A Flexible Authorization Mechanism for Relational Data Management Systems

ELISA BERTINO
Università degli Studi di Milano
SUSHIL JAJODIA
George Mason University
and
PIERANGELA SAMARATI
Università di Milano

In this article, we present an authorization model that can be used to express a number of discretionary access control policies for relational data management systems. The model permits both positive and negative authorizations and supports exceptions at the same time. The model is flexible in that the users can specify, for each authorization they grant, whether the authorization can allow for exceptions or whether it must be strongly obeyed. It provides authorization management for groups with exceptions at any level of the group hierarchy, and temporary suspension of authorizations. The model supports ownership together with decentralized administration of authorizations. Administrative privileges can also be restricted so that owners retain control over their tables.

Categories and Subject Descriptors: D.4.6 [**Operating Systems**]: Security and Protection—*access controls*; H.2.0 [**Database Management**]: General—*security, integrity, and protection*

General Terms: Security, Theory

Additional Key Words and Phrases: Access control mechanism, access control policy, authorization, data management system, relational database, group management support

A preliminary version of this article appeared under the title "Supporting Multiple Access Control Policies in Database Systems" in the *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, Oakland, CA, May 1996 [Bertino et al. 1996c].
The work of S. Jajodia was partially supported by the National Science Foundation under grants IRI-9303416 and INT-9412507 and by the National Security Agency under grant MDA904-94-C-6118.
Authors' addresses: E. Bertino, Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano, Milano, 20135, Italy; S. Jajodia, Center for Secure Information Systems and Department of Information and Software Engineering, George Mason University, Fairfax, VA 22030-4444; P. Samarati, Dipartimento di Scienze dell'Informazione, Università di Milano, Polo Didattico e di Ricerca di Crema, Via Bramante 65, 26013, Crema, Italy.

1. INTRODUCTION

Different data have often different protection requirements. This is true even for data within the same system or database and data owned, managed, and specified by the same user (see Example 14, Section 7). As a matter of fact, each of us would like to store information related to our everyday activity or personal information with different protection requirements: information that can be accessed by everybody, information we do not want anyone else to access, information that can only be accessed by specific users, and information that can only be accessed by everyone except some specific users. One way to achieve this would be to require users to specify for every object they own whether a *specific* access by a user should be allowed or denied by the authorization mechanism. This is not practical, however, since users generally do not think of specifying the protection requirements in terms of individual access requests, but in terms of higher-level access control policies. It is, therefore, desirable that the access control model be powerful and flexible enough to directly mirror the different protection requirements users would like to express. Some of the desirable features an authorization model should provide are as follows:

—*Support for positive and negative authorizations:* Positive authorizations state those accesses that are to be allowed; negative authorizations state accesses to be denied. Most existing authorization models only support one type of authorization, either positive or negative, thus applying either the closed or the open policy[1] *on all the data to which access is to be controlled.* Support for both types of authorizations allows users to express the protection requirements in terms of what should be allowed or, in a complementary way, what should be denied, depending on what is more convenient in a specific situation. For instance, if accesses to some data are to be allowed to only a few users, it is more convenient to express the protection requirements in terms of positive authorizations. In contrast, if accesses are to be allowed to all but a few users, it is more convenient to use negative authorizations instead.

—*Support for exceptions and strong enforcement:* Exceptions allow general statements to be overridden in particular cases. Exceptions in the authorizations allow users to state that some authorizations, which have been specified and should therefore be enforced, are overridden (i.e., these authorizations are not to be considered valid) in some situations. For instance, consider the case where all employees except `tim` should be allowed to read certain data. It would be desirable to give read privileges to the group composed of all the employees (see the support for grouping of subjects below), with an exception stated for `tim`. Without the support for exceptions, one would need to specify an authorization for each

[1]A *closed* policy permits specification of only positive authorizations and allows only those accesses that are explicitly authorized. In contrast, an *open* policy permits specification of only negative authorizations and allows only those accesses that are not explicitly denied.

individual employee. Note that support of both negative and positive authorization is needed to specify exceptions. In the example, one would need to insert a positive authorization for the group `employee` and a negative authorization for `tim`.

In a model supporting exceptions, it is also desirable that users be allowed to state, for each authorization they grant, whether the authorization admits exceptions or not. We call the ability of the users to grant authorizations that do not admit exception *strong* enforcement, in contrast to *weak* enforcement which refers to the ability to grant authorizations that admit exception.

—*Administration:* Every authorization model must include an administrative policy regulating the specification of authorizations. The administrative policy allows users to grant and revoke access authorizations from others. The most common administrative policies in use are the centralized policy (where a privileged user (or group of users) can specify authorizations) and the ownership policy (where each user can regulate the accesses on the objects he or she creates). More sophisticated decentralized policies allow owners to delegate others to grant/revoke authorizations (see the next requirement).

—*Possibility of delegation and control retainment:* Decentralized policies allow the creator of an object to delegate the privilege of specifying authorizations on the object to others. Decentralized administration has the important advantage that it permits delegation. It, however, has the drawback that the owner of an object may lose control with respect to who can access the object. Moreover, if an exception policy is in place, it is possible that the authorizations specified by the owner are overridden by authorizations specified by other users. Therefore, while decentralized administration is generally desirable, it is important to provide a mechanism for delegating only limited forms of administrative privileges (for example, delegating only specific accesses or only authorizations that can be overridden by the owner).

—*Support for grouping of subjects:* Support for grouping of subjects allows the specification of authorizations for collections of subjects. Subject grouping can be enforced by supporting either groups or roles. Groups define grouping of users, and roles define grouping of privileges [Jajodia et al. 1997]. Groups and roles are complementary concepts both of which augment the expressiveness of the model. The need for groups arises naturally. Often, authorizations need to be specified with reference to a collection of users that share some common characteristics (e.g., they are employed in the same department, work on a given project, have a certain citizenship status, and so on). If authorizations were to be specified only for single users, the user who specifies the authorization would need to keep track of which users should belong to the group and then grant and revoke authorizations according to the changes in the group membership. By grouping all the users in a group and specifying

authorizations for the group, this complex administration task is avoided. Authorizations specified for groups will apply to users as long as they retain their membership in the group; there is no need for the user who granted the authorization to keep track of these changes.

In this article we propose an authorization model for relational data management systems that meets the aforementioned requirements. The model supports positive and negative authorizations that can be either strong or weak. The classification of authorizations as strong and weak is useful because it permits users to specify whether an authorization admits or does not admit exceptions. Note that these concepts are not new and have already been used by other models. In particular, strong and weak authorizations were first introduced in the Orion model [Rabitti et al. 1991]. However, we believe that the treatment of negative authorizations in the Orion model, together with the resulting implication relationships between authorizations, makes Orion inappropriate for a flexible system. See Section 9.1 for a detailed comparison.

In our model, subject grouping is supported by a group hierarchy, and our exception policy is based on the structure of this group hierarchy. Our model allows authorization exceptions to be specified at any level in a group hierarchy.[2] In particular, positive authorizations and negative authorizations can be specified at any level. Thus it is possible to specify exceptions to exceptions. Negative authorizations in our model are handled as blocking authorizations. Whenever a user receives a negative authorization taking precedence over positive authorizations, these positive authorizations are retained in the authorization base even though they are not valid any longer. Positive authorizations become valid once again if the negative authorization is revoked. This approach makes it possible to temporarily suspend a privilege from a user by granting a negative authorization for the privilege and by revoking such negative authorization later on. In a system without negative authorization, the privilege would have to be revoked and then granted again later on.

We define an administrative policy by which the creator of a table can grant others administrative privileges on the table. Delegation can be specified for each privilege, and further delegation of the privilege by the recipient can be allowed or disallowed. Moreover, in our model, by delegating only the privilege of specifying weak authorizations, the creator of a table can retain control on the authorizations on the table.

The organization of this article is as follows. Section 2 introduces some notations and definitions. Section 3 presents our authorization model and discusses how access control is performed. Section 4 discusses administration of authorizations. Section 5 discusses the consistency of the authorization state, and Section 6 describes how possible conflicts between authorizations can be resolved. Section 7 illustrates how our model can be used to

[2]In a group hierarchy, a group can be a member of another group. Group nesting can be of arbitrary depth.

represent different policies and protection requirements. Section 8 illustrates the catalogs and the administrative tools used by the implementation. Section 9 surveys other authorization models supporting negative authorizations and group management. In Section 9.1 we describe the main differences between our authorization model and the authorization model of Orion, which first introduced the concept of strong and weak authorizations. Finally, Section 10 presents the conclusions.

2. NOTATIONS AND DEFINITIONS

In this section we introduce notations and definitions that will be used throughout the article and give the basic assumptions on the objects and the subjects of the model and on authorizations that subjects can hold on objects.

2.1 Objects and Privileges

In our model, objects to be protected are tables of the database. Tables can be either *base* tables or *view* tables. A view table (or simply, view) is a table defined in terms of queries on other base tables or views. In the following, notation $t \twoheadrightarrow v$ indicates that view v is built on top of table t (either directly or indirectly). We indicate with BT the set of base tables, with VT the set of view tables, and with $T = BT \cup VT$ the set of all tables in the system. We include views in our discussion since they are a good example of derived objects. Moreover, views represent an effective mechanism to support content-dependent authorizations, as illustrated by the following example.

Example 1. Suppose `Luke` creates table `Salary` and wants to give other users the authorization to select the tuples of `Salary` for which the value of attribute `department` is equal to "deptX." `Luke` can define a view of the form "select * from `Salary` where `department` = deptX" on top of `Salary`, and grant those users the select authorization on the view.

We consider only a privileged subset of users, specified by the database administrator, will be allowed to create tables (base tables or views) in the database.

Privileges that users can exercise on the tables and for which authorizations, either positive or negative, can be specified are all the actions executable on tables: `select` (to retrieve data), `update` (to modify data), `insert` (to write new data), `create-view` (to create view on the table), and so on.

2.2 Subjects

Subjects of authorizations can be either users or groups. Groups can be defined, dropped, and modified only by privileged users explicitly authorized for that. Members of groups can be users as well as other groups. The membership of a subject s in a group G_j can be either *direct* or *indirect*. The membership is direct, written $s \in_1 G_j$, if the subject is defined as a member

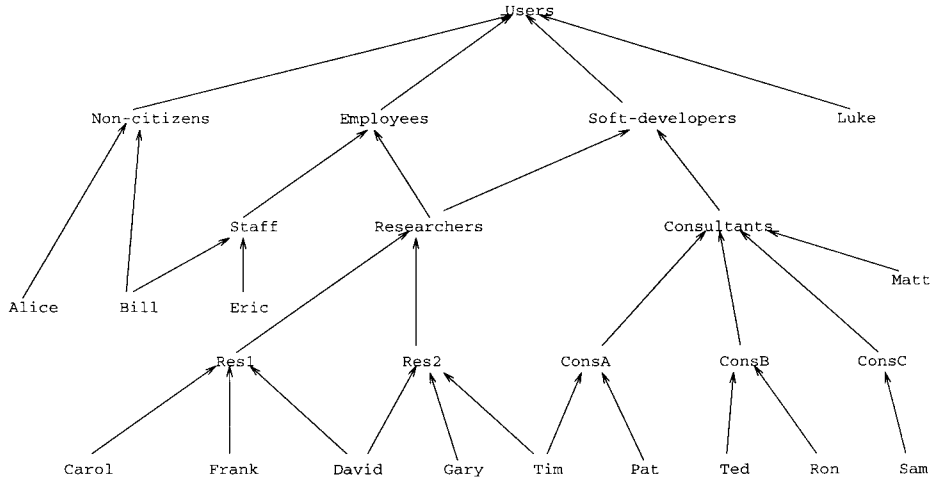


Fig. 1. An example of a membership graph.

of G. The membership is indirect, written $s \in_n G_j$, $n > 1$, if there exists a sequence of subjects $\langle s_1, \ldots, s_{n+1} \rangle$, such that $s_1 = s$, $s_{n+1} = G_j$, and $s \in_1 s_{i+1}$, $1 \leq i \leq n$. Sequence $\langle s_1, \ldots, s_{n+1} \rangle$ is called a membership path of s to G_j, written $mp(s, G_j)$. We say that s_i appears in a membership path mp if s_i is an element of sequence mp. We use $\in_0$ to indicate equality (i.e., for each subject s, $s \in_0 s$)[3] and the sequence $\langle s \rangle$ to represent the membership path of subject s to itself.

