Administration policies in a multipolicy authorization system

Elisa Bertino Elena Ferrari
Dipartimento di Scienze dell'Informazione
Università di Milano
Via Comelico 39/41, 20135 Milano,Italy
e-mail: {bertino,ferrarie}@dsi.unimi.it

Abstract

This paper describes the administration policies supported by the MultiPolicy Authorization System (MPAS). Several administration policies are supported including centralized administration, decentralized administration with delegation and transfer, and joint administration. In the paper we first present an overview of the MPAS architecture. We then discuss the various administration policies and formalize some aspects of the proposed administration model.

Keywords

Multipolicy Access Control Mechanisms, Administration Policies

1 INTRODUCTION

The introduction of an access control system within any organization entails two main tasks. The first task is the identification and specification of suitable *access control policies*. An access control policy establishes for each user (or group of users, or functional role within the organization) the actions the user can perform on which object (or set of objects) within the system under which circumstances. An example of access control policy (Bertino et al. 1996b) is that "all programmers can modify the project files every working day except Friday afternoon". The second task is developing a suitable *access control mechanism* implementing the stated policies. Because of the richness of possible access control policies, this second task is quite difficult. Current access control mechanisms are tailored to few, specific policies and are unable to satisfactorily support access control requirements of several applications. In most cases, either the organization is forced to adopt the specific policy built-in into the access control mechanism at hand, or access control policies must be implemented as application programs. Both situations are clearly un-

Database Security XI T.Y. Lin and Shelly Qian (Eds)

acceptable. Many advanced applications, such as workflow applications, and computer-supported cooperative work, have articulate and rich access control requirements. Therefore, those applications cannot be adequately supported by a single-policy access control mechanism. Implementing access control policies as application programs, on the other hand, makes it very difficult to verify and modify the access control policies and to provide any assurance that these policies are actually enforced.

A possible approach is to develop flexible access control mechanisms, able to support different access control policies for possibly different objects within the system. We refer to such a system as a *multipolicy access control mechanism*. Consider the classical *closed* and *open* policies. It is easy to encounter many situations where some objects within a system must be governed by the open policy, whereas other objects within the same system need a stricter control, thus requiring a closed policy.

Several proposals in the areas of database systems and operating systems have addressed issues related to multipolicy access control mechanisms. Proposals in the database area include the flexible authorization model proposed in (Bertino et al. 1996c), and the Chassis system (Jonscher et al. 1994) specifically addressing access control for federated heterogeneous database systems. Proposals in the operating systems area include Trusted Mach (Branstad et al. 1989) and DTOS (Fine et al. 1993).

All the above proposals, however, do not provide sophisticated authorization administration policies. Authorization administration refers to the function of granting and revoking authorizations. It is the function by which authorizations are entered (removed) into (from) the access control mechanism. In most of the above approaches, the administration policies are either based on the centralized approach, or on the ownership approach possibly complemented with the administration delegation approach. Moreover, no multiple administrative policies within the same system are supported.

Two different administration policies are briefly discussed by Fernandez, Gudes and Song (Fernandez et al. 1994) in the framework of an access control mechanism for object-oriented database systems. The first policy is based on the ownership approach. The users own the data and administer their data. Under the second policy, some special users (administrators) control the access to data. However, their access control model supports only the second policy. By contrast, our model supports both those policies, and many others, so that users may choose the policy which best suits their application requirements.

Finally, we would like to mention the brief preliminary discussion reported in (Varadharajan et al. 1996) addressing the use of delegation and joint actions in authorization systems. These mechanisms are able to support sophisticated authorization schemes, particularly useful when dealing with complex information systems and distributed systems. Even though the authors do not address authorization administration, we believe that many issues pointed out in the discussion are relevant to our approach.

In this paper we present the authorization administration facilities provided as part of the MultiPolicy Authorization System (MPAS) being currently developed at the University of Milano. MPAS supports both the specification and implementation of multiple authorization and administration policies by, at the same time, clearly separating specification from implementation. The system supports a large variety of administration policies from centralized administration, either DBA-* or owner-based, to joint-based administration, by which several users are jointly responsible for authorization administration. MPAS currently supports only discretionary access control policies. The reason is that the applications using discretionary policies are those that usually require high flexibility. We plan, however, to explore other types of access control policies, such as the chinese wall policy.

The remainder of this paper is organized as follows. Section 2 briefly discusses the MPAS architecture. Section 3 presents the various administrative policies supported by MPAS. Section 4 presents a formalization of some aspects of our authorization administration model. Finally, Section 5 concludes the paper.

