A Temporal Authorization Model *

Elisa Bertino Claudio Bettini Pierangela Samarati

Dipartimento di Scienze dell'Informazione
Università di Milano
via Comelico 39/41 Milano 20135, Italy
Phone: (+39) 2-55006227
Fax: (+39) 2-55006253
{ebertino,bettini,samarati}@dsi.unimi.it

Abstract

This paper presents a discretionary access control model in which authorizations contain temporal information. This information can be used to specify temporal intervals of validity for authorizations and temporal dependencies among authorizations. A formal definition of those concepts is presented in the paper, in terms of their interpretation in first order logic. We characterize sets of temporal dependencies that can lead to undesirable states of the authorization system and we sketch an algorithm for their detection. Finally, operations to add, remove, or modify authorizations and temporal dependencies are described.

1 Introduction

The area of database security has been recently a fast growing area. Indeed, there is today an increased awareness of the importance of data protection from unauthorized accesses and from various types of intrusions, such as those through Trojan Horses or covert channels [7]. Due to the recent advancements in the area of secure DBMS during the past years, several vendors of relational DBMS (RDBMS) now have secure versions of their products which are commercially available. Therefore, DBMS security is a research area which has a tangible impact on commercial product development. In particular, most commercial DBMS provide at least an authorization mechanism, by using which users of the database may be granted authorizations to read and/or write data.

Current research directions in DBMS security can be summarized as follows. A first direction concerns increasing the expressive power of authorization models and developing the appropriate tools and mechanisms to support those models. An example of this direction is the introduction of negative authorization [6], and of role-based and task-based authorization models [8, 16, 20]. The increased awareness of the need for data protection and the complexity of new applications have resulted in more articulated authorization policies. Those policies need models and systems directly able to support them. Therefore, it is crucial to develop powerful authorization mechanisms which provide, at the same time, concise languages and adequate tools for their use.

A second direction concerns the development of authorization models for advanced DBMS, like object-oriented DBMS [15, 11, 19] or active DBMS. Those DBMS are characterized by richer data models, including notions such as inheritance hierarchies, composite objects, versions, and methods. Therefore, authorization models first developed for RDBMS [12] must be properly extended to include features related to those additional modeling concepts [5].

A third direction, generally known as mandatory access control, is to develop techniques to protect data against intrusion through sophisticated means, like Trojan Horses and covert channels. Several results have been so far reported for RDBMS [3, 9, 10, 17] and applied to commercial products [14]. Research has also started dealing with advanced DBMS, like OODBMS [22, 4, 13] and active DBMS [18].

The work reported in this paper deals with the first of the above research directions. The goal of our work is to extend the expressive power of a conventional authorization model with temporal information. The need for extending authorizations to the consideration

*The work reported in this paper was partially supported by CNR under Grant no. 94.00450.CT12, by NATO under Collaborative Research Grant no. 930888, and by the Italian M.U.R.S.T.

CCS '94- 11/94 Fairfax Va., USA

of temporal semantics was also pointed out by Thomas and Sandhu in [21].

Our temporal extension to authorization models is based on two main concepts. The first concept is the temporal interval of validity of authorizations. In our model it is possible to specify that an authorization will expire after a specified point in time, or that an authorization is valid only in a specified temporal interval. Note that in many real-life situations authorizations are limited in time - consider for example a badge which is valid for only one day. Therefore, it is important to develop authorization models able to directly support those application requirements. The second concept is the temporal dependency among authorizations. A temporal dependency can be used, for example, to specify that a user has an authorization as long as another user has this authorization. This type of capabilities is very useful in many advanced applications, like for example CSCW applications.

Besides proposing a basic set of operators to specify temporal dependencies, we introduce a formalism to concisely express many dependencies. For example, a single statement can specify that a user can read *all* the files that another user can read, relatively to an interval of time. A very general framework to specify authorization rules has been proposed by Woo and Lam in [24]. Their language to specify rules for the derivation of authorizations has almost the expressive power of first order logic. The tradeoff between expressiveness and efficiency seems to be strongly unbalanced in their approach. We think that it is important to investigate more restricted languages focusing on relevant properties. The temporal language we propose in this paper can be considered as a step in this direction. Another logic language for describing access control policies has been proposed by Abadi et Al. in [1]. However their logic is mainly used to represent concepts such as roles and delegation of authorities and their framework does not provide any mechanism to express temporal operators for the derivation of authorizations.