Groups need not be disjoint, i.e., a subject can belong to several groups. The only constraint on the group configuration is that a group cannot be its own member, neither directly nor indirectly. A subject can thus belong to a group in many ways, either as a direct or as an indirect member. We use $MP(s_i, s_j)$ to denote the set of all membership paths from subject s_i to subject s_j. We use the term membership to refer to either direct or indirect membership. We will explicitly use the terms direct membership or indirect membership when a distinction is needed.

We graphically represent the membership relationship between subjects by a graph, called the *group membership graph*, in which nodes represent subjects and in which an edge from node s_i to node s_j indicates that s_i is a direct member of s_j (i.e., $s_i \in_0 s_j$). Since, by assumption, no subject can be its own member, the group membership graph is a directed acyclic graph. An example of a group membership graph is illustrated in Figure 1.

Note that our treatment of subjects covers systems where no group concept is supported. In particular, a system with no group management can be represented with a subject structure where the set of subjects is the set of users in the system and where the only membership relationship

[3]We continue to use the symbol $\in$ to denote the membership of an element in a set.

holding is the equality ($\in_0$) of a subject to itself. The corresponding graph is a graph having a node for each user and no arcs connecting the nodes.

2.3 Authorizations

In our model, authorizations can be specified for privileges as well as for the negation (i.e., denial) of privileges. To indicate whether an authorization refers to a privilege or to its negation, the privilege type (+ or −) is included in the authorization. Negative authorizations can be specified only on base tables. The reason is that negative authorizations on views would make the access control too complex and difficult to use in practice. If negative authorizations were to be specified also on views, when a user accesses a view, all the authorizations on all the views directly or indirectly referencing some table the view references would have to be checked in order to prevent the user from accessing denied data. This approach is obviously not suitable in general where several views may be defined on the same table and subjects given different authorizations on the views. The following example illustrates the consequences of allowing negative authorizations on views.

Example 2. Consider a table `Fundings` and views V_1 and V_2 defined on it as follows:

V_1 = "select * from `Fundings` where `project` = software"

V_2 = "select * from `Fundings` where `code` = private."

Suppose that the views are not disjoint, i.e., some tuples are contained in both views. Suppose moreover that `Matt` should be allowed access to information about the software projects, but he should be denied access to information about private projects. In terms of the authorizations, `Matt` should have both the positive authorization for the select privilege on view V_1 and the negative authorization for the select privilege on view V_2. Consider the request from `Matt` to access view V_1. Although `Matt` is authorized to select tuples from V_1, with respect to the protection requirements, he should not be allowed to see the tuples in it which are also contained in V_2. Enforcing this restriction would entail comparison of the predicates used in defining the two views.

Note that in some cases, users can express protection requirements where a negative authorization on a view v would be required, by defining a view complementary to v and specifying a positive authorization on it. For instance, suppose a user is to be allowed to select all tuples of table t, but those satisfying predicate p. This requirement can be expressed by defining a view v' containing all the tuples for which predicate p is not satisfied and giving the user the positive authorization on view v'. With reference to the example above, a view V_3 could be defined as V_3 = "select

* from Fundings where project = software and code ≠ private" and Matt be given access to V_3.

Our model allows users to specify whether an authorization they grant admits or does not admit exceptions. Authorizations that admit exceptions are called *weak*. Authorizations that do not admit exceptions are called *strong*. Intuitively, strong authorizations must always be obeyed and cannot therefore be overridden. In contrast, weak authorizations can be overridden by other strong or weak authorizations, as we will explain in the following section.

Let U denote the set of users in the system, G the set of groups, S the set of subjects (with $S = U \cup G$), T the set of tables, and $P =$ {select, insert, delete, update} the set of privileges executable on the tables. Authorizations are defined as follows:

Definition 1 (Access Authorization). An *access authorization* a is a six-tuple $\langle s, p, pt, t, g, at \rangle$ where

—$s \in S$ is the subject (user or group) to whom the authorization is granted;

—$p \in P$ is the privilege for which the authorization is granted;

—$pt \in \{+, -\}$ specifies whether the authorization refers to the privilege or to its negation;

—$t \in T$ is the table to which the authorization refers ($t \in BT$ if $pt = -$);

—$g \in U \cup \{$system, adm$\}$ is the grantor of the authorization;[4]

—$at \in \{$strong, weak$\}$ indicates whether the authorization can be overridden.

Given an authorization a, we use the notation $s(a)$, $p(a)$, $pt(a)$, $t(a)$, $g(a)$, $at(a)$ to denote the subject, the access mode, the table, the time, the grantor, and the authorization type in a, respectively. For example, $s(a)$ is the subject in authorization a.

In the following we will use the terms access authorization and authorization interchangeably.

Example 3. Tuple ⟨Bill,select,+,Reports,Cathy,strong⟩ indicates that Bill can select tuples from table Reports and that this authorization, which cannot be overridden, has been granted by Cathy. Tuple ⟨Consultants,select,−,Fundings,Pam,weak⟩ indicates that group Consultants (i.e., its members) cannot select tuples from table Fundings and that this denial has been specified by Pam. However, since the authorization is weak, exceptions are admitted.

[4]Administration of authorizations will be discussed in Section 4. The meaning of terms system and adm will be explained in Section 4.

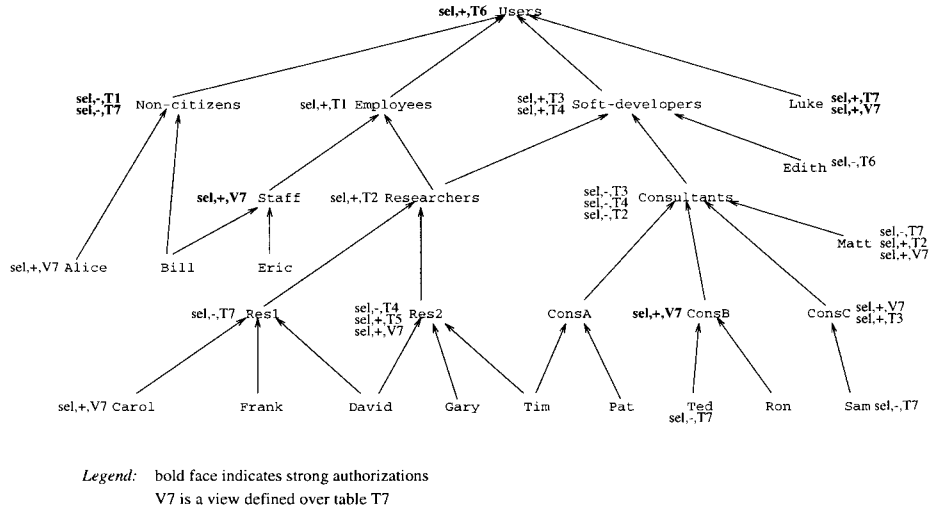


Fig. 2. An example of an authorization state.

The set of authorizations that are present at a given time represents the *authorization state*. We graphically represent the authorization state as follows. We indicate that subject s_i owns a positive authorization for privilege p on table t by associating triple $\langle p,+,t\rangle$ with node s_i in the group membership graph. Analogously, triple $\langle p,-,t\rangle$ associated with node s_i indicates that s_i owns a negative authorization for p on t. To distinguish between strong and weak authorizations, we write strong authorizations in bold type. An example of an authorization state is illustrated in Figure 2.

From now on we will indicate authorizations simply with the four-tuple $\langle s,p,pt,t\rangle$ if the grantor is not of interest in the explanation. As we do for the figures, we will distinguish strong authorizations from weak authorizations by writing them in bold type.

The authorizations of the owner of a view on the view are derived from the authorizations the user holds (either personally or as a member of a group) on the tables underlying the view. Since negative authorizations cannot be specified for views, possible negative authorizations the user may have on the underlying tables are not redefined for the owner on the view.[5] However, they are taken into account in the derivation of the authorizations for the owner on the view. In particular, the owner of a view will receive (and maintain) the authorization for a privilege on the view if and only if he or she has a valid[6] positive authorization for the privilege on

[5]It is possible for a user to define a view on a table even if he or she has some negative authorizations on the table. As an example, suppose a user has both the authorization for the select privilege and the negative authorization for the insert privilege on a table. The user can still define a view on the table. However, the only privilege received on the view is the select privilege.

[6]Informally, a positive authorization is valid for a subject if it is neither overridden nor conflicting over the subject. We will elaborate on this concept in Section 3.

every table directly referenced by the view. This derived authorization is strong if the user has a strong authorization for the privilege on every table directly referenced by the view. It is weak otherwise. Since the authorizations of the owner of a view on the view depend on his or her authorization on the tables underlying the view, changes to these latter authorizations may affect the authorizations on the view. Algorithms for maintaining authorizations derived for views can be used for this purpose [Griffiths and Wade 1976; Selinger 1990]. Note that it is important that authorizations derived for the creator of a view be restricted by those authorizations that the creator holds on the base tables. If the creator of the view was given full privileges on the view because of his or her ownership, it would be possible for users to bypass the restrictions stated by the authorizations simply by defining new views.

3. THE AUTHORIZATION MODEL

In this section we present our authorization model. We start with an overview of the basic concepts of the model and then give the formal definitions and discuss access control.

3.1 Overview

In our model, authorizations can be specified for single users, as well as for groups of users. At any time, a user holds, beside his or her own authorizations, all the authorizations of the groups to which he or she belongs. We do not restrict the user to the authorizations of only one group at a time. Our approach accords with the semantics of user groups. Unlike roles (which are dynamic and users can act in a role or not, at their choice) groups are static, in the sense that a member of a group always carries its membership to the group [Jajodia et al. 1997]. To clarify the difference between the two concepts, consider group `Non-citizens` in our reference group membership graph. It is not correct to allow users to take or release this group at will. Noncitizens are always as such, and the privileges/denials specified for the group `Non-citizens` always apply to its members. Moreover, although authorizations can be specified for both users and groups, access requests are always submitted by users operating as single individuals. Again, this assumption comes from our view on the semantics of user groups in contrast with roles [Jajodia et al. 1997]. These assumptions on subjects are common to most models considering users' groups [Abadi et al. 1991; Castano et al. 1995; Jajodia et al. 1997].

The introduction of both strong authorizations, which do not allow for exceptions, and weak authorizations, which do allow for exceptions, increases the expressiveness of the authorization model. When decentralized administration is in place, the use of strong authorizations may be the only means users have to make sure the authorizations they specify will certainly be obeyed. In contrast, weak authorizations allow for exceptions. Suppose no distinction between strong and weak authorizations was made. Then either all authorizations behave as strongly (i.e., they do not allow

exceptions) or as weakly (i.e., they do allow exceptions). In the first case, it would not be possible to support exceptions to authorizations. In the second case, it would not be possible to enforce with certainty the authorizations.

The basic principles behind our model can be summarized as follows:

—Authorizations can be either negative or positive, and either strong or weak.

—Strong authorizations must always be obeyed. Hence, a subject cannot hold at the same time a positive and a negative strong authorization for an access.

—If a subject holds a negative and a positive authorization for an access such that one of the authorizations is strong and the other is weak, the strong authorization overrides the weak authorization.

—If a subject holds a negative and a positive weak authorization for an access, whether an authorization overrides the other depends on the (membership) relationships between this subject and the subjects for which the authorizations are specified.

—A subject is allowed an access only if he or she holds (1) a positive strong authorization or (2) a positive weak authorization which is neither overridden nor conflicting.

In the remainder of this section we formally define the model and the overriding and access control policies.