2 ARCHITECTURE OF MPAS

The architecture of MPAS is illustrated in Figure 1. The system consists of two main environments: *the policy specification environment* and the *run-time environment*. In discussing the architecture, we will cast the discussion in terms of database systems, by assuming that the items to be protected are data objects, such as relations in a relational database system, or objects in an object database system. However, we believe that the discussion is also valid in other contexts.

2.1 Policy specification environment

The policy specification environment supports all functions concerning policy specification for object authorization and authorization administration.

Example 1 *Consider a table* `Public_info` *containing information available to all employees of a given company. An example of authorization policy is that access to table* `Public_info` *must be governed by the open policy (Bertino et al. 1996c), whereas an example of administration policy is that authorizations on* `Public_info` *can only be granted by a DBA (in practice, this means that only the DBA can issue access denials).*

*DBA stands for database administrator.

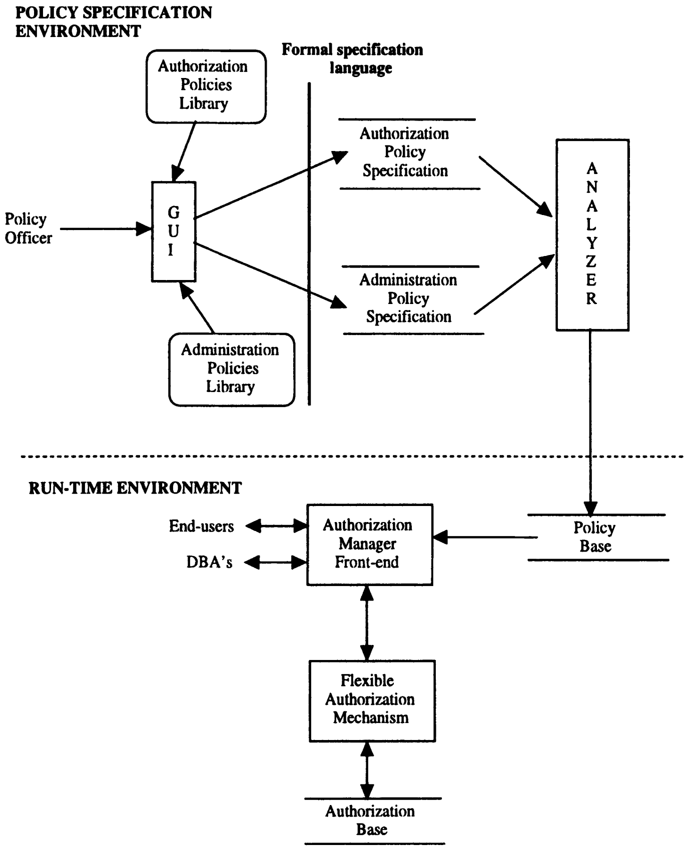
POLICY SPECIFICATION ENVIRONMENT
Authorization Policies Library
Formal specification language
Authorization Policy Specification
Policy Officer
G U I
Administration Policy Specification
Administration Policies Library
A N A L Y Z E R
RUN-TIME ENVIRONMENT
End-users
DBA's
Authorization Manager Front-end
Policy Base
Flexible Authorization Mechanism
Authorization Base

Figure 1 Architecture of the multipolicy authorization system (MPAS)

A number of predefined authorization policies are supported (denoted in the reference architecture as the authorization policies library), including the following: traditional closed and open policies; the closed policy with negation and the conflict resolution principle based on "denials take precedence"; the closed policy with negation and the conflict resolution principle based on "the most specific authorization takes precedence" (Bertino et al. 1996c). In particular, the last two policies have conflict resolution principles for dealing with conflicting authorizations* granted on the same object to the same subject. It is also possible for the policy officer to specify custom-made policies by using a special purpose language, based on rules (Bertino et al. 1997).

A number of predefined administration policies are supported (denoted in the reference architecture as administration policies library) that will be discussed in the next section. All the specifications are checked for consistency

*Two authorizations conflict if one is a positive authorization whereas the other is a negative one.

and correctness by the analyzer (Bertino et al. 1997). The results of the analysis is a policy base encoding the administration and authorization policies for each data object.

