might be unauthorized at another. To help implement such policies, the ticket-based authorization model facilitates temporary trust with dynamic distribution and revocation of keys and locks.

Mandatory Access Control (MAC) requires subjects and objects to be tagged with security clearance or classification levels defined by pairs of labels: a security level and a category, e.g., `(Top_Secret, Medical)`. MAC models respect the Simple Property (no read up) and the *-Property (no write down/declassification).

The ticket-based access control scheme implements MAC by mapping tickets to MAC classification labels. Classes, instances and principals carry keys and locks associated with MLS classifications. Classification domains (comprising rank/compartment pairs) form a partial order. The ordering on levels is satisfied by embedding (within each message) all the tickets for all the levels dominated by a subject. The Simple and *-Properties are preserved by enforcing the use of accessor methods for all read and write operations.

The object-oriented database at headquarters that enforces a MAC policy is shown in Figure 8. Note that `Region` and `Toxin` are Secret classes (the compartment concept is omitted to simplify the presentation); `Sector_12` and `Sarin` are respective instances of these classes. Both classes and instances are regarded as objects in the underlying model and utilize a common protection mechanism

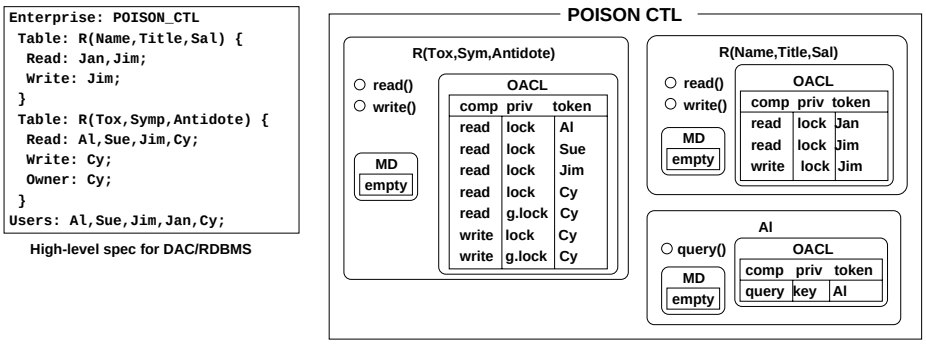

Figure 5. Poison Control – DAC.

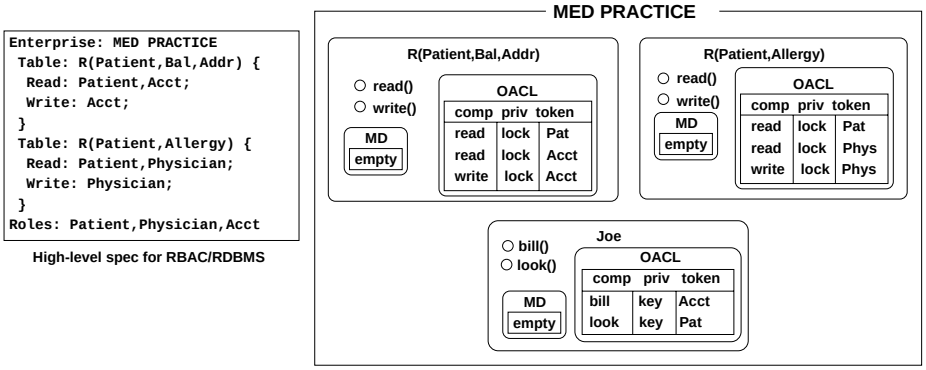

Figure 6. Healthcare example – RBAC.

(ACLs). The principal difference between classes and instances is that the metadata of an instance is empty since it cannot be used to create instance objects.

The protection scheme associates keys and locks with tokens mapping to classifications, e.g., `S` and `TS`. Other tokens, such as `gter` and `atox`, are used to enable private communication between method proxies in instances and method bodies in classes. Figure 8 also shows part of a high-level specification for the enterprise that expresses class, instance, method and slot protection policies.

3. Mediation Infrastructure

This section describes a middleware architecture for security policy mediation in federated information enclaves. It begins by assessing existing technology for software interoperability.