3.2 Formal Definitions

According to the semantics of authorization types, it is obvious that whenever a strong and a weak authorization contrast, i.e., one states that an access should be authorized, while the other states that the same access should be denied, the strong authorization takes precedence. It is also obvious that the situation where two strong authorizations contrast cannot be accepted. The case where two weak authorizations contrast is less obvious. In our model, the decision of which of the authorizations (if any) should take precedence is based on the concept of more specific authorization. Intuitively, authorizations given to a member of a group are always more specific than authorizations given to the group over the membership paths passing from both the member and the group. The reason why the overriding relationship is referred to specific membership paths is to take into consideration the fact that a subject can belong to a group through different membership paths. Although an authorization can be overridden for a subject over a membership path, the authorization can reach the subject through other membership paths.

Example 4. Given the membership graph in Figure 1, the authorizations specified for group `Soft-developers` can reach `Tim` through both the membership path passing from `Researchers` and the membership path passing from `Consultants`. Authorizations specified for `Consultants`

(respectively, Researchers) override authorizations specified for Soft-developers only over the membership path passing through group Consultants (respectively, Researchers).

The overriding rule between authorizations is formalized as follows.

Definition 2 (Overriding Authorization). An authorization a_i *overrides* an authorization a_j, with $p(a_i) = p(a_j)$, $pt(a_i) \neq pt(a_j)$, and $t(a_i) = t(a_j)$ or $t(a_i) \twoheadrightarrow t(a_j)$ with respect to subject s, $s \in_n s(a_i)$, $s \in_m s(a_j)$, $(n, m \geq 0)$ over a path $mp \in MP(s, s(a_j))$ (written $a_i \triangleright_s^{mp} a_j$) if and only if any of the following conditions is satisfied:

—$at(a_i)$ = "strong" and $at(a_j)$ = "weak";

—$at(a_i) = at(a_j)$ = "weak"; $s(a_i) \neq s(a_j)$; and $s(a_i)$ appears in mp.

If authorization a_i overrides a_j over all the membership paths of s to $s(a_j)$, we simply say that a_i overrides a_j with respect to s (written $a_i \triangleright_s a_j$).

Definition 2 states that

—a strong authorization a_i overrides a weak authorization a_j, with the same privilege, a different privilege type, and either the same table or such that $t(a_j)$ is a view referencing table $t(a_i)$, with respect to any subject s which belongs to both $s(a_i)$ and $s(a_j)$ over all membership paths from s to $s(a_j)$;

—a weak authorization a_i overrides a weak authorization a_j, with same privilege, different privilege type, and either same table or such that $t(a_j)$ is a view referencing table $t(a_i)$, with respect to a subject s belonging to $s(a_i)$, with $s(a_i) \neq s(a_j)$, over all membership paths from s to $s(a_j)$ passing through $s(a_i)$.

Example 5. Consider the authorization state of Figure 2. Some overriding relationships are as follows:

$\langle$**Non-citizens,sel,−,T$_1$**$\rangle \triangleright_{\texttt{Bill}} \langle \texttt{Employees,sel,+,T}_1 \rangle$;

$\langle$**Users,sel,+,T$_6$**$\rangle \triangleright_{\texttt{Edith}} \langle \texttt{Edith,sel,−,T}_6 \rangle$;

$\langle$**Matt,sel,+,T$_2$**$\rangle \triangleright_{\texttt{Matt}} \langle \texttt{Consultants,sel,−,T}_2 \rangle$;

$\langle$**ConsC,sel,+,T$_3$**$\rangle \triangleright_{\texttt{Sam}} \langle \texttt{Consultants,sel,−,T}_3 \rangle$;

$\langle$**Res2,sel,-,T$_4$**$\rangle \triangleright_{\texttt{Tim}}^{mp} \langle \texttt{Soft-developers,sel,+,T}_4 \rangle$
for mp = $\langle$**Tim,Res2,Researchers,Soft-developers**$\rangle$;

$\langle$**Consultants,sel,-,T$_4$**$\rangle \triangleright_{\texttt{Tim}}^{mp} \langle \texttt{Soft-developers,sel,+,T}_4 \rangle$
for mp = $\langle$**Tim,ConsA,Consultants,Soft-developers**$\rangle$.

$\langle$**Non-citizens,sel,−,T$_7$**$\rangle \triangleright_{\texttt{Alice}} \langle \texttt{Alice,sel,+,V}_7 \rangle$;

$\langle \texttt{Sam,sel,−,T}_7 \rangle \triangleright_{\texttt{Sam}} \langle \texttt{ConsC,sel,+,V}_7 \rangle$.

In our model, if two contrasting authorizations are specified, one for a group G and one for a member m of G, although the authorization for m overrides the authorization for G with respect to m, it may not necessarily be so with respect to the members of m. The reason behind our approach is illustrated by the following example.

Example 6. Consider the group membership graph of Figure 1. Suppose Luke creates a table T_2 containing the evaluation of the consultants working for the company. The table should be read by the researchers of the company but, obviously, should not be read by the consultants, unless explicitly authorized. In particular, Matt can read it. This situation can be expressed by specifying a weak negative authorization for the group Consultants and a positive authorization for group Researchers and for Matt. The privilege type of the authorizations for Researchers and for Matt depends on whether Luke wishes them to be strongly obeyed or to allow for exceptions. There is a person in the company, Tim, who works as both a researcher in group Res2 and a consultant in group ConsA. The question arises of whether Tim should be granted the access in view of his membership in Res2 or should be denied the access due to his membership in ConsA. If the authorization granted to Researchers is strong, obviously the access should be granted. If the authorization granted to Researchers is weak (Figure 2), the system cannot decide a priori which of the two authorizations give precedence. Thus, neither of the two authorizations takes precedence over the other.

Consider now a similar case where Bill creates a table T_3 and wants to give all soft-developers except consultants the authorization to read this table. Moreover, among the consultants, those in group ConsC should be authorized. This requirement can be expressed by specifying a negative weak authorization for Consultants and a positive authorization for Soft-developers and for ConsC. Again, the privilege type of these authorizations depend on whether Bill wants them to be strongly obeyed. Consider the case where the authorization for Soft-developers is weak (Figure 2). Consider the authorizations for user Tim. Again, we claim the system cannot decide a priori. In this case, the reason is that a different semantics could be assigned to the presence of the authorizations for Soft-developers and Consultants. A permissive interpretation is that Consultants are not authorized, but all other Soft-developers are. In this case Tim, who belongs to Soft-developers also in virtue to its membership in Researchers, should be allowed to read the table, i.e., the positive authorization should take precedence. A stricter interpretation is that only the Soft-developers who are not consultants can read the table. In this case the negative authorization should take precedence. Therefore, no decision can be taken by the system. For this reason, neither of the two authorizations is considered to take precedence over the other.

Authorizations that are applicable, i.e., that must be considered for a subject, are then defined as follows:

Definition 3 (Applicable Authorization). An authorization a_j is *applicable* to a subject $s \in_n s(a_j)$, with $n \geq 0$, if and only if there exists a membership path of s to $s(a_j)$ over which a_i is not overridden with respect to s. Formally, a_i is applicable to s if and only if $\exists mp \in MP(s, s(a_j))$ such that $\nexists a_i \in AS$, $a_i \triangleright_s^{mp} a_j$.

An authorization a_j which is not applicable to a subject is said to be *ineffective* for the subject. In terms of the overriding relationship, an authorization is ineffective for a subject s, with s equal to or member of $s(a_j)$, if it is overridden over all the membership paths from s to $s(a_j)$.

Example 7. Consider the authorization state illustrated in Figure 2.

Authorization ⟨Soft-developers, sel, +, T_3⟩ is applicable to Tim;

authorization ⟨Soft-developers, sel, +, T_4⟩ is ineffective for Tim;

authorization ⟨Employees, sel, +, T_1⟩ is ineffective for Bill;

authorization ⟨ConsC, sel, +, V_7⟩ is ineffective for Sam.

Two authorizations with the same privilege and table but different privilege type in which both are applicable to a subject are said to be in conflict with respect to the subject. Two authorizations with the same privilege and table but different privilege type such that both are applicable to a subject are said to be in conflict with respect to the subject. Conflicting authorizations are formally defined as follows:

Definition 4 (Conflicting Authorizations). Two authorizations a_i and a_j *conflict* with respect to subject s to which they are applicable (written $a_i \diamondsuit_s a_j$), if and only if $p(a_i) = p(a_j)$, $pt(a_i) \neq pt(a_j)$, and any of the following conditions is satisfied:

—$at(a_i) = at(a_j) =$ "strong" and $(t(a_i) = t(a_j) \vee t(a_i) \twoheadrightarrow t(a_j))$

—$at(a_i) = at(a_j) =$ "weak" and $t(a_i) = t(a_j)$.

Definition 4 states that two authorizations with the same privilege but with different privilege type conflict with respect to a subject to which both of them are applicable if either

both authorizations are strong; and the tables in the authorizations are either equal, or one references the other; or

both authorizations are weak and they are on the same table.

Example 8. Consider the authorization state of Figure 2. The following conflicts hold:

⟨Soft-developers, sel, +, T_3⟩ $\diamondsuit_{\text{Tim}}$ ⟨Consultants, sel, −, T_3⟩;

⟨Researchers, sel, +, T_2⟩ $\diamond_{\text{Tim}}$ ⟨Consultants, sel, −, T_2⟩;

⟨**Non-citizens,sel,−,T_7**⟩ $\diamond_{\text{Bill}}$ ⟨**Staff, sel, +, V_7**⟩.

According to our definitions, two weak authorizations for the same privilege but different privilege type such that the table in one of them is a view referencing the table in the other which are both applicable on a subject do not conflict. The subject will be allowed to access the view but will not be allowed to access the whole table. The reason for this is that a view is considered as more specific than the tables on which it is defined. Although the (positive) authorization on the view takes precedence with respect to the (negative) authorization on the base table, we cannot talk about overriding here. As a matter of fact, the negative authorization is not overridden, but it is applicable to the subject that, although allowed to access the view, will not be allowed to directly access the table itself. For instance, with reference to the authorization state of Figure 2, Carol, David, Matt, and Ted will be allowed the select access to view V_7 but will be denied the select access to table T_7.

Note that this reasoning only applies to weak authorizations, where not obeying the denial stated by the negative authorization on the base table accords with the semantics of the specifications. It does not apply to strong authorizations, where the semantics requires that the negative authorization on the base table be always obeyed, not permitting exceptions for more specific subjects or objects.

Since strong authorizations must be necessarily obeyed, conflicts between strong authorizations are not allowed. The reason for this is that if two strong authorization conflicts, any conflict resolution policy would require to override one of the authorizations, therefore not obeying the principle that strong authorizations cannot be overridden.

If two strong authorizations exist which conflict with respect to a given subject, we say that the authorization state is *inconsistent*; it is *consistent* otherwise. This is formalized by the following definition.

Definition 5 (Consistency of the Authorization State). The authorization state is *consistent* if and only if strong authorizations do not conflict (with respect to any subject). Formally, the authorization state is consistent if and only if $\nexists s \in S$, a_i, $a_j \in AS$, such that $at(a_i) = at(a_j) =$ "strong," an $a_i \diamond_s a_j$.

It is the task of the access control system to ensure the consistency of the authorization state. In particular, execution of an operation modifying the authorization state (e.g., grant or revocation of authorizations) or the group membership graph (e.g., addition or removal of members from groups) will succeed only if the resulting authorization state is consistent (see Section 5).

By contrast, we admit conflicts between weak authorizations. Our decision is justified by the fact that since weak authorizations allow for exceptions, conflicts can always be resolved. Users can either use excep-

tions to explicitly resolve conflicts at their will (see Section 5) or can rely on the default policy used by the system to resolve conflicts at the access control time (see Section 3). This default policy, which obeys the denials-take-precedence principle, can be seen as an exception policy to be used when no other exceptions have been specified by the users.

3.3 Access Control

The access control is based on the concept of valid authorization. An authorization for a privilege on a table is considered *valid* for a given subject only if the authorization is applicable to the subject and does not conflict with other authorizations for that subject, as expressed by the following definition.

Definition 6 (Valid Authorization). An authorization $a \in AS$ is *valid* for a given subject s, $s \in_n s(a)$, for which a is applicable if and only if $\nexists a_i \in AS$, $a_i \diamond_s a$.