An important aspect of our system is that *parametric policies* can be also specified. Such a policy is parametric with respect to the objects to which it applies. It allows to specify policies such as "All tables created by the role Secretary must be authorized under the open policy" or "All documents pertaining to project MPAS must be authorized under the closed policy". Parametric policies basically contain conditions stated against *authorization attributes* of data objects. Authorization attributes contain information about the administered objects that is relevant from security point of view. They compose the *security profile* of the data object. Each subject also has a similar security profile. Parametric policies are very important for systems with large numbers of subjects and objects to be protected. In such systems, specifying policies for each object would not be viable.

2.2 Run-time environment

The run-time environment consists of a flexible authorization mechanism which is the core of MPAS, and of a front-end. The flexible authorization mechanism implements a generalized authorization model able to support a large number of policies (Bertino et al 1996c). The front-end has the task of verifying the user's and DBA's requests with respect to the policies stated by the policy officer (stored in the policy base) and mapping them onto the generalized authorization model.

Example 2 *Consider table* `Public_info` *of Example 1 for which an open authorization policy and a DBA administration policy have been specified. Suppose that a DBA enters a positive authorization for a certain user on this table. Such authorization is rejected by the front-end because only negative authorizations can be specified for this table, according to the open policy.*

3 ADMINISTRATION POLICIES

In this section we discuss the predefined administration policies supported by our system. Note, however, that additional policies can be specified by using the rule-based language supported by the policy specification environment.

Before discussing the policies it is important to recall that authorization administration consists of issuing grant and revoke requests to the authorization system. Those requests, that must be consistent with the administration policies, enter or remove authorizations from the authorization base (cfr. Figure 1). They are issued by users that must be, in turn, properly authorized. We will also refer to the notion of *object creator*; it is the user who has created the object or on behalf of whom the object has been created (for example, within an application program run by the user).*

The following administration policies are supported by MPAS:

- **DBA administration**: under this policy, only the DBA can issue grant and revoke requests on a given object.* This policy is highly centralized (even though different DBAs can manage different parts of the database) and it is seldom used in current DBMSs, but in the simplest systems.
- **Object "curator" administration**: under this policy, a user, not necessarily the creator of the object, is named administrator of the object. Under such policy, even the object creator must be explicitly authorized to access the object.
- **Object owner administration**: under this policy, which is commonly adopted by DBMSs and operating systems, the creator of the object is the owner of the object and is the only one authorized to administer the object.

The second and third administration policies above can be further combined with *administration delegation* and *administration transfer*. Even though delegation and transfer could be applied to DBA administration, we do not have included this possibility because it is not very significant.*

By administration delegation we mean that the administrator of an object (either the owner or the curator) can delegate other users administration functions on the object. Delegation can be specified for selected access modes, for example only for read operations. In most cases, delegation of administration to another user implies also granting this user the privilege of accessing the object according to the same access mode specified in the delegation. Most current DBMSs support the administration policy based on the owner administration with delegation. Note that under the delegation approach, the initial administrator of the object does not lose his/her privilege to administer the object. Therefore, authorizations on the same object can be granted by different administrators. With respect to revoke operations, we take the approach, common to most systems, that only the *grantor* of an authorization can revoke it.

Administration transfer, like delegation, has the effect of giving another user the right to administer a given object. However, the original administrator loses his administration authorization, whereas under delegation the original administrator keeps his administration right. When transfer is used with owner administration it has the semantics of owner transfer; we call it *ownership transfer*. Therefore, the owner of the object is actually replaced by the user who has been delegated the administration. Transfer is very relevant in workflow applications, where objects (such as documents, and office

*In some systems, a DBA can create an object on behalf of some users.

*Note that DBAs have also the authorization to give users the privilege to connect to the DBMS, and can also read and write all data objects created by other users.

*It can be simply implemented in terms of the other two policies.

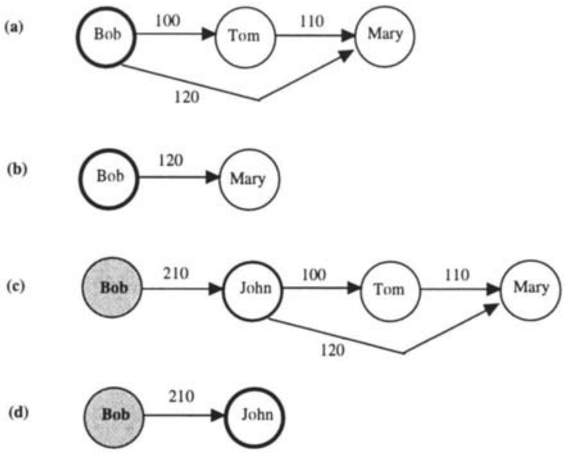

Figure 2 (a) An example of administration graph; (b) the graph after Bob revokes the delegation to Tom; (c) the graph after Bob transfers the ownership to John with grantor transfer; (d) the graph after Bob transfers the ownership to John with recursive revoke