In this paper, we only deal with discretionary access control and not with mandatory access control. Note, however, that the majority of DBMS only provide discretionary access control. Therefore, since the focus of our research is how to extend the authorization facilities provided by a conventional DBMS, we only address discretionary access control. Recent multilevel DBMS (like Trusted Oracle [14]) provide mandatory access control coupled with discretionary access control. Therefore, the new features provided by our model could be orthogonally incorporated into such systems as well.

The remainder of this paper is organized as follows. In Section 2 we present the temporal authorization model. We start by formalizing our notion of time and making some assumptions used in the proposed model. We then define a Temporal Authorization Base as composed of explicit authorizations and derivation rules (dependencies). A first order logic semantics is provided. This section ends defining when an access request is authorized. In Section 3 we show that we cannot have sets of rules leading to an inconsistent authorization base, but we formally characterize sets of rules that could have *undesirable* effects and we sketch an algorithm for their detection. The issue of administration, including formal syntax of primitives for adding, removing, and modifying temporal authorizations is dealt with in Section 4. A few remarks about possible extensions of this work conclude the paper.

2 The authorization model

In this section we introduce our authorization model.

2.1 Model of time and basic assumptions

We assume a discrete model of time. In the database community, a *chronon* usually identifies the smallest indivisible unit of time. We can take a chronon or a fixed multiple of a chronon as our time unit. In any case, our model of time is isomorphic to the natural numbers $\mathbb{N}$ with a total order relation $<$.

For the moment, we assume the transactions in the database to be instantaneous. Moreover, we ignore the time required by the system to check if a certain authorization is present or can be derived. These assumptions allow us to consider the time $\mathtt{t}$ at which an access is requested and to look for a temporal authorization allowing that access at time $\mathtt{t}$. In practice, while the system is trying to derive the authorization, the instant $\mathtt{t}$ could become an instant in the past and the actual access will not be made at that instant. Several implementation techniques can be used to overcome these problems, but we do not address in this paper the issue of relaxing these assumptions.

We do not make any assumptions on the underlying data model to which accesses must be controlled, and therefore on the access modes users can exercise in the systems. The choice of the data model and of the access modes executable on the objects of the model is to be made when a system is initialized. In this way our authorization model can be used for the protection of information in different data models.

2.2 Temporal authorizations

Authorizations specify which accesses by users to the objects of the system are allowed. Let S denote the set of subjects (users) in the system, O the set of objects, and M the set of access modes. Authorizations are formally defined as follows.

Definition 2.1 (Authorization) *An authorization is a triple* $(\mathtt{s},\mathtt{o},\mathtt{m})$ *where*

- $\mathtt{s} \in S$ *is the subject (user) to whom the authorization is granted*
- $\mathtt{o} \in O$ *is the object on which the authorization is granted*
- $\mathtt{m} \in M$ *is the access mode, or privilege, for which the authorization is granted.*

Tuple (s,o,m) states that user s is authorized to execute access mode m on object o.

In our model we consider a temporal constraint to be associated with each authorization. We refer to an authorization together with a temporal constraint as a *temporal authorization*. Temporal authorizations are defined as follows.

Definition 2.2 (Temporal authorization) *A temporal authorization is a pair* (time, auth), *where* time *is a time interval* $[t_i, t_j]$, *with* $t_i \in \mathbb{N}$, $t_j \in \mathbb{N} \cup \infty$, $t_i \leq t_j$, *and* auth *is an authorization.*

Temporal authorization $([t_1,t_2],(s,o,m))$ states that subject s is allowed to exercise access mode m on object o in the interval $[t_1,t_2]$ including time instants t_1 and t_2.

A simple authorization, i.e., an authorization without any temporal constraint, can be represented as a temporal authorization whose validity spans from the time to which the authorization is granted to infinity.[1]

2.3 Derivation rules

In our model additional authorizations can be derived from the explicitly specified authorizations. The derivation is based on temporal propositions used as rules. They allow new temporal authorizations to be derived on the basis of the presence or absence of other temporal authorizations. The authorizations considered valid at a given time are, beside the authorizations explicitly stated, all the authorizations which can be derived from the rules. Derivation rules are defined as follows.

Definition 2.3 (Derivation rule) *A derivation rule is defined as* $(t_r: A_1 \langle \text{temp-operator} \rangle A_2)$, *where* t_r *is the time at which the rule has been specified,* A_1 *and* A_2 *are authorizations, and* ⟨temp-operator⟩ *is one of the following operators:* WHENEVER, ASLONGAS, WHENEVERNOT, UNLESS.

Unlike authorizations, derivation rules do not have associated time intervals. A rule is considered valid from the time t_r of its insertion until the time it is deleted or infinity.