3.1. Interoperability

Achieving interoperability of heterogeneous distributed databases and applications presents many challenges. Identifying a suitable connectivity solution for an information enclave requires handling heterogeneity issues in operating systems and development platforms, and structural disparities in databases. Moreover, interoperation entails deriving a consistent global information schema and access policy from federated database systems. Our infrastructure employs JDBC, CORBA, and mediators –technologies that have surfaced to address issues in database connectivity and application integration.

JDBC attempts to accommodate database heterogeneity by delivering a common API that provides access to different relational databases [10]. JDBC drivers exist for most commercial relational database systems, including Oracle, Sybase and Informix. A standard data access interface, JDBC consists of a set of Java classes and interfaces for executing SQL statements. Applications using JDBC are written in Java, and are therefore platform independent themselves. The principal benefit of JBDC is that one does not have to write multiple applications to achieve heterogeneous database access and cross-platform application deployment. However, JDBC does not directly support security policy coordination between heterogeneous distributed databases.

The Common Object Request Broker Architecture (CORBA) is a general purpose architecture for application integration in distributed computing environments [15]. It supports interoperability via the Interface Definition Language (IDL) that allows language-neutral object interface specifications. CORBA's Object Request Broker (ORB) is a

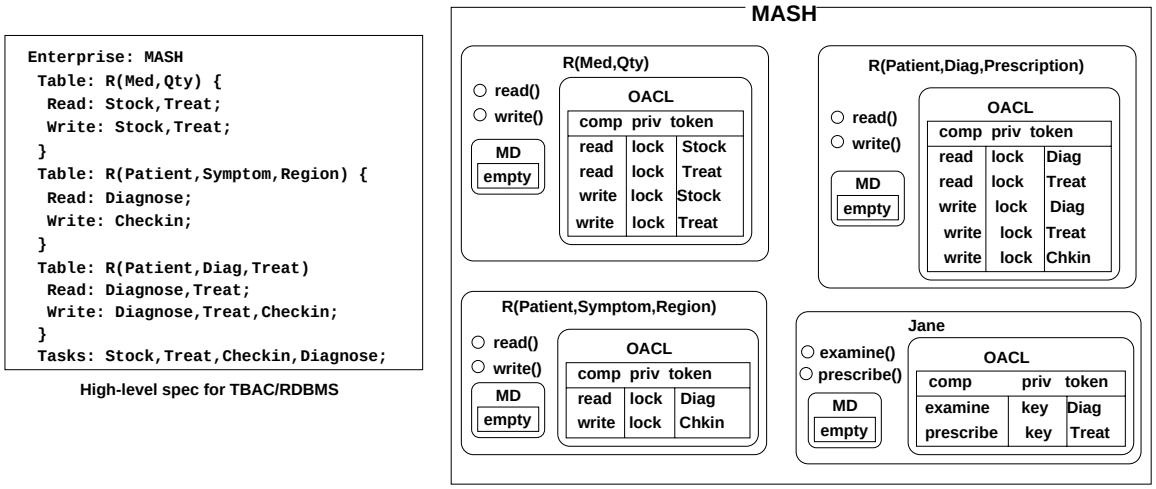

Figure 7. M.A.S.H. – TBAC.

transparent infrastructure that facilitates distributed communication across heterogeneous systems. CORBA includes a set of standard services for (among other things) naming, trading, transactions, persistence and security.

The CORBA Security Service (CORBASEC), based on the Distributed Computing Environment's (DCE's) security architecture, provides access control, authentication and nonrepudiation for distributed object systems [16]. CORBASEC adopts a flexible engagement model that permits security mechanisms to operate transparently or to be called directly by clients and servers. Authenticated principals are given credentials containing their security attributes that indicate the rights they own. CORBASEC promotes a number of secure interoperability schemes, differentiated by various security attribute/credential delegation options.

While CORBA facilitates application-level integration, it does not address higher-level issues. It offers a middleware solution for remote method invocation of heterogeneous distributed objects, but stops short of providing an infrastructure for policy mediation. In particular, CORBA (and CORBASEC) does not have mechanisms for reconciling disparate security policies and models in heterogeneous databases.