forms) migrate among different departments (or organizational units) within the same organization. Often, because of those transfers, objects may enter different administration domains and it is, thus, important that the privileges of administering the objects be properly modified. Also, security requirements may dictate owner transfer. Consider a document which is initialized by a secretary and later on transferred to her boss who enters confidential information. In such a case, it is important that the initial owner, i.e., the secretary, is no longer authorized to administer the object.

When dealing with transfer an important question concerns the authorizations granted by the former administrator (such problem does not arise in delegation because the former administrator retains his right to administer the object). The following two approaches are supported by MPAS for dealing with authorizations granted by the former administrator:

1. *Recursive revoke*: all authorizations granted by the former administrator are recursively revoked.
2. *Grantor transfer*: all authorizations granted by the former administrator are kept; however, the new administrator replaces the old one as grantor of the authorizations (and is thus able to revoke them). Note that the grantor transfer is not recursive. Therefore, if the older administrator has delegated other users for administration, those grants are left in place. Only, the new administrator becomes their grantor.

We provide a further transfer option. Transfer can be *with acceptance* or *without acceptance*. By acceptance we mean that the user to whom the administration (or ownership) is transferred must explicitly accept the administration responsibility. Transfer without acceptance means that such explicit acceptance is not required. Explicit acceptance is important especially when the object ownership is transferred. Since the object owner is ultimately responsible for the object and may be liable for the content of the object, the explicit acceptance avoids that a user be transferred the ownership of an object, without even knowing about it.

Our model also supports *joint administration* of data objects. Joint administration means that several users are jointly responsible for administering the object. Joint administration can be used in both the object "curator" administration and object owner administration policy. Joint administration is particularly useful in computer-supported cooperative work (CSCW) applications where typically users cooperate to produce a complex data object (such as a document, a book, a piece of software, a VLSI circuit). In such applications, each user in the work group is responsible for producing a component of the complex object; therefore, no single user is the owner of the entire complex object. Authorization for a user to access a data object, administered under the joint administration policy, requires that all the administrators of the object issue a grant request. As a further option, MPAS supports joint administration with *quorum*. Under such option, an authorization is granted to a user, if a number of administrators equal to the quorum have issued the proper grant request.

It is important to note that joint administration can be combined with the various options illustrated before. For example, one of the administrators of a data object may transfer or delegate its administration right to another user. A large spectrum of administration policies is thus obtained.

In what follows we give some examples of the various administration policies supported by our model.

In the examples, we represent a sequence of delegation operations as a graph, called **administration graph**. In such a graph, a boldface node denotes the owner of an object, whereas a non-boldface node denotes a user who has been delegated the administration right on the object. There is an arc from node i to node j if the user represented by node i has delegated the administration right to the user represented by node j; each arc is furthermore labeled with the delegation time. An example of administration graph is illustrated in Figure 2(a).

Sequences of grant requests for a given access mode $\mathtt{m}$ on a given object $\mathtt{o}$ are represented by means of **authorization graphs**. Nodes represent users. There is an arc from node i to node j if the user represented by node i has granted an authorization for $\mathtt{m}$ on $\mathtt{o}$ to the user represented by node j. The arc is labeled with the time of the grant and with the granted access mode. Joint authorizations are represented by a special place-holder denoted by a vertical boldface bar. An example of authorization graph is shown in Figure 3(a).

The following example illustrates some of the administration options illustrated above for the case of single (i.e. non-joint) administration.