An access request is granted if and only if there exists a valid positive authorization for it. We view the access control as a function *access*() that, given an access request $\langle u,p,t \rangle$ stating that user u is requesting to exercise privilege p on table t, returns `true` if the request must be authorized and `false` otherwise. Function *access*() is defined as follows:

$$\textbf{access(u, p, t)} = \begin{cases} strong\_auth(u, p, t) & \text{if strong\_auth(u, p, t)} \\ & \neq \texttt{undecided} \\ weak\_auth(u, p, t) & \text{otherwise} \end{cases}$$

where functions *strong_auth*() and *weak_auth*() are as follows:

$$\textbf{strong\_auth(u, p, t)} = \begin{cases} \texttt{true} & \text{if } \exists \text{ is a strong positive} \\ & \text{authorization for the access} \\ \texttt{false} & \text{if } \exists \text{ is a strong negative} \\ & \text{authorization for the access} \\ \texttt{undecided} & \text{otherwise} \end{cases}$$

$$\textbf{weak\_auth(u, p, t)} = \begin{cases} \texttt{true} & \text{if } \exists \text{ is a positive weak authorization} \\ & \text{for the access valid for u} \\ \texttt{false} & \text{otherwise} \end{cases}$$

The algorithms implementing functions *strong_auth*() and *weak_auth*() can be found in Bertino et al. [1996b]. We briefly describe them here.

The algorithm implementing function *strong_auth*() checks the strong authorizations of the user requesting the access. If some authorization, either positive or negative, is found the reply is returned accordingly. Otherwise, if no authorization is found, the authorizations of the groups to which the user belongs are checked. Again, if some authorization is found

the reply is returned according to the privilege type in the authorization. If no authorization is found, value `undecided` is returned. Note that if the request is on a view table, function *strong_auth*() controls, beside the authorizations on the view, also the negative authorizations on the base tables underlying the view. Note also that there is no need to examine all the authorizations; the algorithm terminates at the first authorization (positive or negative) found.

The algorithm implementing function *weak_auth*() works as follows. Consider first the case that the request is on a base table. The negative authorizations of the user requesting the access are checked. If a negative authorization for the access is found, then a `false` is returned, and the algorithm terminates. Otherwise the positive authorizations of the user are checked. If a positive authorization is found, value `true` is returned, and the algorithm terminates. (Note that since weak authorizations may conflict it is important to first look for a negative authorization, and only in case no negative authorization is found look for a positive authorization.) If no authorization is found for the user, the authorizations of the groups to which the user belongs are checked. Groups are considered by traversing the membership paths from the user in increasing distance. Whenever a positive authorization is found for a group, it is not necessary to proceed further on that membership path, since this authorization overrides those of the groups above on that membership path. Hence, either the authorizations of those groups are ineffective for the user, or those groups are reachable through some other membership paths. The algorithm terminates when either a negative authorization is found or when there are no more groups to be considered. In the first case `false` is returned. In the second case, a `false` is returned if no positive authorization was found; a `true` is returned otherwise.

Consider now the case of view tables. The algorithm first checks if the user holds a positive authorization on the view. If so, a `true` is returned, and the algorithm terminates. Otherwise, the algorithm checks if the user holds a negative authorization on any of the base tables underlying the view. If such an authorization is found, a `false` is returned, and the algorithm terminates. Note that, unlike for base tables, in this case positive authorizations must be checked first because the positive authorization on the view is valid even in presence of negative authorizations on the base tables. If no authorization is found for the user, the groups to which the user belongs are considered by traversing the membership paths from the user in increasing distance. For each group the existence of positive authorizations on the view and of negative authorizations on base tables underlying the view is checked. Whenever a negative authorization is found for a group, the algorithm does not proceed any longer on that membership path because the negative authorization on the table overrides, over that membership path, all the possible authorizations of the groups above for the view. The process terminates when either there are no more groups to be considered, or a positive authorization is found. In the first case a `false` is returned. In the second case a `true` is returned.

Example 9. Consider the authorization state illustrated in Figure 2.

Request (Bill,sel,T_1) is denied because of authorization ⟨**Non-citizens,sel,−,T_1**⟩;

request (Tim,sel,T_4) is denied because of authorization ⟨Res2,sel,−,T_4⟩;

request (David,sel,T_2) is authorized thanks to authorization ⟨Researchers,sel,+,T_2⟩.

request (David,sel,V_7) is authorized in view of authorization ⟨Res2,sel,V_7⟩.

Access control therefore requires the evaluation of the authorizations of the users and, possibly, of the groups to which the user belongs. The cost of the access control is the cost C_s of evaluating function *strong_auth* and, possibly, the cost C_w of evaluating function *weak_auth*, where C_w applies only if no strong authorization has been found. This cost can be expressed in terms of the number of subjects whose authorizations must be evaluated to determine whether a user request must be granted or denied. Consider a request by user u belonging to N_u groups to access a base table. As for the control of strong authorization, where the order in which the authorizations of the groups are checked is not important, the cost C_s is 1 in the best possible case (where a strong authorization exists for the user) and N_u in the worst possible case (where no strong authorization exists for the users of the groups to which the user belongs).

The algorithm implementing function *weak_auth* evaluates the group in a particular order, from the bottom of the hierarchy to the top with a breadth-first strategy traversing the different paths from the user to the root/s. This evaluation method allows us to control authorization overriding without the need of further control (the first authorization found on a path when going bottom up overrides those above) and minimizes the number of subjects whose authorizations must be controlled. The cost C_w of this algorithm is therefore equal to

—1 , if a weak authorization exists for the user;

—$1 + \sum_{i=1}^{i=m} \min(w_i, h_i)$, if (1) no authorization exists for the user and (2) no valid negative authorization exists for any of the groups to which he or she belongs;

—$1 + \sum_{i=1}^{i=m} \min(w_i, h_i, \bar{n} - 1) + r$, otherwise;

where m is the number of different paths from the user to a root node; h_i is the length of the ith path; w_i is the minimum distance from u to a group with a weak authorization on the ith path; $\bar{n}$ is the minimum distance from u to a group with a nonoverridden negative authorization; and r is 1 in the

best case and is equal to the number of paths i such that $\min(h_i, w_i) \geq \bar{n}$ in the worst case.

Note that when illustrating the algorithms we have assumed that the groups to which a user directly or indirectly belongs is always known. In particular, to efficiently enforce function *strong_auth* it is important that all the groups to which the user belong be known without the need to determine them as the transitive closure of the direct membership relationship every time. Strategies to efficiently determine and maintain transitive closures such as the one proposed by Gagliardi et al. [1989] can be used for this purpose.

4. ADMINISTRATION OF AUTHORIZATIONS

The user creating an object is considered the owner of the object and as such is the only one allowed to drop it. When a user creates a base table, he or she receives a strong positive authorization for all the privileges on the table. These authorizations, which have the system as grantor (denoted as "`system`") will be deleted upon deletion of the table. As owner, the user can grant other users authorizations (positive/negative, strong/weak) for the privilege on the table. The user can also grant other users the administrative authorizations for privileges on the table. We consider two types of administrative privileges: `adm-access` and `administer`. If a user has the `adm-access` privilege for a privilege on a table, the user can grant and revoke other subject's authorizations (positive/negative) for the privilege on the table. If a user has the `administer` privilege for a privilege on a table, the user can grant and revoke other subject's authorizations (positive/negative) for the privilege on the table as well as authorizations for the administration of the privilege on the table.

A type (strong or weak) is also associated with each administrative authorization. This type does not refer to the administrative authorization itself but to the authorizations which can be granted through it, as illustrated in Table I. Administrative authorizations themselves are not classified as strong or weak, as no overriding policy is applied on them. Administrative authorizations cannot be overridden (intuitively they are always strong). To ensure that no subject will be allowed to administer an access if it could not receive itself an authorization for it, we require subjects with an (explicit or implicit) administrative authorization for a privilege on a table to not hold any (implicit or explicit) strong negative authorization for the privilege on the table.

Definition 7 (Administrative Authorization). An *administrative authorization* a is a six-tuple $\langle s,p,ap,gat,t,g \rangle$ where

—$s \in S$ is the subject (user or group) to whom the authorization is granted;

—$p \in P$ is the privilege to which the authorization refers;

—$ap \in$ {`adm-access`,`administer`} is the administrative privilege;

Table I. Administrative Authorizations and Their Semantics

Administrative Authorization	Semantics
⟨*s*,*p*,adm-access,strong,*t*,*g*⟩	can grant and revoke weak/strong positive/negative access authorizations for *p* on *t*
⟨*s*,*p*,adm-access,weak,*t*,*g*⟩	*s* can grant and revoke weak positive/negative access authorizations for *p* on *t*
⟨*s*,*p*,administer,strong,*t*,*g*⟩	*s* can grant and revoke weak/strong positive/negative access authorizations for *p* on *t* *s* can grant and revoke weak/strong authorizations for the administration of *p* on *t*
⟨*s*,*p*,administer,weak,*t*,*g*⟩	*s* can grant and revoke weak positive/negative access authorizations for *p* on *t* *s* can grant and revoke weak authorizations for the administration of *p* on *t*

—$gat \in$ {strong,weak} indicates the type of authorizations which *s* can grant;

—$t \in T$ is the table to which the authorization refers;

—$g \in U$ is the grantor of the authorization;

For instance, authorization ⟨Luke,sel,admin-access,weak,T_1,Cathy⟩ states that Luke can grant, and revoke, other subjects' weak authorizations for the select privilege on table T_1 and that this authorization was granted by Cathy.

Note that we do not separate the administration of positive and negative authorizations: if a user has an administrative authorization for a privilege on a table the user can specify both permissions and denials for the privilege. Since negative/positive authorizations can be specified as exceptions to other negative/positive authorizations, we believe that users allowed to specify negative authorizations should also be allowed to specify positive authorizations and vice versa. However, the model could be simply extended to include different administrative authorizations for the administration of permissions and denials.

As we noted earlier, administrative authorizations are not classified as strong or weak, since they do not admit exceptions. The reason for this choice is that allowing overriding of administrative authorizations would raise the problem of dealing with the authorizations granted by a user whose administrative authorizations become overridden. Taking no effect on them would imply that the authorizations granted by a user are still valid while the administrative authorizations of the user are not. Propagating the "overriding" effect would make the authorization administration task very complicated, since the overriding relationship should not only

consider the groups configuration but also the grantors of the authorizations. We believe such a complicated framework to be unnecessary.

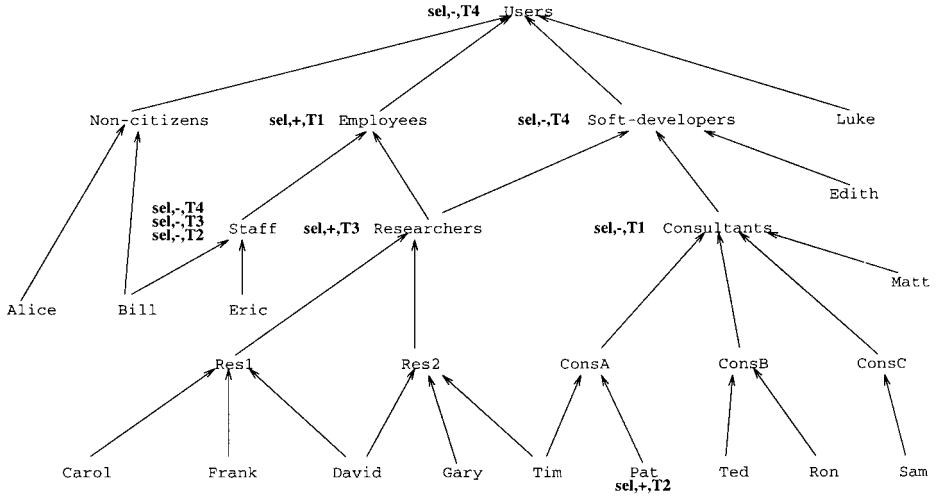
When a user defines a view, administrative authorizations on the view are derived for the user from the authorizations on the tables used in the view definition. In order to receive an administrative authorization for a privilege on a view, the user must have an administrative authorization for the privilege on every table used in the view definition. If the user holds an `administer` authorization for a privilege on every table directly referenced by the view, he or she will receive the `administer` authorization for privilege on the view. Otherwise, if the user holds either the `adm-access` or the `administer` authorization for a privilege on every table directly referenced by the view, he or she will receive the `adm-access` authorization for the privilege on the view.