The intuitive semantics of derivation rules is as follows:

- $(t_r: A_1$ WHENEVER $A_2)$.
  A_1 can be considered valid for each time instant for which A_2 is valid.
- $(t_r: A_1$ ASLONGAS $A_2)$.
  If A_2 is valid at the time t_r at which the rule is specified, then also A_1 is valid and it will remain valid until A_2 validity will expire. With respect to WHENEVER, using this operator, we do not ensure validity of A_1 in future instants where A_2 will be valid again.
- $(t_r: A_1$ WHENEVERNOT $A_2)$.
  A_1 can be considered valid for each time instant for which A_2 is not valid.
- $(t_r: A_1$ UNLESS $A_2)$.
  If A_2 is not valid at the time t_r at which the rule is specified, then A_1 becomes valid and it will remain valid until the last instant before the one at which A_2 becomes valid. With respect to WHENEVERNOT, using this operator, we do not ensure validity of A_1 the next time that A_2 becomes not valid again.

Example 2.1 Consider the authorizations and derivation rules illustrated in Figure 1.

The following temporal authorizations can be derived:

- ([5,9], (Bob,o_1,read)) from rule R_1 and authorization A_1, where 5 is the instant at which R_1 is inserted, and 9 is the instant preceding the first instant at which (Alice,o_1,read) becomes valid.
- ([6,9], (John,o_1,read)), ([21,29], (John,o_1,read)), and ([41,∞], (John,o_1,read)), from rule R_2.
- ([10,20], (Sam,o_1,read)), and ([30,40], (Sam,o_1,read)), from rule R_3 and authorizations A_1 and A_2.
- ([15,20], (Matt,o_1,read)) from rule R_4 and authorization A_1, where 15 is the instant at which R_4 is inserted, and 20 is the instant preceding the first instant at which (Alice,o_1,read) becomes not valid.

Our model allows derivation rules where up to two elements of the authorizations can be left unspecified, i.e., can be bound to different values. We refer to these rules as parametric derivation rules. Parametric derivation rules are formally defined as follows.

Definition 2.4 (Parametric derivation rule) *A parametric derivation rule is a derivation rule where symbol "–" appears for subjects, objects, or access modes in the authorizations. If symbol "–" appears in an authorization of the rule, it must appear, in the same position, also in the other authorization.*

Symbol "–" is a parameter which denotes any subject, object, or access mode depending on its position in the authorization.

Example 2.2 Consider the following parametric rules:

- (10:(Bob,o_1,–) WHENEVER (Alice,o_1,–)) states that, starting at time 10, Bob can exercise, on object o_1, all privileges which Alice can exercise.
- (10:(Bob,–,read) WHENEVER (Alice,–,read)) states that, starting at time 10, Bob can read all objects which Alice can read.

[1] This corresponds to having an implicit ALLTIME operator, often used in temporal logics [23].

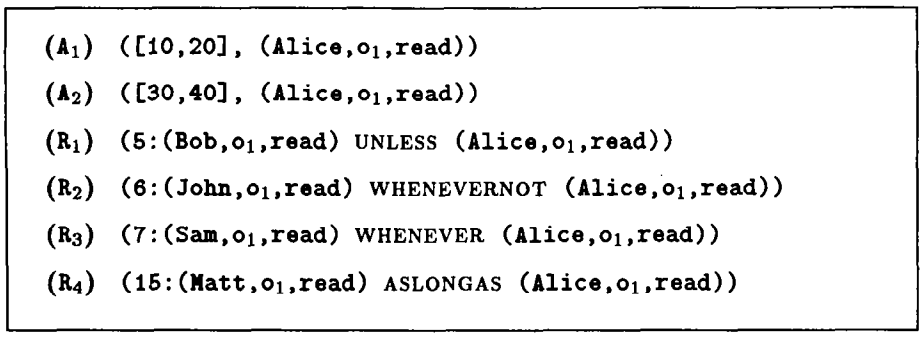


Figure 1: An example of authorizations and derivation rules

In the following, we use the term derivation rule to refer interchangeably to parametric and nonparametric derivation rules. We will explicitly refer to parametric or nonparametric derivation rules when a distinction is needed.

Note that parametric derivation rules also allow the users to specify taxonomies of subjects, objects, and access modes [15]. These rules support the derivation of authorizations on the basis of these taxonomies without the need of introducing any further control in the model.

For example, so far we have considered users as the only subjects of the authorizations. However, our model allows authorizations to be granted to groups of subjects, where subjects can be either users or groups. Authorizations specified for a group are valid for all the members of the group. To illustrate, suppose a user (or group) u is to be considered as a member of a group G. This can be expressed with a rule of the form:

$(t_r:(u,-,-)$WHENEVER$(G,-,-))$

where t_r is the membership time of u in G. According to this rule all authorizations valid for subject G are valid also for subject u, which is exactly the semantics of group membership.