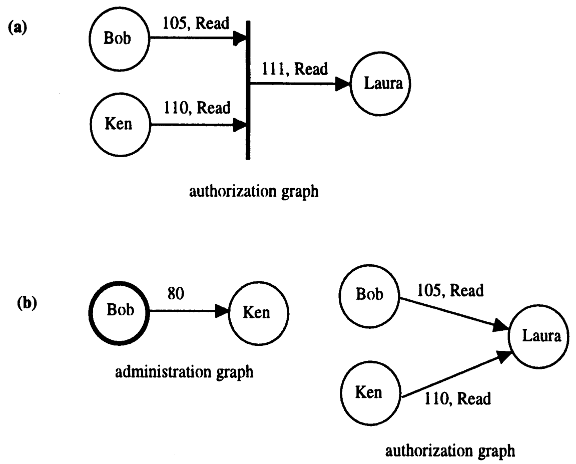

Figure 3 (a) The authorization graph for Example 4; (b) The administration and authorization graphs for Example 5

Example 3 *Consider a table* T *which is administered under the owner administration policy with both delegation and transfer options. Suppose that* Bob *is the owner of the table. Under the above options,* Bob *can delegate other users the right to administer the object. Suppose that he grants such right to* Tom *at time* 100[*] *and to* Mary *at time* 120. *Suppose, moreover, that* Tom *in turn delegates* Mary *the administration right at time* 110. *Figure 2(a) illustrates the corresponding administration graph. Consider now the following cases:* 1) Bob *revokes the administration right from* Tom. *As a consequence,* Mary *loses the right received from* Tom. *However, she keeps the right received directly from* Bob. *Figure 2(b) shows the resulting graph.* 2) Bob *transfers at time* 210 *the ownership to* John. *Suppose that the transfer policy has been specified with the grantor transfer option. As illustrated by the graph in Figure 2(c), both* Tom *and* Mary *keep their administration rights. The graph includes a third type of node, called* shadow node, *which is used to keep track of previous object owners.* 3) Bob *transfers at time* 210 *the ownership to* John. *Suppose that the transfer policy has been specified with the recursive revoke option. As illustrated by the graph in Figure 2(d), both* Tom *and* Mary *lose their administration rights. As in the previous case, information about the former owner is kept in the graph.*

The following examples illustrates joint administration.

Example 4 *Consider a table* T, *administered under the joint administration policy. Suppose that* Bob *and* Ken *are the owners of the table. Suppose that the*

[*] We use here a simplified representation of time as an integer number. In most implementations, the system timestamp is used.

following grant operations are performed: 1)Bob *grants* Laura *the* Read *access on* T *at time* 105. 2) Ken *grants* Laura *the* Read *access on* T *at time* 110. *Figure 3(a) shows the resulting authorization graph.*

Consider the following access requests issued by Laura*:* 1)Read *access to* T *at time* 105*; such access is denied because, at time* 105, Laura *does not have the authorization from* Ken. 2) Read *access to* T *at time* 115*; such access is allowed because, at time* 115, Laura *possesses authorizations from both* Bob *and* Ken.

The following example illustrates the difference between joint administration and delegation.

Example 5 *Consider again table* T. *Suppose that it is administered under a non-joint policy and that* Bob *is its owner. Moreover, suppose that* Bob *delegates* Ken *the administration right at time* 80. *Suppose now that* Bob *and* Ken *perform the same grant operations illustrated in Example 4. Figure 3(b) shows the corresponding administration and authorization graphs. Consider again the access requests, listed in Example 4, issued by* Laura*:* 1) Read *access to* T *at time* 105*; such access is allowed, because* Laura *possesses the authorization granted from* Bob. 2) Read *access to* T *at time* 115*; such access is allowed because* Laura *possesses two authorizations, one from* Bob *and another from* Ken.

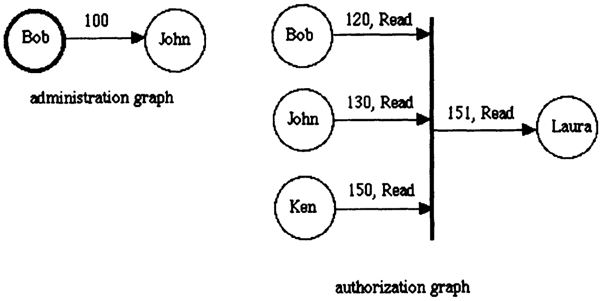

Figure 4 Administration and authorization graphs for Example 6

Delegation and transfer also differ with respect to the semantics of the revoke operations (Bertino et al. 1997).

Several interesting issues are related to the semantics of the delegation option when combined with joint administration. Indeed, one of the administrators of the object may delegate other users the administration of the object.

This means that the delegated user may issue a grant request instead of the original object administrator. This grant request must be combined with the grant requests from all the other administrators (or from users delegated by them) before the authorization can actually be issued. If a quorum option is used, multiple grant requests from an administrator and his delegates amount to a single request with respect to the quorum computation. Such approach avoids that an administrator may reach the quorum by simply delegating other users the administration rights. The following examples illustrates joint administration with delegation.