Mediators [22] facilitate a three-tiered architecture for enterprise computing. The bottom tier houses information resources and the top tier comprises high-level applications; the middle tier contains mediators that marshal feedback between them. Mediators can serve many purposes, including generating consistent global views from heterogeneous information resources, optimizing performance, analyzing and summarizing information, and authorizing access to information resources.

Security mediators provide managed gateways (sometimes implying human interaction) for database queries and responses. Security mediators can accept query requests from applications, pre-process requests, deliver queries to databases, receive and process results, and record activity in logfiles.

Mediators can also be used to reconcile security policy disparities in federated information enclaves. This approach employs CORBA and JDBC as foundations for integrating applications and heterogeneous distributed databases in open environments. Application subjects hold a partially implicit and potentially heterogeneous collection of rights to various information resources. Mediators in our architecture would determine these rights according to the prevailing "global" authorization policy synthesized from local policies.

3.2. Software Architecture

Security policy coordination of federated information enclaves is achieved within a special architecture that integrates CORBA, JDBC and mediators. JDBC provides a standard API for accessing heterogeneous databases, while CORBA enables cross-platform application-level integration. These technologies can be used in concert to provide a common interface for security policy mediators. Figures 9 and 10 illustrate the software architecture for security policy coordination in heterogeneous information systems.

Each enterprise manages its own policy mediator, which rests on CORBA and JDBC layers. Each mediator contains a current model of its database and prevailing security policy (Figure 10). This is held as a mapping to the primitive authorization model described in Section 2. As seen in Figure 9, mediators are also deployed at principals' sites. These mediators are responsible for fragmenting queries and disseminating them along with appropriate credentials to mediated information systems.

Mediators contain separate coordination policies for

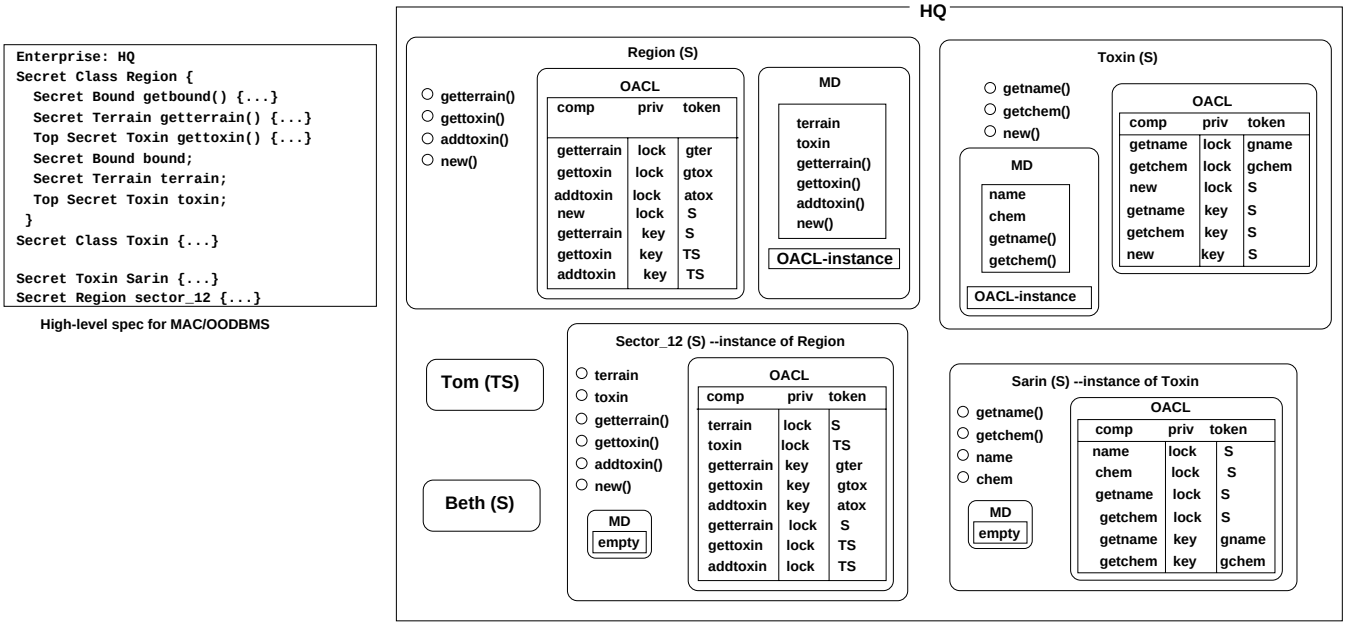

Figure 8. Headquarters – MAC.