Authorizations can be revoked only by the users who granted them. Moreover, a user can appear as grantor of an authorization for a privilege on a table only if the user holds an administrative authorization for the privilege on the table. As a consequence, when a user is revoked the administrative authorization for a privilege on a table, a recursive revocation, may take place to delete the authorizations granted by the user or derived for the users on views. Revocation algorithms enforcing recursive deletion of authorizations proposed in other models [Bertino et al. 1997; Fagin 1978; Gagliardi et al. 1989; Griffiths and Wade 1976; Informix 1993] can be easily adapted to our model. Note that different approaches can be taken to revocation in case this recursive deletion of authorization is not wanted. For instance, a *nonrecursive* revocation can be enforced where a revoke operation is not allowed if it would imply deletion of further authorizations [Melton 1990] or where authorizations granted by the user from whom the administrative privilege is revoked are not deleted but respecified with, as grantor, the user who required revocation [Bertino et al. 1997].

5. AUTHORIZATION STATE CONSISTENCY AND CONFLICT RESOLUTION

The collection of authorizations applicable to subjects may change upon execution of administrative operations by the users. These are operations affecting the authorization state (i.e., granting and revocation of privileges on tables), operations affecting the group membership graph (i.e., addition and removal of members from groups), and operations affecting the tables (i.e., creation and deletion of tables). Although some of these operations do not have any effect on the authorizations themselves, they may affect the authorizations applicable to subjects and may therefore introduce conflicts. For instance, if a user is added to a group, the authorizations of the group may become applicable to the user.

As stated in Section 3, we require that strong authorizations do not conflict. By contrast, we permit conflicts among weak authorizations. In the remainder of this section we illustrate how administrative operations can



Legend: bold face indicates strong authorizations

Fig. 3. An example of authorization state.

affect the consistency of the authorization state and how users can resolve conflicts among weak authorizations in accordance with their needs.

5.1 Ensuring Consistency of the Authorization State

Operations that decrease the authorizations applicable to the subjects (i.e., revocation of authorizations, removal of members from groups, or deletion of tables) do not obviously affect consistency. By contrast, operations that may increase the strong authorizations applicable to the subjects (grant of strong authorizations, insertion of new members in groups, and creation of new tables) may result in an inconsistent authorization state and must therefore be controlled. We have already discussed, in Sections 3 and 4, how the new authorizations to be assigned to the user defining a new table are determined. The derivation of new authorizations upon creation of new tables (either base tables or view tables) does not affect consistency, i.e., applied to a consistent authorization state it always produces a consistent authorization state. The proof of this statement is trivial. A user is given a strong authorization on a view only if he or she holds a strong authorization on all the tables used in the view definition. Hence, if the new authorization would conflict, a conflict should have existed among the authorizations on the table underlying the view, which would contradict the assumption that the authorization state before the operation was consistent.

Let us now consider the operations of granting a strong authorization and adding a member to a group.

Suppose a user wishes to insert a new strong authorization a. Inconsistencies may arise if subject $s(a)$, any member of $s(a)$, or any group to which

$s(a)$ or any of its members belongs already owns a strong authorization for privilege $p(a)$ on a table t such that $t = t(a)$, $t \twoheadrightarrow t(a)$, or $t(a) \twoheadrightarrow t$, with a privilege type different from $pt(a)$.

Example 10. Consider the authorization state illustrated in Figure 3, and suppose that a new strong positive authorization for the select privilege on table $\mathtt{T}_4$ is being granted to group `Employees`, i.e., $a = \langle \mathbf{Employees,sel,+,T_4} \rangle$. The following conflicts would be introduced:[7]

$\langle \mathbf{Employees,sel,+,T_4} \rangle \diamondsuit_{\mathtt{Employees}} \langle \mathbf{Users,sel,-,T_4} \rangle$

$\langle \mathbf{Employees,sel,+,T_4} \rangle \diamondsuit_{\mathtt{Staff}} \langle \mathbf{Staff,sel,-,T_4} \rangle$

$\langle \mathbf{Employees,sel,+,T_4} \rangle \diamondsuit_{\mathtt{Researchers}} \langle \mathbf{Soft\text{-}developers,sel,-,T_4} \rangle$

Consider the addition of a member m to a group G_k. Inconsistencies may arise if m, any of its members, or any of the groups to which m itself or any of m's members belong own some authorization which conflicts with an authorization owned by group G_k or by any of the groups to which G_k belongs.

Example 11. Consider the authorization state illustrated in Figure 3. Suppose that group `ConsA` is to be temporarily included in the group `Staff`. A request to add group `ConsA` to group `Staff` is submitted to the system. The following conflicts would be introduced:

$\langle \mathbf{Staff,sel,-,T_2} \rangle \diamondsuit_{\mathtt{Pat}} \langle \mathbf{Pat,sel,+,T_2} \rangle$

$\langle \mathbf{Employees,sel,+,T_1} \rangle \diamondsuit_{\mathtt{ConsA}} \langle \mathbf{Consultants,sel,-,T_1} \rangle$

$\langle \mathbf{Staff,sel,-,T_3} \rangle \diamondsuit_{\mathtt{Tim}} \langle \mathbf{Researchers,sel,+,T_3} \rangle$

Algorithms that determine if the insertion of a strong authorization or the addition of a member to a group would result in an inconsistent authorization state together with a proof of their correctness can be found in Bertino et al. [1996b]. We briefly describe them here.

The algorithm for controlling the authorization state consistency upon insertion of a new authorization works as follows. Suppose a new authorization a is being granted. The algorithm controls the authorizations of $s(a)$, of the subjects that belong to $s(a)$, and of the subjects to which either $s(a)$ or any of its members belong. The set of authorizations with the same privilege as a, different privilege type, and with a table either equal, referencing, or referenced by $t(a)$ is determined. If the set is empty, then no conflicts would be introduced, and `ok` is returned. Otherwise, for each authorization found, the conflicts which would arise are returned. Note, that, if two authorizations conflict over a group and one of its member, only the conflict over the group is returned.

[7]Note that to avoid unnecessary long lists of conflicts, if an authorization conflicts with the new authorization over two subjects s and s' such that s belongs to s', only the conflict over s' is returned.

The algorithm controlling the authorization state consistency upon addition of a member m to a group G_k checks all the authorizations specified for G_k or for the groups to which G_k belongs and to which m does not belong against the authorizations specified for either (1) the subjects which belong to m and do not belong to G_k (including m itself) or (2) the groups to which some members of m belong and that are neither a member of m nor a member of G_k and to which neither G_k or any of its members belong. If two authorizations are found with the same privilege, different privilege type, and with the table either equal, or such that the table in one of the authorizations references the table in the other authorizations, the conflict which would be introduced is returned. Note that also in this case, if two authorizations conflict over a group and one of its members, only the conflict over the group is returned.

5.2 Resolving Conflicts among Weak Authorizations

In case of simultaneous presence of conflicting weak authorizations for an access, our model denies the access. However, using the default of denial is not the only option for resolving conflicts among weaj authorizations. Users can resolve conflicts by inserting additional authorizations in the authorization state. This means that the users do not have to accept the system default option if they do not want; they can also eliminate conflicts in ways that best suit their needs.

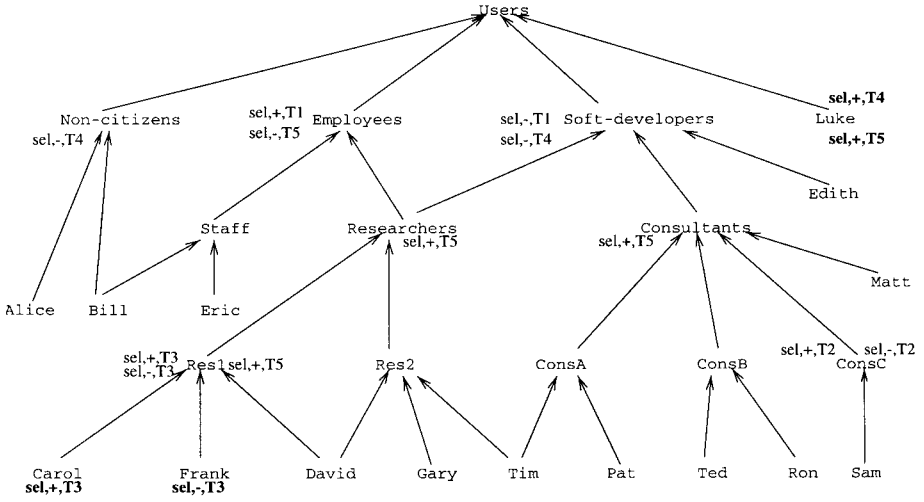
We claim that every conflict, with one exception, can be resolved by the addition of a new authorization. The exception occurs when conflicting authorizations are specified for the same subject.

To see this, suppose that there exist two authorizations a_i and a_j for a privilege on a table that conflict with respect to a subject s. Thus, $p(a_i) = p(a_j)$ and $t(a_i) = t(a_j)$, but $pt(a_i) \neq pt(a_j)$. There are three cases to consider.

Case 1: $s(a_i) \neq s(a_j)$. Clearly, $s \neq s(a_i)$ and $s \neq s(a_j)$ (since otherwise a_i and a_j will not conflict with respect to s). The conflict between a_i and a_j with respect to subject s can be resolved by inserting another authorization a_m with $p(a_m) = p(a_i)$, $t(a_m) = t(a_i)$, $s(a_m) = s$, and either $pt(a_m) = +$ or $pt(a_m) = -$, depending on whether the grantor wishes to give or deny the privilege to subject s.

Consider the authorization state of Figure 4. Conflict $\langle \texttt{Employees}, \texttt{sel}, +, \texttt{T}_1 \rangle \diamondsuit_{\texttt{Researchers}} \langle \texttt{Soft-developers}, \texttt{sel}, -, \texttt{T}_1 \rangle$ can be resolved by specifying an authorization (either positive or negative) for the `select` privilege on table $\texttt{T}_1$ for `Researchers`. Note that the insertion of this authorization resolves also the conflict between the two authorizations with respect to all the members of `Researchers`.

Case 2: $s(a_i) = s(a_j)$, *but* $s \neq s(a_i)$. The conflict can once again be resolved by inserting another authorization a_m as given in Case 1 above. For instance, with reference to the authorization state of Figure 4, conflict



Legend: bold face indicates strong authorizations

Fig. 4. An example of an authorization state.

$\langle \texttt{ConsC}, \texttt{sel}, +, \texttt{T}_2 \rangle \diamondsuit_{\texttt{Sam}} \langle \texttt{ConsC}, \texttt{sel}, -, \texttt{T}_2 \rangle$ can be resolved by specifying an authorization (either positive or negative) for the `select` privilege on table $\texttt{T}_2$ for `Sam`.

Case 3: $s(a_i) = s(a_j) = s$. In this case, the conflict can only be resolved by either deleting one of the two authorizations or by inserting a strong authorization a with s as the subject and with the same privilege and table as the two conflicting authorizations. Note, however, that the insertion of the strong authorization may not always be possible, since it may result in an inconsistent authorization state. For instance, consider the authorization state of Figure 4. Conflict $\langle \texttt{Res1}, \texttt{sel}, +, \texttt{T}_3 \rangle \diamondsuit_{\texttt{Res1}} \langle \texttt{Res1}, \texttt{sel}, -, \texttt{T}_3 \rangle$ can be resolved only by deleting any of the two authorizations. The insertion of any strong authorization for the select privilege on table $\texttt{T}_3$ for subject `Res1` would conflict with either the authorization of `Carol` or the authorization of `Frank`. We claim that it is proper to require that one of the two authorizations be deleted in order to resolve the conflict, since the simultaneous presence of a negative and a positive authorization specified for the same subject is intrinsically ambiguous.