Example 6 *Consider a table* `T`*, administered under the joint ownership administration policy with quorum option. Suppose that* `Bob`*,* `Ken` *and* `George` *are the owners of the table. Suppose that the quorum is 2. Suppose that the following delegation and grant operations are issued:* 1) `Bob` *delegates* `John` *the administration authorization on* `T` *at time* `100`*.* 2) `Bob` *grants* `Laura` *the* `Read` *access on* `T` *at time* `120`*.* 3) `John` *grants* `Laura` *the* `Read` *access on* `T` *at time* `130`*.* 4) `Ken` *grants* `Laura` *the* `Read` *access on* `T` *at time* `150`*. Figure 4 shows the resulting administration and authorization graphs. Consider the following access requests issued by* `Laura`*:* 1) `Read` *access to* `T` *at time* `130`*; such access is not allowed, because two read authorizations have been issued for* `Laura`*, but they are from an administrator and a delegate of his.* 2) `Read` *access to* `T` *at time* `160`*; such access is allowed because* `Laura` *now possesses two authorizations, granted by two different administrators (or their delegates).*

We refer the reader to (Bertino et al. 1997) for a discussion and for a formal definition of the semantics of delegation when combined with joint administration.

4 FORMAL MODEL

In this section we formalize some aspects of our authorization administration model.

Let O be the set of objects and U the set of users in the system. Let $\mathcal{DBA}$ denotes the set of all users having DBA authorizations. Let $\mathcal{N}$ denotes the set of natural numbers. Moreover, let $\mathcal{PS}$ = {`DBA,object-curator,object-owner, joint-object-curator,joint-object-owner`} denotes the set of administration policy types. An administration policy is defined as follows.

Definition 1 (Administration policy) *An administration policy is a 7-tuple* `[o,pt,delegation_opt,transfer_opt,acceptance_opt,revoke_opt, vote_opt]`*, where* `o` $\in O$*,* `pt` $\in \mathcal{PS}$*,* `delegation_opt` $\in$ {`delegation, no-delegation,nil`}*,* `transfer_opt` $\in$ {`transfer,no-transfer,nil`}*,* `acceptance_opt` $\in$ {`acceptance,no-acceptance,nil`}*,* `revoke_opt` $\in$ {`revoke,grantor-transfer,nil`}*,* `vote_opt` $\in$ {`quorum,totality,nil`}.

In the above definition, a number of components of the 7-tuple defining an administration policy are actually flags indicating specific options. Also, those flags may take a nil value. The nil value simply denotes that the flag is not significant for the specific policy type. Finally, the `vote_opt` component is only significant for joint administration policy (for non-joint administration policies it always takes the nil value). It takes the `totality` value to denote that for an authorization to be granted all the administrators must have issued the proper grant requests; it takes the value `quorum` otherwise.

Example 7 *The policy specifications below illustrate the above definition:*

- *The policy specification:* `[Public_info,DBA,nil,nil,nil,nil,nil]` *states that table* `Public_info` *must be administered by the* `DBA`*. All other components are not significant for this policy and are thus set to* `nil`.
- *The policy specification:* `[T,object-owner,delegation,transfer,no-acceptance,grantor-transfer,nil]` *states that table* `T` *must be administered by the owner. Moreover, both delegation and transfer are allowed on this table. Transfer is without acceptance and all granted administration authorizations are not revoked if the ownership is transferred. This policy is the one exemplified in case 2 of Example 3. Note that the* `vote_opt` *is* `nil` *since the policy type is not a joint one.*

The policy administration base is a set of administration policies, denoted as $\mathcal{PAB}$.

Information specified by the administration policies are complemented with information about: *(i)* which users are DBA; *(ii)* which administrators have delegated (transferred) to which other users administration authorizations; *(iii)* for each data object, its owner (owners), if the object is administered under the (joint) owner administration policy, or its "curator" ("curators"), if the object is administered under the (joint) "curator" administration policy; *(iv)* for each object, administered under a joint administration policy with quorum, the actual quorum value to use.

The above information is stored as facts into the policy base (see Figure 1).

Definition 2 (Delegation and transfer specification) *A delegation (transfer) specification is a triple* `[grantor,grantee,o]`*, with* `grantor,grantee` $\in U$*, and* `o` $\in O$.

The delegation (transfer) base is a set of delegation (transfer) specifications, denoted as $\mathcal{DL}$ ($\mathcal{TR}$).