One rule of the form above must be specified for each member of a group. Deletion of the rule is equivalent to the removal of the member from the group.

Derivation of authorizations on the basis of relationships between access modes can be supported in an analogous way. For example, we can state that the authorization to write an object implies the authorization to read the same object as follows:

$(t_r:(-,-,$read$)$ WHENEVER $(-,-,$write$))$.

2.4 Formal semantics

In this section we formalize the semantics of temporal authorizations and authorization rules.

Definition 2.5 (Temporal Authorization Base) *A Temporal Authorization Base* (TAB) *is a set of temporal authorizations and derivation rules.*

Definition 2.6 (Valid authorization) *An authorization* (s,o,m) *is valid at time* t *if, at time* t, *a temporal authorization* $([t_1,t_2],(s,o,m))$, *with* $t_1 \leq t \leq t_2$ *is present in* TAB *or it can be derived from it through the derivation rules.*

We define a function f which, given an authorization, a TAB at a time t, and an authorization A = (s,o,m), returns "true" if A is valid at time t, returns "false" otherwise. In the following, notation $f(t,(s,o,m))$ denotes that (s,o,m) is valid at time t, whereas $\neg f(t,(s,o,m))$ indicates that (s,o,m) is not valid at time t.

The semantics of temporal authorizations and derivation rules is given in first order logic and is reported in Table 1. The semantics of a set X of temporal authorizations and/or derivation rules, denoted by $\mathcal{S}(X)$, is the conjunction of the first order formulas corresponding to each element in the set.

Note that a temporal authorization and a derivation rule can be removed and therefore not be applicable anymore for the derivation of authorizations. In the formalization, we take this possibility into account by associating with each temporal authorization and derivation rule, the time t_d at which it is removed. Note that time t_d is not a constant and it is not known a priori. We use it as a shorthand for expressing the point up to which a temporal authorization, or a derivation rule, is applicable.[2] A function *removed()* can be defined which, given a temporal authorization, or a derivation rule, X, and a time t returns "false" if, at time t, X is still present in the TAB, and "true" otherwise. Time t_d is the smallest time t for which function *removed*(t,X) returns "true".

The semantics illustrated in Table 1 considers only nonparametric derivation rules. In case of parametric rules, the semantics must be extended by adding a variable for each parameter (the same variable is used for corresponding parameters in the two authorizations of the rule) and quantifying over this variable. For example, the semantics of a parametric WHENEVER rule, respectively with one and two parameters is as follows.

[2] If the considered temporal authorization, or derivation rule, is never removed, t_d is ∞.

Auth/Rule	Semantics
$[\mathtt{t_i},\mathtt{t_j}],(\mathtt{s},\mathtt{o},\mathtt{m})$	$\forall t\,(\mathtt{t_i} \leq t \leq \min(\mathtt{t_j}, \mathtt{t_d}-1)) \rightarrow f(t,(\mathtt{s},\mathtt{o},\mathtt{m}))$
$\mathtt{t_r}{:}\mathtt{A}_1$ WHENEVER $\mathtt{A}_2$	$\forall t\,(\mathtt{t_r} \leq t < \mathtt{t_d} \wedge f(t,\mathtt{A}_2)) \rightarrow f(t,\mathtt{A}_1)$
$\mathtt{t_r}{:}\mathtt{A}_1$ ASLONGAS $\mathtt{A}_2$	$\forall t\,(\mathtt{t_r} \leq t < \mathtt{t_d} \wedge f(t,\mathtt{A}_2) \wedge \neg\exists t'\,\mathtt{t_r} \leq t' < t \wedge \neg f(t',\mathtt{A}_2)) \rightarrow f(t,\mathtt{A}_1)$
$\mathtt{t_r}{:}\mathtt{A}_1$ WHENEVERNOT $\mathtt{A}_2$	$\forall t\,(\mathtt{t_r} \leq t < \mathtt{t_d} \wedge \neg f(t,\mathtt{A}_2)) \rightarrow f(t,\mathtt{A}_1)$
$\mathtt{t_r}{:}\mathtt{A}_1$ UNLESS $\mathtt{A}_2$	$\forall t\,(\mathtt{t_r} \leq t < \mathtt{t_d} \wedge \neg f(t,\mathtt{A}_2) \wedge \neg\exists t'\,\mathtt{t_r} \leq t' < t \wedge f(t',\mathtt{A}_2)) \rightarrow f(t,\mathtt{A}_1)$