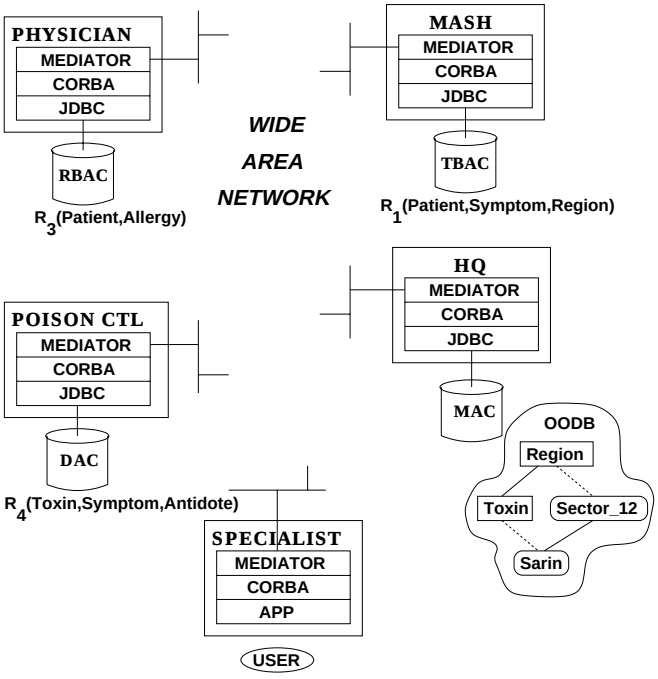

Figure 9. Software architecture.

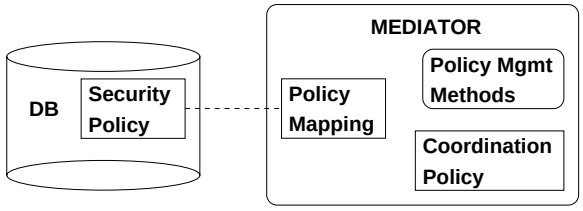

Figure 10. Policy mediator.

managing access by foreign principals. Foreign principals send credentials (called tickets in our authorization model) along with access requests. Mediators intercept access requests along with foreign principal credentials to enforce their specific coordination policies. Coordinating policies can take many forms, including:

- Mapping foreign principals to local principals
- Assigning local proxies to act as trusted delegates of foreign principals
- Requesting vouchers from trusted sources for foreign principals
- Mandating joint authorization with local principals

In the example in Figure 1, the specialist accesses heterogeneous databases, sending her credentials – say, a token pair $(Specialist, Jill)$ – to each intervening mediator. (We assume the credentials offered by the principal's mediator are authentic.) The coordination policy for the enterprise

may dictate mapping a $Specialist$ token from the foreign principal into a $Physician$ role. The poison control center may not have a $Jill$ registered and so the mediator might register her upon obtaining a voucher from a trusted source. The coordination policy at the M.A.S.H. unit could mandate the creation of a local proxy with limited authority to execute specific tasks for the foreign principal. Finally, joint authorization of the specialist and a local principal with suitable clearance could be mandatory at headquarters for accessing sensitive records.

The security policy mediation architecture extends the security perimeter of an enterprise by permitting controlled access to databases by foreign principals. The benefit of this architecture is increased functionality without the sacrifice of control. The challenge is to understand the resulting policy for the larger system.

3.3. Security Analysis

Guaranteeing the enforcement and consistency of global security policies is difficult for information enclaves. The use of a primitive authorization model as a common substrate for heterogeneous policies provides an opportunity for their formal analysis. Our architecture maps disparate security policy models to a common framework to permit the verification of global consistency and ensure enforcement.