Definition 3 (Owner and curator specification) *An owner (curator) specification is a pair* `[o,u]`*, with* `o` $\in O$ *and* `u` $\in U$*;* `u` *is referred to as an initial administrator of* `o`.

Note that when the joint administration policy is used for an object, there are several initial administrators for the object.

The owner (curator) base is a set of owner (curator) specifications, denoted as $\mathcal{OWB}$ ($\mathcal{CB}$).

Definition 4 (Quorum specification) *A quorum specification is pair* [o,n], *with* $\mathrm{o} \in O$, $\mathrm{n} \in \mathcal{N}$.

The quorum base is a set of quorum specifications, denoted as ($\mathcal{QB}$).

The following definition introduces the notion of delegates and states how the delegates of a given administrator are determined from the delegation base. Note that an administrator can have indirect delegates; indeed, an administrator may delegate a user the administration right and this user may, in turn, delegate other users.

Definition 5 (Delegates) *Let* u_i *and* u_j *be users. Let* o *be an object. We say that* u_j *is a delegate of* u_i *for the administration of* o *if predicate* $\mathtt{d}(\mathrm{u}_i,\mathrm{u}_j,\mathrm{o})$, *defined below, is True.*

- $\mathtt{d}(\mathrm{u}_i,\mathrm{u}_j,\mathrm{o})$ = *True, if* $[\mathrm{u}_i,\mathrm{u}_j,\mathrm{o}] \in \mathcal{DL}$;
- $\mathtt{d}(\mathrm{u}_i,\mathrm{u}_j,\mathrm{o})$ = *True, if there exists* $\mathrm{u}_k \in U$ *such that* $\mathtt{d}(\mathrm{u}_i,\mathrm{u}_k,\mathrm{o})$ = *True and* $\mathtt{d}(\mathrm{u}_k,\mathrm{u}_j,\mathrm{o})$ = *True.*

Let o be an object and let u be a user such that u is an initial administrator of o. The set $\mathcal{A}_{\mathrm{u}} = \{\mathrm{u}\} \cup \{u_i \mid u_i \in U \text{ and } \mathtt{d}(\mathrm{u},u_i,\mathrm{o}) = True\}$ denotes a set including u and all the delegates of u.

The following definition states the notion of *independent administrators.* It is the basis of the rule determining when a grant becomes valid in the case of joint administration.

Definition 6 (Independent administrators) *Let* u_i *and* u_j *be users. Let* o *be an object. We say that* u_i *and* u_j *are independent administrators of* o, *if the following conditions are verified:* i) $\exists \mathrm{u}'_i$ *initial administrator of* o *such that* $\mathrm{u}_i \in \mathcal{A}_{\mathrm{u}'_i}$; ii) $\exists \mathrm{u}'_j$ *initial administrator of* o *such that* $\mathrm{u}_j \in \mathcal{A}_{\mathrm{u}'_j}$; iii) $\nexists \mathrm{u}_k$ *initial administrator of* o *such that* $\mathrm{u}_i \in \mathcal{A}_{\mathrm{u}_k}$ *and* $\mathrm{u}_j \in \mathcal{A}_{\mathrm{u}_k}$; u_i *and* u_j *are called non-independent administrators, otherwise.*

The above definition states that two users, who have been delegated by other users to administrate a given object are independent if they have received their administration privilege from users that are, in turn, independent. Note that the initial administrators of the object are always independent.

The following rule formally states when a set of authorizations granted by administrators of an object to a user actually enables the authorization for this user.

Joint administration rule

Let $GU = \{u_1, \ldots, u_n\}$ be the set of users who have granted an authorization for the same privilege on object o to user u. Let v be the vote policy for object o. Let GU^* be the maximal subset of GU such that $\forall u_i, u_j \in GU^*$, u_i and u_j are independent administrators of o.* The granted authorization is enabled if the following conditions are verified: *i)* If v = `totality`: let adm be the number of initial administrators of object o. Then $card(GU^*)$* is greater than or equal to adm; *ii)* If v = quorum: let q be the quorum required for object o. Then $card(GU^*)$ is greater than or equal to q.

Note that the above definition implies that when two users u_1 and u_2 give the same authorization to the same user, these authorizations are both effective to enable the authorization for the user only if u_1 and u_2 have obtained the administration privilege by two independent sources, that is, there does not exist a user which gave the administration privilege to both u_1 and u_2.