Table 1: Semantics of temporal authorizations and rules

$$\mathcal{S}(\ \mathtt{t_r}{:}(\mathtt{s}_1,\mathtt{o}_1,-)\ \text{WHENEVER}\ (\mathtt{s}_2,\mathtt{o}_1,-)) = \forall t,m\ \ (\mathtt{t_r} \leq t < \mathtt{t_d} \wedge f(t,(\mathtt{s}_2,\mathtt{o}_1,m))) \rightarrow f(t,(\mathtt{s}_1,\mathtt{o}_1,m))$$

$$\mathcal{S}(\mathtt{t_r}{:}(\mathtt{s}_1,-,-)\ \text{WHENEVER}\ (\mathtt{s}_2,-,-)) = \forall t,o,m\ \ (\mathtt{t_r} \leq t < \mathtt{t_d} \wedge f(t,(\mathtt{s}_2,o,m))) \rightarrow f(t,(\mathtt{s}_1,o,m))$$

Similarly, we can obtain the semantics of the parametric temporal rules involving other temporal operators and parameters in different positions.

2.5 Access Control

The access control determines which requests by users[3] to access the objects of the underlying data model can be allowed.

Definition 2.7 (Access request) *An access request is a pair* $(\mathtt{t},\mathtt{A})$, *where* $\mathtt{t} \in \mathbb{N}$ *is the time at which the access is requested, and* $\mathtt{A}$ *is the triple* $(\mathtt{s},\mathtt{o},\mathtt{m})$ *with*

$\mathtt{s} \in S$ *the user who requires the access*

$\mathtt{o} \in O$ *the object to be accessed*

$\mathtt{m} \in M$ *the privilege to be exercised on object* $\mathtt{o}$.

Every access request is checked against the current TAB to determine whether the access is authorized. An authorized access request is a request for which an authorization exists or it is derivable from the TAB. This is formalized by the following definition.

Definition 2.8 (Authorized access request) *An access request* $(\mathtt{t},\mathtt{A})$ *is authorized if and only if* $\mathtt{A}$ *is valid at time* $\mathtt{t}$.

Example 2.3 Consider the TAB illustrated in Figure 2.

- Request $(\mathtt{30},(\mathtt{John},\mathtt{o}_1,\mathtt{write}))$ is authorized based on rules $\mathtt{R}_2$ and $\mathtt{R}_1$ and on the fact that authorization $(\mathtt{Bob},\mathtt{o}_1,\mathtt{write})$ is not valid at time 30.
- Request $(\mathtt{30},(\mathtt{John},\mathtt{o}_2,\mathtt{write}))$ is authorized based on rule $\mathtt{R}_2$ and authorization $\mathtt{A}_3$.
- Request $(\mathtt{30},(\mathtt{Alice},\mathtt{o}_2,\mathtt{write}))$ is authorized based on rule $\mathtt{R}_3$ and authorizations $\mathtt{A}_1$ and $\mathtt{A}_3$. Both $\mathtt{A}_1$ and $\mathtt{A}_3$ are necessary for the validity of authorization $(\mathtt{Ann},\mathtt{o}_2,\mathtt{write})$ in the interval $[\mathtt{11},\mathtt{30}]$ as required for the application of rule $\mathtt{R}_3$.
- Request $(\mathtt{30},(\mathtt{Alice},\mathtt{o}_2,\mathtt{read}))$ is not authorized. In fact, the only rule from which this authorization could be derived is $\mathtt{R}_3$, however this rule is applicable only if the authorization $(\mathtt{Ann},\mathtt{o}_2,\mathtt{read})$ has been valid at each instant of the interval $[\mathtt{11},\mathtt{30}]$. This is not the case for the considered TAB.

[3] Although authorizations can be specified for groups of users, access requests are always made by users.

3 Critical sets of Rules

Considering the above semantics and the fact that we do not have negative authorizations, an inconsistency, intended as an unsatisfiable formula obtained as the semantics of a set of derivation rules, cannot occur in our system. This will be formally stated by Proposition 3.1. However, we identify undesirable states of the system that we characterize by the presence of a critical set of rules.

Intuitively, critical sets of rules are those that, starting from $\neg f(t,\mathtt{A})$, can derive $f(t,\mathtt{A})$ (the opposite cannot happen). A simple example is $(\mathtt{t_r}{:}\mathtt{A}_1$ WHENEVERNOT $\mathtt{A}_1)$. This rule states that every time $\mathtt{A}_1$ is not valid, it is valid. Indeed, if $\mathtt{A}_1$ is not in TAB nor can be derived by other rules, then, through this rule, it can be derived. By the given semantics, this is equivalent to state that $\mathtt{A}_1$ is always valid starting at $\mathtt{t_r}$.