Policy consistency is of primary concern in federated systems where security management is a decentralized process with no common authority. Enterprises may compose conflicting global policies from internally consistent ones. Adding an enterprise (and its security policy) to a federated system of enterprises can introduce conflicts in the global policy. Similarly, global policy consistency can be jeopardized when policies are adjusted. For example, a new or adjusted enterprise policy might allow principals to foil joint-authority requirements by enabling them to pose as dual identities (i.e. by allowing one principal to present two unique sets of credentials).

New or modified policies can be "statically checked" for global consistency when a policy compiler maps the high-level policy specification into the ticket-based access control model. The formalisms in the underlying model (see Figure 3) coupled with denotational semantics for policy specification languages can make such compile-time checks possible. This enables security managers to understand the broader implications of establishing new policies or adjusting existing ones.

Policy enforcement occurs dynamically, i.e., at run-time. Principal mediators bundle credentials with query fragments to distribute to remote information systems. Mediators at the information systems apply their coordination policy to incoming requests based on the bundled credentials. The combination of compile-time and run-time security policy analysis and management facilitates global policy consistency and enforcement.

4. Comparison with Other Work

A comparison with related work falls across several lines: authorization models, forms and levels of heterogeneity, and the overall approach.

Authorization models differ in protection granularity for objects and the basis of privilege for subjects. The growing popularity of RBAC [18] and TBAC [21] demonstrates the need for distinct models in different settings. The advent of semantically-diverse object-oriented databases has led to the development of even more models [2,4,12,17,20]. The integration of such variegated information systems mandates the coordination of equally diverse security policies.

There are many difficulties in coordinating security policies in federated systems, where enterprises must collaborate and yet maintain some autonomy [7]. Managing heterogeneity at the policy level is particularly challenging. De Capitini di Vimercati and Samarati [7] have proposed an authorization model that addresses these problems for tightly coupled federations. Their solution guarantees authorization autonomy by supporting decentralized access control between local sites but relying on a central authority for managing federated schema and objects. Our approach differs in that it focuses on loosely coupled federations where no central authority for federation management is possible.

The Argos system unifies heterogeneous access control models in an open distributed environment [13] by permitting the configuration of various identity-based authorization policies. Identity-based authorization models are prevalent, but our ticket-based access control scheme can be used to model radically different policy models. Argos also introduced the notion of domains for subject behavior and object protection rights. Domains generate classes of protection rights and behavior, materializing basic access types (e.g., read and write) from complex object systems. Argos remains one of the largest efforts to achieve a general multi-policy authorization service model and implementation for object-oriented databases.

Flexible schemes that support multipolicy access control are becoming increasingly relevant [1,3]. The architecture described in [3] employs flexible mechanisms and mediators [22] for tunable access control policies. Candan *et al.* [5] used the Heterogeneous Reasoning and Mediator System (HERMES) as a foundation for the secure mediation of databases. Their work promotes a principle of cautious cooperation to enforce local and global security policies while providing widely integrated services. Dawson *et al.* [6] describe a mediator architecture for reconciling heterogeneous data and MLS lattices in federated databases. However, our approach is unique in that policy heterogeneity is addressed

at the model level, i.e., the integration of RBAC, MAC, and DAC policies.

The metapolicy concept –a policy about policies– was introduced in [11]. Metapolicies were applied to policy negotiation with the design of a formal "multipolicy machine" in [1]. Kuhnhauser and von Kopp Ostrowski [14] have engaged metapolicies to construct a formal framework that supports multiple access control policies for loosely coupled federations. The focus of their effort is to provide support for application-specific policy development, coexistence and re-use in open environments. Metapolicies in this framework are realized through the construction of cooperation and conflict matrices – policies themselves are the objects in these matrices. The implementation [14] uses "custodians," similar to the mediators used in this work.

5. Conclusions

Security managers struggle with policy coordination between enterprises in information enclaves implementing disparate data and security models. Mediators can be used in conjunction with other integration technologies to achieve a standard architecture for security policy coordination designs and analyses. The primitive authorization model embraced by the security policy mediation architecture presented in this paper provides a common foundation for policy coordination and facilitates static and dynamic analyses of security.

References

[1] D. Bell. Modeling the multipolicy machine. *Proceedings of the New Security Paradigms Workshop*, 2–9, 1994.