6. AUTOMATIC DETECTION OF CONFLICTS

In the previous section we have illustrated how conflicts among weak authorizations can be easily resolved by adding new, more specific authorizations. The number of subjects and the relationships among them, however, may make this task not easy. As a matter of fact, resolving conflicts requires users to be informed of the existing authorizations, as well as of the relationship among the subjects in them. To be feasible, this task must

therefore be supported by appropriate tools that can inform the users of the conflicts introduced upon execution of a given operation and their possible solutions. In this section we illustrate two algorithms (whose complete descriptions and commentaries can be found in Bertino et al. [1996b]) for the automatic detection of conflicts introduced upon the insertion and deletion of weak authorizations on base tables. The reason why only authorizations on base tables are considered is, that, according to Definitions 2 and 4, conflicts among weak authorizations can exist only for authorizations on base tables. Algorithms for detecting conflicts introduced by the addition or the removal of members from groups are analogous. Users can then resolve the conflicts communicated by the detection algorithms by specifying appropriate authorizations (negative or positive, according to how they would like the conflicts to be resolved) for the considered privilege and table on the subjects over which conflicts are returned. (Note that the insertion of the authorizations resolving conflicts must also be controlled for possible conflicts.)

The algorithm for finding conflicts introduced by the insertion of a new weak authorization works as follows. Suppose a new weak authorization a is granted. If the subject or any of the groups to which the subject belongs owns a strong authorization for the privilege on the table, authorization a is overridden, and therefore no conflict is introduced. Otherwise, if the subject owns some weak authorizations for the privilege on the table with privilege type different from a, the conflicts of these authorizations with a over the subject are returned, and the process terminates. If no such authorizations exist, no conflict over the subject is introduced. However, a could conflict over the subject's members with the authorizations specified for some groups to which the members belong. To find these conflicts, the following process is executed for each direct member s of $s(a)$. If s owns an authorization for the privilege on the table or if any of the groups to which s belongs owns a strong authorization for the privilege on the table, then a is overridden for s, and therefore no other control need to be executed for s or for its members. Otherwise, if no such authorizations exist, all the membership paths from s to the groups to which s belongs are traversed. The traversal of a membership path is stopped when either an authorization, positive or negative, is found or when there are no more groups to be considered on it. Hence, if some authorizations with a privilege type different from a have been found, the conflict of these authorizations with a over s is returned. Otherwise, no conflict is originated over s. Hence, since a could conflict over s's members, the process is recursively repeated for each member of s.

Example 12. Consider the authorization state of Figure 4 and the request by `Luke` to grant a weak positive authorization for the select privilege on table T_4 to `Employees`. The following conflicts would be returned by the algorithm:

$\langle \texttt{Employee}, \texttt{sel}, +, \texttt{T}_4 \rangle \ \diamond_{\texttt{Bill}} \ \langle \texttt{Non-citizen}, \texttt{sel}, -, \texttt{T}_4 \rangle$

$\langle \texttt{Employee}, \texttt{sel}, +, \texttt{T}_4 \rangle \ \diamond_{\texttt{Researchers}} \ \langle \texttt{Soft-developers}, \texttt{sel}, -, \texttt{T}_4 \rangle$

The algorithm for controlling conflicts introduced by the removal of a weak authorization works as follows. Suppose a weak authorization a is removed. If the subject owns an authorization for the privilege on the table or if any of the groups to which it belongs owns a strong authorization for it, then the removal of a cannot generate any conflict, and therefore the process terminates. Otherwise, the subject does not have any other authorizations, and therefore the weak authorizations of the groups to which it belongs, which first were overridden by a, now become applicable to the subject and, possibly, to its members. These authorizations can therefore potentially create conflicts over s or its members. To find these authorizations, the groups to which $s(a)$ belongs are checked by traversing all membership paths from $s(a)$ to the root and stopping over a membership path whenever an authorization for the privilege on the table is found. If both positive and negative authorizations are found, the conflicts between each positive authorization and each negative authorization found over s are returned, and the process terminates. If no authorizations were found or if all the authorizations found have the same privilege type as a, no conflict has been introduced, and the process terminates. Otherwise, all authorizations found have a privilege type different from a, and the fact that they are now applicable to the subject's members may introduce possible conflicts. Hence, the authorizations applicable to the subject's members must be controlled. Let us refer to the set of the authorizations found for the groups to which $s(a)$ belongs as *confl_auth*. Then, each member of the subject is considered, and the existence of authorizations with a privilege type different from that of the authorizations in *confl_auth* specified for other groups to which the member belongs and applicable to the member is checked. For each member for which such authorizations are found, the conflict of each of such authorizations with each of the authorizations in *confl_auth* is returned. The process is also recursively repeated for each member for each member for which none of such authorizations have been found.

Example 13. Consider the authorization state of Figure 4 and the request by `Luke` to revoke the weak positive authorization for the select privilege on table $\texttt{T}_5$ from `Researchers`. The following conflicts would be returned by the algorithm:

$\langle \texttt{Employee}, \texttt{sel}, -, \texttt{T}_5 \rangle \ \diamond_{\texttt{Tim}} \ \langle \texttt{Consultants}, \texttt{sel}, +, \texttt{T}_5 \rangle$

$\langle \texttt{Employee}, \texttt{sel}, -, \texttt{T}_5 \rangle \ \diamond_{\texttt{David}} \ \langle \texttt{Res1}, \texttt{sel}, +, \texttt{T}_5 \rangle$

7. EXPRESSIVENESS OF THE MODEL

In this section we illustrate how our authorization model can be used to mirror the following policies: the traditional closed and open policies, the

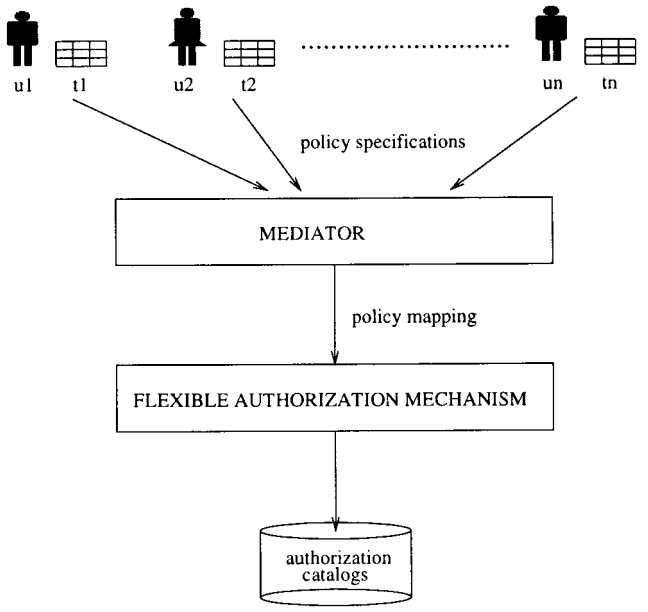


Fig. 5. The system architecture.

closed policy with negative authorizations and conflict resolution based on the denials-take-precedence principle, and the closed policy with negative authorizations and conflict resolution based on the most-specific-authorization-takes-precedence principle. (Note that the last policy is the one actually enforced by our model.) We also show, by means of an example, how the model can represent different protection requirements users may need to express on their data. Here, we assume an architecture such as that illustrated in Figure 5, based on a mechanism implementing our authorization model and on a mediator interfacing the users and the authorization model and enforcing the policy mapping.

The implementation of the closed policy is straightforward. When only positive authorizations are used, our model reduces to the traditional closed-policy approach. It is the task of the mediator to ensure that only positive authorizations can be specified. The mediator rejects any attempts by the users to specify negative authorizations.

The open policy can be easily represented by requiring the mediator to ensure the satisfaction of the following conditions. For every object, a weak positive authorization is specified for the group representing the root of the graph.[8] Users will then be allowed to specify only strong negative authorizations. Since a strong negative authorization overrides the weak positive authorization in our model, the result is an open policy where all accesses

[8]If the membership graph is not single rooted, a group, called `public`, containing all subjects in the system can be inserted.

for which users do not specify negative authorizations are allowed. Note that current DBMSes have a group, called public, containing all the users of the system, to which authorizations can be granted; however, since negative authorizations are not supported, no exceptions to the authorizations granted to the group public can be enforced.

The denials-take-precedence policy can be implemented by the mediator by ensuring that all positive authorizations are weak, while all negative authorizations are strong, so that negative authorizations will always override positive authorizations.

A major advantage of using our framework is that all users are not constrained to the use of a single policy. They can choose to apply different policies on different relations, as illustrated by the following example.

Example 14. Consider the group membership graph illustrated in Figure 1 and a user Luke who creates the following tables with their different protection requirements:

—Public_info: everybody is to be allowed access. This can be accomplished by granting a strong positive authorization to Users.

—Nat_pub_info: everybody who is a citizen has access. A weak positive authorization can be granted to Users and a strong negative authorization granted to Non-citizens.

—Internal_report: everybody can access unless explicitly denied. A weak positive authorization is granted to Users. Negations can be specified at any time.

—Organization_info: Luke wishes to retain control of all access authorizations on the relation except for the select privilege, which can be administered by the members of the group Staff. Moreover, Luke does not want Matt to be able to read information in the relation. To this end, Luke grants the weak adm-access for the select privilege on the relation to Staff and grants a negative strong authorization for the select privilege on the relation to Matt.

—Fundings: it contains all information about fundings. Those fundings that have been already approved can be accessed by people explicitly authorized. Access authorizations are to be administered by Edith. Luke creates a view Appr-fundings defined as "select * from Fundings where status = approved." Edith is granted the adm-access privilege on Appr-fundings.

—Software_project: contains information about software projects. People involved in software development should be authorized. A weak positive authorization is specified for Soft-developers.

—Software_project: temporarily contains some private information not yet to be released. Consultants must be temporarily forbidden access to the relation. A negative (strong or weak) authorization can be specified for the select privilege on the relation for Consultants. There is no

need to revoke the authorization previously granted, since it will once again become valid upon revocation of the denial.

We therefore believe that users can find in our model a flexible framework in which protection requirements can be expressed in a way that best suits their needs.

8. IMPLEMENTATION

A prototype of the authorization system described in this article has been implemented. In the implementation, the authorization catalogs of a commercial DBMS have been simulated and have been extended to support advanced features of our model. Moreover, tools to support authorization management have been developed.

The current version of the prototype has been developed on top of the Informix relational DBMS. The access control functions are written in ESQL/C. The graphical interface for the tools has been implemented using OFS/MOTIF libraries for X Windows.

In the following, we briefly discuss the two main components of the prototype, namely, the authorization catalog manager and the authorization administration environment.

8.1 Authorization Catalog Manager

A critical aspect in the authorization catalog manager is the catalog structure. In our prototype, we have followed the approach, common to many relational DBMSes, of implementing the catalogs as relations. The entire authorization system uses the following four catalogs:

NSYSTABAUTH:	it stores for each table/view, the access privileges granted on the table/view.
NSYSSUBJECT:	it stores all subjects authorized to connect to the database and any special privilege the subjects may have such as DBA or Resource.
NSYSREACHABLE:	it records, for each subject, the groups to which the subject belongs, either directly or indirectly.
NSYSGROUPS:	it records for each subject, the groups to which the subject directly belongs.

The structure of those catalogs is similar to the structure of authorization catalogs of many commercial relational DBMSes with some differences due to the peculiarities of our authorization model.

Note that the information contained in the NSYSGROUPS could be derived from the NSYSREACHABLE catalog, and vice versa. The tables are both maintained to efficiently retrieve the different information on group membership. In particular, when only direct membership must be retrieved, access to NSYSGROUPS is more efficient, since the number of tuples in it is considerably smaller than the number of tuples in NSYS-

Table II. Field of Table NSYSTABAUTH and Their Semantics

Field Name	Field Description
grantee	subject receiver of the authorization
table-id	internal identifier of the table/view
auth-descr	authorization descriptor
grantor	subject who granted the authorization
date	when the authorization was granted
time	instant at which the authorization was granted

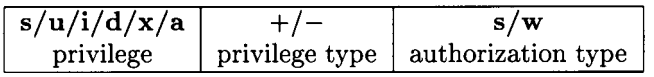


Fig. 6. Field auth-descr of table NSYSTABAUTH.