Critical sets can result from a combination of several rules. Even supposing that only the system administrator can insert the rules, it can happen that he inserts a rule that triggers a critical set formed by a long rule chain. This can happen without an explicit will of the administrator and it can have an unexpected effect.

We do not allow critical sets of rules to be present in the system. The TAB is maintained free of critical sets. Each time a rule insertion is requested, it is accepted only if it does not form a critical set with the existing rules. In the rest of this section we formally define critical sets and we outline an algorithm for their detection.

We start by introducing the notion of instance of a parametric rule.

Definition 3.1 (Rule instance) *A derivation rule is an instance of a parametric rule if each occurrence of*

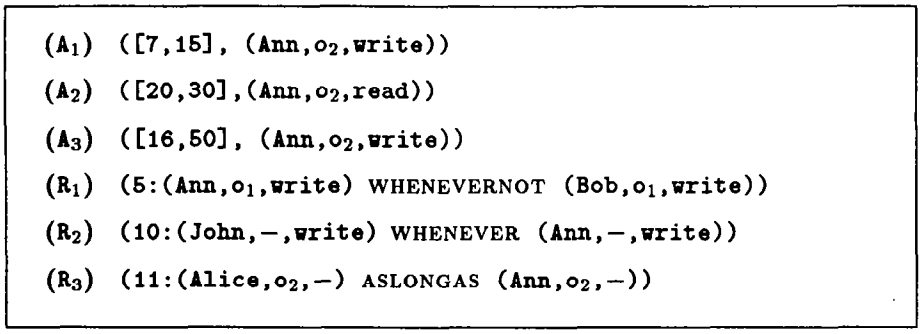


Figure 2: Example of Temporal Authorization Base with parametric rules

the symbol "−" has been substituted by an element of the domain S, O, or M, depending on the position of the symbol "−" in the authorization. The same element must be used in the corresponding position of the two authorizations.

Example 3.1 Consider the following rule

R = (t_r:(s_1,−,−) WHENEVER (s_2,−,−))

- (t_r:(s_1,o_1,write) WHENEVER (s_2,o_1,write)) is an instance of R
- (t_r:(s_1,o_1,read) WHENEVER (s_2,o_1,write)) is not an instance of R, since two different values have been substituted for symbol "−" in the access mode position.

To formally characterize critical sets, we first consider how we can derive an authorization from another one through a chain of WHENEVER and ASLONGAS operators.

Definition 3.2 (Positive derivability) *An authorization* A_m *positively derives from an authorization* A_n *at time* t *(from now on we write* $A_m \leftarrow_t A_n$*) if one of the following conditions holds at time* t*:*

* m = n
* *a rule* (t_r:A_m WHENEVER A_n) *is in* TAB *or it is an instance of a parametric rule in* TAB
* *a rule* (t_r:A_m ASLONGAS A_n) *is in* TAB *or it is an instance of a parametric rule in* TAB *such that* $\forall t'$ $t_r \leq t' \leq t$ *implies* $f(t', A_n)$
* *an authorization* A_k *exists such that* $A_m \leftarrow_t A_k \leftarrow_t A_n$.

We are now ready to identify critical sets of derivation rules.

Definition 3.3 (Critical set) *A* TAB *contains a critical set of rules at time* t *if one of the following conditions holds at time* t*:*

* *the rule* (t_r:A_n WHENEVERNOT A_m) *is in* TAB *or it is an instance of a parametric rule in* TAB *and* $A_m \leftarrow_t A_n$
* *the rule* (t_r:A_n UNLESS A_m) *is in* TAB *or it is an instance of a parametric rule in* TAB *such that* $\forall t'$ $t_r \leq t' < t$ *implies* $\neg f(t', A_m)$*; moreover* $A_m \leftarrow_t A_n$.

In the following, we use the term *negative operator* (NEGOP) to refer to WHENEVERNOT or UNLESS, and *negative rule* to refer to a rule using one of these operators.

Example 3.2 Consider the TAB reported in Figure 2 and suppose that the following rule is added at time 40:
(R_4) (40:(Bob,o_1,−) ASLONGAS (John,o_1,−))
This rule generates a critical set together with rule R_2 and with rule R_1 playing the role of the negative rule of the set. It is easily checked that Definition 3.3 applies to this set of rules. Intuitively, rule R_1 can be applied since, for example, ([40,40], (Bob,o_1,write)) is not in TAB and cannot be derived. However, the application of R_1 gives ([40,40], (Ann,o_1,write)); then, R_2 gives ([40,40], (John,o_1,write)) and finally, the application of R_4 returns ([40,40], (Bob,o_1,write)), contradicting its non derivability.