5 CONCLUSIONS

In this paper we have presented an overview of the administration policies supported by the MPAS multipolicy authorization system. The system supports a variety of policies that are obtained by providing several options, such as delegation and administration. Joint administration policies are also supported. A notable feature of MPAS is the notion of parametric authorization policies. The system is currently under implementation. A preliminary implementation of a flexible authorization mechanism has been completed. We are extending both our model and architecture to provide a more accurate modeling of roles, along the lines discussed by Sandhu in (Sandhu 1996), and to incorporate temporal authorization features (Bertino et al. 1996a). A final research direction we plan to pursue is related to the use of multipolicy systems in heterogeneous systems (Gong 1994).

REFERENCES

[1] E. Bertino, C. Bettini, E. Ferrari and P. Samarati (1996a) A Temporal Access Control Mechanism for Database Systems. *IEEE Trans. on Knowledge and Data Engineering*, 8(1):67–80.

[2] E. Bertino, C. Bettini, E. Ferrari and P. Samarati (1996b) Supporting Periodic Authorizations and Temporal Reasoning in Database Access Control. *Proc. of the 22nd Int'l Conf. on Very Large Data Bases Conference (VLDB'96)*, Bombay (India), Morgan Kaufman Publishers.

[3] E. Bertino and E. Ferrari (1997) A Multipolicy Access Control System - Authorization Model and Architecture. Technical Report, Dipartimento di Scienze

*This implies that there does not exists $GU' \subseteq GU$ such that $\forall u_i, u_j \in GU'$, u_i and u_j are independent administrators of o, and $GU^* \subset GU'$.

*$card(GU^*)$ denotes the cardinality of set GU^*.

dell'Informazione, Università di Milano.

[4] E.Bertino, S.Jajodia and P. Samarati (1996c) Supporting Multiple Access Control Policies in Database Systems. *Proc. of the IEEE Symposium on Research in Security and Privacy*, Oakland (CA).

[5] M. Branstad, H.Tajalli, F.Mayer and D.Dalva (1989) Access Mediation in a Message Passing Kernel. *Proc. of the IEEE Symposium on Research in Security and Privacy*, Oakland (CA).

[6] E. B. Fernandez, E. Gudes and H. Song (1994) A Model for Evaluation and Administration of Security in Object-Oriented Databases. *IEEE Trans. on Knowledge and Data Engineering*, **6**(2):275–92.

[7] T.Fine and S.E.Minear (1993) Assuring Distributed Trusted Mach. *Proc. of the IEEE Symposium on Research in Security and Privacy*, Oakland (CA).

[8] L.Gong and X.Qian (1994) The Complexity and Composability of Secure Interoperation. *Proc. of the IEEE Symposium on Research in Security and Privacy*, Oakland (CA).

[9] D. Jonscher and K. Dittrich (1994) An Approach for Building Secure Database Federations. *Proc. of the 20th Int'l Conf. on Very Large Data Bases Conference (VLDB'96)*, Santiago (Chile), Morgan Kaufman Publishers.

[10] F. Rabitti, E. Bertino, W. Kim and D. Woelk (1991) A Model of Authorization for Next-Generation Database Systems. *ACM Trans. on Database Systems*, **16**(1):88-131.

[11] R. S.Sandhu (1996) Role Hierarchies and Constraints for Lattice-based Access Controls. *Proc. of 4th European Symposium on Research in Computer Security*, Rome (Italy).

[12] V. Varadharajan and P.Allen (1996) Joint Actions Based Authorization Schemes. *Operating Systems Review*, **30**(3):32–45.

BIOGRAPHY

Elisa Bertino is professor in the Department of Computer Science of the University of Milan. She has also been on the faculty in the Department of Computer and Information Science of the University of Genova, Italy. She has been a visiting researcher at the IBM Research Laboratory (now Almaden) in San Jose and at the Microelectronics and Computer Technology Corporation in Austin, Texas. Prof. Bertino is a co-author of the book "Object-Oriented Database Systems - Concepts and Architectures" (Addison-Wesley, 1993) and is on the editorial board of the IEEE Transactions on Knowledge and Data Engineering.

Elena Ferrari received a MS degree in Information Sciences from the University of Milan in 1992. Since November 1993, she has been a PhD student at the Department of Computer Science of the University of Milan. Her main research interests include database security, temporal data models and multimedia databases. On these topics she has published several papers. She has been a visiting researcher at George Mason University, VA and at Rutgers University, NJ.