[2] E. Bertino, S. Jajodia and P. Samarati. Access control in object-oriented database systems: Some approaches and issues, in *Advanced Database Concepts and Research Issues (LNCS 759)* (eds. N. Adam and B. Bhargava), Springer-Verlag, Amsterdam, 17–44, 1993.

[3] E. Bertino, S. Jajodia and P. Samarati. Supporting multiple access control policies in database systems. *Proceedings of the IEEE Symposium on Research in Security and Privacy*, 94–109, 1996.

[4] H. H. Bruggemann. Rights in an object-oriented environment. *Database Security, V: Status and Prospects* (eds. C. Landwehr and S. Jajodia), Elsevier, Amsterdam, 99–115, 1992.

[5] K. S. Candan, S. Jajodia and V. S. Subrahmanian, Secure mediated databases. *Proceedings 12th International Conference on Data Engineering*, 28–37, 1996.

[6] S. Dawson, S. Qian and P. Samarati, Secure interoperation of heterogeneous systems: A mediator-based approach. *Proceedings of the 14th IFIP TC-11 International Conference on Information Security*, 1998.

[7] S. De Capitani di Vimercati and P. Samarati. Authorization specification and enforcement in federated database systems. *Journal of Computer Security*, **5(2)**, 155–188, 1997.

[8] V. Gilgor, J. Huskamp, S. Welke, C. Linn, and W. Mayfield. Traditional capability-based systems: An analysis of their ability to meet the trusted computer security evaluation criteria, Institute for Defense Analyses, IDA Paper P-1935, 1987.

[9] J. Hale, J. Threet, and S. Shenoi. A framework for high assurance security of distributed objects. *Database Security, X: Status and Prospects* (eds. P. Samarati and R. Sandhu), Chapman and Hall, London, 99–115, 1997.

[10] G. Hamilton, R. Cattell and M. Fisher. *JDBC Database Access With Java: A Tutorial and Annotated Reference*. Addison-Wesley, New York, 1997.

[11] H. H. Hosmer. The multipolicy paradigm for trusted systems. *Proceedings of the New Security Paradigms Workshop*, 19–32, 1993.

[12] S. Jajodia and B. Kogan. Integrating an object-oriented data model with multilevel security. *Proceedings of the IEEE Symposium on Research in Security and Privacy*, 76–85, 1990.

[13] D. Jonscher and K. R. Dittrich. Argos – A configurable access control system for interoperable environments, in *Database Security, IX: Status and Prospects* (eds. D. Spooner *et al.*), Chapman and Hall, London, 43–60, 1995.

[14] W. E. Kuhnhauser and M. von Kopp Ostrowski. A framework to support multiple security policies. *Proceedings of the 7th Annual Canadian Computer Security Symposium*, 1995.

[15] T. J. Mowbray and R. Zahavi. *The Essential CORBA: Systems Integration Using Distributed Objects*. John Wiley, New York, 1995.

[16] W. Rosenberry, D. Kenney and G. Fisher. *Understanding DCE*. O'Reilly and Associates, Inc., Sebastopal, California, 1993.

[17] A. Rosenthal, J. Williams, W. Herndon and B. Thuraisingham. A fine grained access control model for object-oriented DBMSs, in *Database Security, VIII: Status and Prospects* (eds. J. Biskup *et al.*), Elsevier, Amsterdam, 319–334, 1994.

[18] R. Sandhu. Access control: The neglected frontier. *Proceedings of the First Australian Conference on Information Security and Privacy*, 1996.

[19] R. Sandhu, E. Coyne, H. Feinstein and C. Youman. Role-based access control models. *IEEE Computer*, **29(2)**, 38–47, 1996.

[20] R. K. Thomas and R. Sandhu. Discretionary access control in object-oriented databases: Issues and research directions. *Proceedings of the Sixteenth National Computer Security Conference*, 63–74, 1993.

[21] R. K. Thomas and R. Sandhu. Task-based authorization controls (TBAC): A family of models for active and enterprise-oriented authorization Management, in *Database Security, XI: Status and Prospects* (eds. T.Y. Lin and S. Qian), Chapman and Hall, London, 166–181, 1997.

[22] G. Wiederhold. Mediators in the architecture of future information systems: A new approach. *IEEE Computer*, **25(3)**, 38–49, 1992.