These sets of rules are considered critical essentially because they can have the effect of giving a permanent authorization when not intended by the administrator. This is considered a dangerous situation. However, the presence of a critical set does not lead to an inconsistency, as stated by the following proposition.

Proposition 3.1 *The semantics of a critical set of rules is always a satisfiable formula.*

Proof. The semantics of a critical set is a conjunction $F_1 \wedge \ldots \wedge F_k$ of logical formulas. If F_j represents the semantics of the only negative rule (t_r:A_m NEGOP A_n) in the critical set, F_j is an implication whose antecedent contains $\neg f(t, A_n)$. We take a first order interpretation assigning the TRUE value to $f(t, A_i)$ for each $t \in \mathbb{N}$ and for each A_i appearing in the critical set. This

interpretation satisfies each formula in the conjunction and hence the whole formula. Indeed, it satisfies F_j since $\neg f(t, A_i)$ evaluates to FALSE and, hence, the antecedent of the implication evaluates to FALSE. It satisfies also the other formulas since the consequent of the implications always evaluates to TRUE under this interpretation. □

A straightforward extension of Proposition 3.1 is obtained considering an arbitrary TAB (with or without critical sets) instead of a single critical set. This result actually implies that an inconsistent TAB cannot be generated in this model.

To explain why a critical set can have the effect of giving a "permanent" authorization we make the following remark.

Remark 1 *If $C1$ is a critical set of rules at time* t *with* (t_r:A_n WHENEVERNOT A_m) *as negative rule, then the semantics of $C1$ implies $f(t', A_m)$ (validity of the authorization A_m) for each time t' starting at instant* t *until any of the rules in the critical set is deleted.*

The result stated by the above remark can be formally checked quite easily. Consider the case of a WHENEVER rule as the only positive rule:

$$C1 = \{(t_1{:}A_2 \text{ WHENEVERNOT } A_1),\ (t_2{:}A_1 \text{ WHENEVER } A_2)\}$$

with $t_1 \leq t_2 \leq t_d$, where t_d is the minimum time between t_{d1} and t_{d2} that are respectively the instants at which the first and the second rules are deleted.

$$\begin{aligned} S(C1) &= \forall t\,(t_1 \leq t < t_{d1} \wedge \neg f(t, A_1) \rightarrow f(t, A_2)) \\ &\quad \wedge (t_2 \leq t < t_{d2} \wedge f(t, A_2) \rightarrow f(t, A_1)) \\ &\equiv \forall t\,(t_1 \leq t < t_2 \wedge \neg f(t, A_1) \rightarrow f(t, A_2)) \\ &\quad \wedge (t_2 \leq t < t_{d1} \wedge \neg f(t, A_1) \rightarrow f(t, A_2)) \\ &\quad \wedge (t_2 \leq t < t_{d2} \wedge f(t, A_2) \rightarrow f(t, A_1)) \\ &\equiv \forall t\,(t_1 \leq t < t_2 \wedge \neg f(t, A_1) \rightarrow f(t, A_2)) \\ &\quad \wedge (t_2 \leq t < t_d \wedge \neg f(t, A_1) \rightarrow f(t, A_2)) \\ &\quad \wedge (t_2 \leq t < t_d \wedge f(t, A_2) \rightarrow f(t, A_1)) \\ &\quad \wedge \ldots \\ &\equiv \forall t\,(t_2 \leq t < t_d \rightarrow f(t, A_1)) \wedge \ldots \end{aligned}$$

From the last expression we can see that the semantics of $C1$ implies the validity of A_1 ($f(t, A_1)$) from the time the critical set appears (t_2) until any of the rules in the critical set is deleted (t_d).

Example 3.3 With reference to Example 3.2 the introduction of rule R_4 would imply the validity of authorization (Bob,o_1,write) starting at time 40 until the time at which one of the rule in the critical set is deleted.

A remark similar to Remark 1 can be stated when considering the negative operator UNLESS.

Remark 2 *If $C1$ is a critical set of rules at time* t *with* (t_r:A_n UNLESS A_m) *as negative rule, then the semantics of $C1$ implies $f(t, A_m)$, i.e., it implies the validity of the authorization A_m only at time* t.