REACHABLE. This table is therefore used in the access control process when evaluating weak authorizations (function *weak_auth*). Table NSYS-REACHABLE is instead used to efficiently retrieve, with a single access, all the groups to which a subject belongs, either directly or indirectly. This table is used in the access control process when evaluating strong authorizations (function *strong_auth*), where all the groups to which a subject belongs must be considered, regardless of the path through which they are reachable.

Table II shows the structure of the NSYSTABAUTH catalog. We do not show here the structure of the other catalogs and refer the reader to Loretti [1996] for this. Among the fields of NSYSTABAUTH, the AUTHORIZATION field is particularly important because it encodes, beside the privilege, the sign (positive or negative) and the type (strong or weak) of the authorization. This field is organized as a three-byte pattern, structured as illustrated in Figure 6.

8.2 Authorization Administration Environment

This environment consists of a number of tools supporting the task of managing authorizations. Basically, all commands for registering users and granting and revoking authorizations can be performed using some high-level tools, rather than SQL commands. The tools are mainly useful for visualizing the effects of grant and revoke commands and for illustrating the group structure.

These tools represent the most innovative aspect of our implementation. When dealing with authorization exceptions, the semantics of the authorization model becomes more complex, and therefore it may be difficult to properly administrate authorizations. The tools we have developed provide explanation facilities as well as visualization and exploration mechanisms which allow the user to view the effects of administrative operations on the authorization state. Explanation facilities provide users with information about the effects of an operation on the users' privileges or about why a given administrative operation cannot be executed. For example, if an

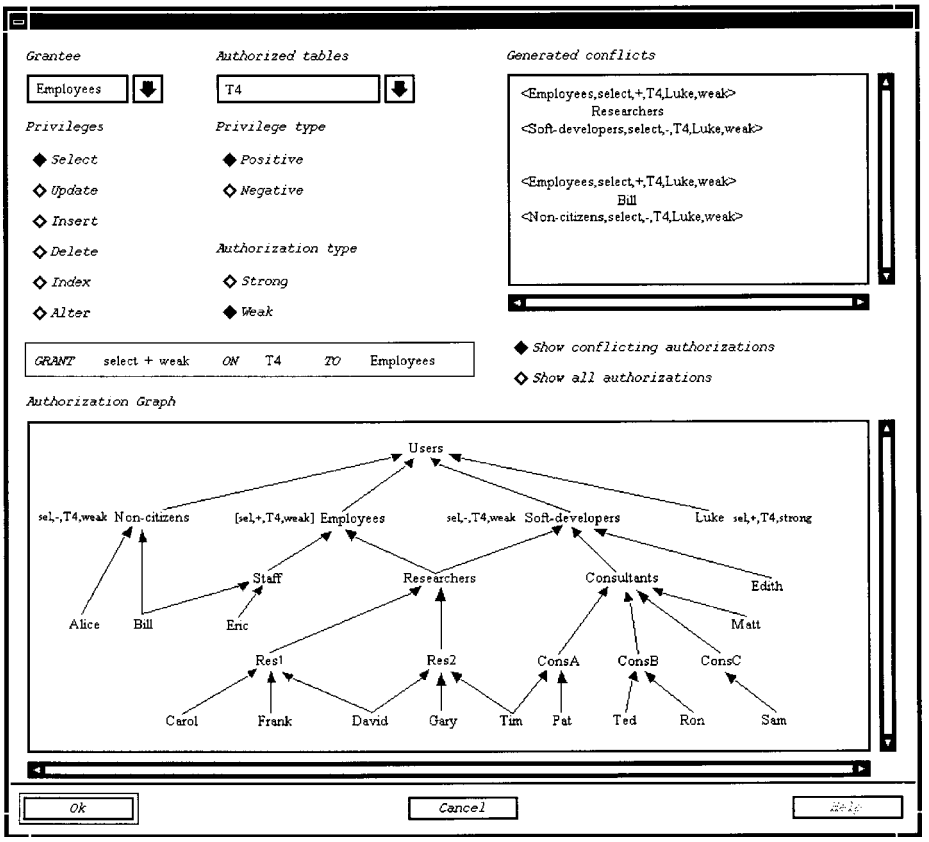


Fig. 7. An example of a "grant authorization" dialog.

administrative request cannot be executed because it would produce an inconsistent authorization state, the reasons for the rejection (i.e., the conflicts which would have been introduced) are returned to the user. To the best of our knowledge, current commercial DBMSes do not provide any advanced support for authorization management. For example, no graphical tools are provided in available DBMSes for visualizing the state of the authorization base and the relationships among the authorizations in it.

As an example, Figure 7 illustrates the window shown to the user for the execution of the grant command reported in Example 12. When the grant command is called, the window is visualized on the screen with all areas in it empty, but for the one titled "authorization graph," which contains the graphical representation of the group membership graph. The user then specifies, by writing in the appropriate area, or clicking on the appropriate selection button, the grantee to which the authorization is to be given, the table on which the authorization is specified, the privilege being granted, and the privilege and authorization type. When all the parameters are specified, the command area summarizes the grant command. Possible conflicts introduced by the execution of the grant command are listed in the

area titled "Generated Conflicts." Conflicting authorizations are visualized in the group membership graph. The authorization whose insertion has been required is written between square brackets to distinguish it from the other authorizations in the graph.

9. RELATED WORK

Early authorization models [Castano et al. 1995] only allowed the specification of positive authorizations. More recent authorization models for operating systems and database systems also permit specification of negative authorizations stating accesses to be denied. As for operating systems, an authorization model supporting positive as well as negative authorizations has been proposed in the context of the Andrew File System [Satyanarayanan 1989]. Subjects of the authorizations can be users and groups of users. A group is a set of groups and users and is associated with an owner who is allowed to add and remove members from the group. A user operates with the union of his or her authorizations and that of the groups to which he or she belongs. A user can temporarily disable personal membership to some groups. Authorizations of disabled groups are not available to the user. In case of the simultaneous presence of negative and positive authorizations, the model adopts the denials-take-precedence policy. The model provides a limited form of authorization administration, where only the owner of a file can grant and revoke authorizations to other subjects. Moreover, since objects of the authorizations can only be files, the problem of authorizations on derived objects does not arise.

In the database systems context, Bertino et al. [1997] extend the authorization model proposed in the framework of System R [Griffiths and Wade 1976] to include negative authorizations. Authorizations can be specified for both single users and user groups. Negative authorizations can be specified with reference to specific privileges on tables, and the denials-take-precedence policy is used for resolving conflicts among authorizations. No administrative policy is provided.

A more flexible authorization model supporting both positive and negative authorizations has been developed at SRI in the context of the SeaView project [Lunt 1989; Lunt et al. 1989]. The SeaView model supports negative authorizations by introducing a special access mode, called "null." Granting the null privilege on an object to a user means denying the user *all* accesses on the object. Thus, if a user is given a null privilege on an object, the user will not be able to access the object, even if he or she owns an authorization for the access. The administration of privileges is controlled through the access modes "grant" and "give-grant." If a user has the grant access mode on a table, he or she can grant and revoke any access mode (including null) on the table from other subjects in the system. If a user has the give-grant access mode on a table, he or she can additionally grant and revoke the grant and give-grant access modes on the table from other subjects in the system. Although the SeaView model permits authorizations to be specified for single users as well as groups of users, a subject operating on behalf of

a user is constrained to the privileges owned by at most one group to which the user belongs. Conflicts among authorizations are resolved on the basis of the following policy: (1) authorizations specified for a user take precedence over authorizations specified for the groups to which the user belongs and (2) the authorization for the null access mode specified for a user (group) takes precedence over any other authorization specified for the same user (group). The SeaView model suffers from several limitations. First, the model does not support negative authorization at the level of a single access mode. Thus, for example, it is not possible to state that a user is authorized for the select access mode, but denied for the insert access mode on an object. Second, administrative authorizations are referred to all privileges executable on an object and not to single privileges. Thus, it is not possible to give a user the authorization to administer a specific privilege (e.g., select) on a table. Third, only users can belong to groups. Fourth, authorizations specified for groups are considered only when no authorizations are explicitly specified for the user. Hence, if we would like to give a user some authorization in addition to the authorization he or she has as a member of some groups, we need to respecify the authorizations of the groups for the user himself. Moreover, the SeaView model allows a user to exercise an authorization on an object specified for a group to which the user belongs even if the user belongs to another group which has been explicitly given the null privilege for the object.

Another model supporting negative authorization has been proposed in the context of object-oriented systems in the framework of the ORION/ITASCA project [Rabitti et al. 1991]. Authorizations can be specified only for "roles" (which are groups of users) and not for single users. The model, which supports both positive and negative authorizations, introduces the concept of strong and weak authorizations. Weak authorizations can be overridden whereas strong authorizations cannot. The model enforces derivation of additional authorizations, called implicit, from the authorizations explicitly specified by the users. Resolution of possible conflicts between positive and negative authorizations is based on the concept of more specific authorization. However, the Orion authorization model suffers from several limitations and drawbacks (see Section 9.1).

Brüggemann [1992] proposes a model for the protection of object-oriented databases, based on the concepts introduced in Orion, where authorization conflicts are resolved based on explicit integer-valued priorities between authorizations.

Another model supporting both negative and positive authorizations for protection in object-oriented systems has been proposed by Gal-Oz et al. [1993]. Authorizations specified on a class propagate to its subclasses. Conflicts among positive and negative authorizations are resolved by considering explicit authorizations as prevailing over implicit authorizations together with the denials-take-precedence policy. This model does not consider groups of users; authorizations can be specified for single users only.

Shen and Dewan [1992] propose an authorization model supporting both positive and negative authorizations for the protection of information in collaborative environments. Authorizations are specified for roles which can be dynamically taken and released by users. Privileges and objects can be specified as sets of other privileges and objects, respectively. In this way, a single authorization can be specified which allows a role to exercise a set of privileges on a set of objects. Resolution of conflicts between authorizations is based on the concept of more specific authorization. An authorization a is considered more specific than an authorization a' if the subject of a is a member of the subject of a'. When no resolution policy can be applied, since none of the subjects in the authorizations a and a' is a member of the other, the access decision is based on an explicit precedence relationship. The precedence relationship is based on the assumption that authorizations are specified as ordered ACLs attached to objects. When the conflicts between two authorizations cannot be solved on the basis of the most specific rule, the authorization that appears first in the ACL wins over the other. We note that the conflict resolution based on the inheritance hierarchy is similar to that provided by us for weak authorizations. Notice, however, that the overriding policy of Shen and Dewan is not exactly the same as ours. In our model, authorization a overrides a' only along the membership path to the subject of a' which passes through the subject of a (see Section 3). A major difference between our model and the model by Shen and Dewan concerns the way conflicts among authorizations are solved when the subjects in the conflicting authorizations are not related in the hierarchy. Our model assumes that no decision can be taken and therefore takes the safest solution of denying the access. The model by Shen and Dewan, instead, relies on explicit priorities. The idea of explicit priorities is interesting and presents some advantages. In particular, the model by Shen and Dewan, in some cases, is more flexible than ours, since the consideration of explicit priorities may avoid the strict application of the denials as default. However, this approach also presents some drawbacks. A first problem concerns the specification of authorizations. In the model by Shen and Dewan administrative privileges are not mentioned. However, we note that decentralized administration would create some problems in such a model. Since ACLs are position sensitive, users who specify the authorizations should be given the possibility of specifying where in the ACL a new authorization must be positioned. In particular, a user could insert a new authorization before other existing authorizations, thus giving to the new authorization higher priority. Then a user who specifies an authorization cannot make sure that the system will definitely obey it (like in the case of our strong authorizations). If insertions in the ACLs are not controlled, the case could also be where authorizations specified by some delegated users override those specified by the owner, despite the willingnesss of the owner for that. In our model, instead, the owner of an object can always keep control on the accesses to his or her objects by specifying strong authorizations and by delegating only weak

Table III. Comparison of Approaches with Respect to the Requirements Stated in Section 1

	Requirements				
Model	Pos. and Neg. Auth.	Exceptions/ Strong Enforc.	Admin.	Delegation/Control Retainment	Subject Grouping
Andrew [Satyanarayanan 1989]	yes	limited/no	yes	no/not appl	groups (dynamic)
Bertino et al. [1997]	yes	limited/no	no	not appl/not appl	groups
Sea-View [Lunt 1989; Lunt et al. 1989]	limited	yes/no	yes	yes/no	groups (not nested)
Orion [Rabitti et al. 1991]	yes	yes/yes	yes	yes/no	roles
Brüggemann [1992]	yes	yes/yes	no	not appl/not appl	roles
Gal-Oz et al. [1993]	yes	limited/no	no	not appl/not appl	no
Shen and Dewan [1992]	yes	yes/limited	no	not appl/not appl	roles
Our approach	yes	yes/yes	yes	yes/yes	groups

`adm-access` privileges. The owner can have authorizations overridden, if he or she wishes so, if the authorizations are specified as being weak. Another drawback of the model by Shen and Dewan is that the order among authorizations is taken into consideration only if the subjects in the authorizations are not related. Then, the more specific relationship is always applied first. This implies that authorizations of a group will always be overridden by the authorizations of its members. It is therefore impossible to ensure that an authorization specified for a group will always be obeyed (it is instead possible in our model by specifying the authorization for the group as strong). As for the capability of representing the classical access control policies, it is true that putting all the negative authorizations before the positive ones in ACLs enforces denials-take-precedence. Likewise, putting all the positive authorizations before the negative ones in ACLs enforces permissions-take-precedence. However, again, these policies apply only when the two conflicting authorizations have subjects such that none of them is a member of the other. Therefore, they cannot be enforced properly.

Table III summarizes how the different models discussed in this section address the requirements stated in Section 1. A "yes" value indicates that the considered model addresses the requirement; a "no" value indicates that the model does not address the requirement; a "limited" value indicates that the model addresses the requirement only partially; and a "not appl" value indicates that the requirement is not applicable. (For instance, it is not meaningful to talk about delegation for a model which does not support any administrative policy to talk about control retainment for a model which does not support delegation.)

9.1 Comparison with Orion

The concept of strong and weak authorizations used in our model has been first introduced in the Orion authorization model [Rabitti et al. 1991]. However, our model has a number of important differences from the

authorization model of Orion. These differences can be summarized as follows:

—In the Orion authorization model, positive authorizations propagate from more general to more specific subjects, and negative authorizations propagate from more specific to more general subjects. For instance, with reference to our group configuration graph, negative authorizations specified for group `Employees` would not be propagated to `Staff` and `Researchers` but to the group `Users`. Then, *it is not possible to specify negative authorizations for a group which propagate to the members of the group.* In our view, *this is not consistent with the concept of user groups* and is due to the fact that the semantics of user groups and user roles is mixed in Orion. This difference between the Orion model and our model is very important in that it implies different concepts of more specific authorizations and overriding relationships.

—In the Orion authorization model *it is not possible to specify authorizations for single users,* but only for groups of users. A user operates with union of the privileges of all the groups to which he or she belongs. Moreover, the relationship between users and groups is not taken into account in the model when evaluating the consistency of the authorization state. Hence, although the model guarantees absence of conflicts among the authorizations of each single group, a user may belong to groups with conflicting authorizations. For instance, with reference to our group membership graph, the situation in which `Non-citizens` are strongly denied an access and `Employees` are strongly authorized for the same access will be allowed (although the groups intersect). In evaluating the access requests, positive authorizations take precedence over negative authorizations. A user will then be allowed to exercise an access even though he or she belongs to a group with a strong negative authorization. With reference to our example, `Bill` (who is both an `Employees` and a `Non-citizens`) will be allowed to exercise the access, thus not obeying the strong requirement that `Non-citizens` must not be allowed for the access. Then, it is not possible to really enforce strong authorizations at the level of single users. In our model, authorizations can be specified also for single users. The relationship between users and groups is taken into consideration to ensure that no users holds conflicting valid authorizations.

—The Orion authorization model requires the authorization base to be complete: for every possible access request an authorization, either positive or negative, strong or weak, implicit or explicit, must exist. This approach *requires users to specify all possible authorizations and forces the use of negative authorizations to represent cases where the authorization is simply not to be given rather than to support exceptions or denials.* We do not require the authorization state to be complete. We assume a closed-world policy: if no authorization is specified for a given access, the access is denied.

—The Orion authorization model *does not provide any administrative policy*. In the model, the assumption is made that every user who is authorized for an access can also grant this access to other users. This approach has the drawback that granting a subject the privilege to exercise an access on an object implies granting him or her the privilege to administer the access, i.e., to grant and revoke authorizations for the access to other subjects. In this approach, it would therefore be impossible for the creator of an object to maintain control of who can access, and how, the created objects. In our model, we provide an administrative policy based on the ownership approach which supports different types of administration.

—In the Orion authorization model *no conflicts, neither among strong nor among weak authorizations, are allowed*. Each operation producing a conflict in the authorization state is therefore rejected. By contrast we allow conflicts between weak authorizations and handle negative authorizations as blocking authorizations. If two weak authorizations conflict, the positive authorization is retained in the authorization base even though it is not valid any longer. It will become valid again once the conflicting negative authorization is revoked. This approach has the great advantage of providing temporary suspensions of authorizations [Bertino et al. 1996a]. If conflicts were not allowed, the positive authorization would have to be revoked and then granted again later on. This characteristic of our model represents a real advantage, since the revocation of an authorization can have disruptive effects such as the dropping of views and revocation of other authorizations [Bertino et al. 1997]. If the revoked authorization is granted again later on, the views must be redefined and the other authorizations granted again. With our approach, views and authorizations are preserved, although temporarily invalidated. They become valid again when the negative authorization is revoked.

—In the Orion authorization model *no algorithm is proposed for finding or resolving conflicts among authorizations*. In our model we allow conflicts among weak authorizations and apply a denials-take-precedence policy in case of conflicts. In this way, users can temporarily live with conflicts, resolving them according to their needs as we have illustrated in Section 5. Moreover, we have given algorithms for the detection of conflicts. As for conflicts among strong authorizations these algorithms provide the users with the explanation as to why the operation introducing the conflicts is denied. As for weak authorizations, the algorithms provide the users with the list of all the conflicts introduced by the administrative operation requested. This information is then used by the users for the specification of authorizations resolving the conflicts.

—The Orion model *does not address authorization issues with respect to derived objects, like views*. The problem of assigning authorizations to the creator of an object is never discussed. The tacit assumptions seem to be

taken that the user creating an object can exercise all the privileges on the object. However, it is not clear how this is performed, since ownership is never mentioned and since authorizations can be specified only for roles.

10. CONCLUSIONS AND FUTURE WORK

In this article, we have presented an authorization model that can be used to express different policies (open versus closed policies, and denials-take-precedence versus exceptions-take-precedence) for different data in the same database. The model permits both positive and negative authorizations and supports exceptions at the same time. The model is flexible in that the users can specify, for each authorization they grant, whether the authorization can allow for exceptions or whether it must be strongly obeyed. It provides authorization management for groups with exceptions at any level of the group hierarchy and temporary suspension of authorizations. The model supports ownership together with decentralized administration of authorizations. Administrative privileges can also be restricted so that owners retain control over their tables.

Although we couch our authorization model in terms of relational DBMSes, it can be adapted to other data management systems as well. The features provided in our model are relevant not only in the context of advanced applications and cooperative environments, but also in federated systems, where different policies need to be represented in a common authorization model. Issues concerning authorization policies in federated systems have not been adequately addressed so far. We believe that our model can find a wide application scope in those systems.

ACKNOWLEDGMENTS

The authors wish to thank associate editor Prasun Dewan and the anonymous referees for their detailed and insightful comments.

REFERENCES

ABADI, M., BURROWS, M., LAMPSON, B., AND PLOTKIN, G. 1993. A calculus for access control in distributed systems. *ACM Trans. Program. Lang. Syst. 15*, 4 (Sept.), 706–734.

BERTINO, E., BETTINI, C., FERRARI, E., AND SAMARATI, P. 1996a. A temporal access control mechanism for database systems. *IEEE Trans. Knowl. Data Eng. 8*, 1 (Feb.).

BERTINO, E., JAJODIA, S., AND SAMARATI, P. 1996b. A flexible authorization mechanism for relational data management systems. Tech. Rep. Computer Science Department, Università di Milano, Milan, Italy.

BERTINO, E., JAJODIA, S., AND SAMARATI, P. 1996c. Supporting multiple access control policies in database systems. In *Proceedings of the IEEE Symposium on Security and Privacy* (Oakland, CA, May). IEEE Press, Piscataway, NJ.

BERTINO, E., SAMARATI, P., AND JAJODIA, S. 1997. An extended authorization model for relational databases. *IEEE Trans. Knowl. Data Eng. 9*, 1 (Jan.-Feb.), 85–101.

BRÜGGEMANN, H. H. 1992. Rights in an object-oriented environment. In *Database Security V, Status and Prospects*, C. Landwehr and S. Jajodia, Eds. Elsevier North-Holland, Inc., New York, NY.

CASTANO, S., FUGINI, M. G., MARTELLA, G., AND SAMARATI, P. 1995. *Database Security*. ACM Press/Addison-Wesley Publ. Co., New York, NY.

FAGIN, R. 1978. On an authorization mechanism. *ACM Trans. Database Syst. 3*, 3, 310–319.

GAGLIARDI, R., LAPIS, G., AND LINDSAY, B. 1989. A flexible and efficient database authorization facility. Tech. Rep. RJ 6826(65360). IBM Almaden Research Center.

GAL-OZ, N., GUDES, E., AND FERNANDEZ, E. B. 1993. A model of methods authorization in object-oriented databases. In *Proceedings of the International Conference on Very Large Data Bases* (Dublin, Ireland). Morgan Kaufmann Publishers Inc., San Francisco, CA.

GRIFFITHS, P. G. AND WADE, B. 1976. An authorization mechanism for a relational database system. *ACM Trans. Database Syst. 1*, 3 (Sept.), 243–255.

INFORMIX. 1993. *Informix-Online/Secure. Security Features User's Guide*. Informix Software, Inc.

JAJODIA, S., SAMARATI, P., AND SUBRAHMANIAN, V. S. 1997. A logical language for expressing authorizations. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy* (Oakland, CA, MAY). IEEE Press, Piscataway, NJ, 31–42.

LORETTI, S. 1996. Flexauth system—User manual. Computer Science Department, Università di Milano, Milan, Italy.

LUNT, T. F. 1989. Access control policies for database systems. In *Database Security II: Status and Prospects*, C. E. Landwehr, Ed. North-Holland Publishing Co., Amsterdam, The Netherlands, 41–52.

LUNT, T. F., DENNING, D. E., SCHELL, R. R., HECKMAN, M., AND SHOCKLY, W. R. 1989. Secure distributed data views. Tech. Rep. Computer Science Laboratory, SRI International, Menlo Park, CA. Volumes 1–4.

MELTON, J. 1990. ISO/ANSI working draft—Database language sql2. Tech. Rep. ANSI X3H2-90-309. ANSI, New York, NY.

RABITTI, F., BERTINO, E., KIM, W., AND WOELK, D. 1991. A model of authorization for next-generation database systems. *ACM Trans. Database Syst. 16*, 1 (Mar.), 88–131.

SATYANARAYANAN, M. 1989. Integrating security in a large distributed system. *ACM Trans. Comput. Syst. 7*, 3 (Aug.), 247–280.

SELINGER, P. G. 1990. Authorizations and views. In *Distributed Data Bases*, I. W. Draffan and F. Pooe, Eds. Cambridge University Press, New York, NY.

SHEN, H. AND DEWAN, P. 1992. Access control for collaborative environments. In *Proceedings of the International Conference on Computer Supported Cooperative Work*. ACM Press, New York, NY, 51–58.

Received: September 1994; revised: August 1996, September 1996, January 1997, and December 1997; accepted: December 1997