1. Consider the set of rules (t_r:A_n WHENEVERNOT A_m) and (t_r:A_n UNLESS A_m) and substitute each parametric rule with its possible instances.
2. Exclude UNLESS rules such that there exists t' with $t_r \leq t' < t$ and $f(t', A_m)$.
3. FOR each rule in the computed set DO
   (a) Check if $A_m \leftarrow_t A_n$. Instances of ASLONGAS rules must be checked for applicability as formally specified in Definition 3.2.
   (b) IF the previous check succeeds THEN RETURN "true".
4. RETURN "false"

Figure 3: Critical Set Detection

Example 3.4 Consider the TAB reported in Figure 2 and suppose that the following rule is added at time 60: (R_4) (60:(Ann,o_3,write) UNLESS (John,o_3,write)) This rule generates a critical set together with rule R_2. The presence of these two rules in the TAB would imply the validity of authorization (John,o_3,write) at time 60. Note that after time 60 rule R_4 would not be applicable anymore, i.e., (Ann,o_3,write) cannot be derived from it.

In Figure 3 we describe a general procedure for recognizing a critical set at time t. Essentially, it is necessary to instantiate all parametric rules and considering only rules that are applicable at time t. Note that in the implementation it is reasonable to maintain rules in their instantiated form. Moreover, the implementation could provide a mechanism to maintain the *current* set of applicable rules. This means ignoring UNLESS and ASLONGAS rule instances that cannot be applied any more. The main step consists in checking for each negative rule (t_r:A_n NEGOP A_m) if there exists a *positive* derivation from A_n to A_m. Several techniques can be used to implement this step as, e.g., some form of backward chaining. Several heuristics can be used for optimization. For example, the number of instances of negative parametric rules that must be considered can be reduced using a look-ahead technique.

4 TAB administration

Authorizations can be changed upon execution of administrative operations. In the present paper, we consider a centralized policy for the administration of authorizations, where administrative operations can be executed only by the *administrator*.

Administrative operations allow the administrator to add, remove, or modify temporal authorizations and derivation rules. Each temporal authorization, and each derivation rule, in the TAB is identified by a unique label assigned by the system at the time of its insertion. The label allows the administrator to refer to a specific temporal authorization or derivation rule upon execution of administrative operations.

Figure 4 reports the syntax, in BNF form, of the administrative operations considered by our model. Non terminal symbols ⟨subject⟩,⟨object⟩,⟨access-mode⟩, and ⟨nat-number⟩ represent elements of the domains S, O, M, and $\mathbb{N}$ respectively. Non terminal symbols ⟨aid⟩ and ⟨rid⟩ represent system labels.

A description of the administrative operations follows.

GRANT To grant a privilege on an object to a subject. The grant operation allows the administrator to give a subject the authorization for a privilege on an object from a given time (start time) to an end time. The end time can be specified explicitly, i.e., is a given instant, or as the span from the start time. The grant operation results in the addition of a new temporal authorization. The start time of the authorization must be greater than or equal to the time at which the authorization is inserted (i.e., it is not possible to specify retroactive authorizations). When a new authorization is inserted, a label (authorization identifier, *aid*) is assigned by the system.

REVOKE To revoke a privilege on an object from a subject. The revoke operation results in the deletion of all the temporal authorizations of the subject for the privilege on the object. Revoke can also be used to remove a specific authorization if a label is given as the only argument.

MODIFY To modify the temporal constraint of an authorization previously granted. The start time can be changed only if it is greater than the time at which the modification is requested, i.e., if the validity of the temporal authorization has not started yet. Analogously, the end time can be changed only if it is greater than the time at which the modification is requested, i.e, if the authorization has not expired yet. Moreover, the start time resulting after the operation must be greater than or equal to the time at which the modify operation is requested.

ADDRULE To add a new derivation rule. When a new rule is inserted, a label (rule identifier, *rid*) is assigned by the system.

DROPRULE To drop a derivation rule previously specified. The operation requires, as argument, the label of the rule to be deleted.

In our model, the validity of some authorizations (those derivable through rules) at a given time may depend on the validity of other authorizations at the same or at a different time. In particular, supporting temporal operators such as ASLONGAS and UNLESS requires evaluating the validity of other authorizations in a past time interval.

For this reason, we consider that every time a REVOKE/DROPRULE operation is entered the corresponding authorization/rule is not removed, rather is tagged as unusable with the time t_d at which the administrator asked for its deletion.

We classify temporal authorizations and derivation rules as follows:

Withdrawn Temporal authorizations and derivation rules whose removal has been explicitly required (with a REVOKE/DROPRULE command) by the administrator.

Expired Temporal authorizations and derivation rules which have not been withdrawn but which are not applicable anymore. They are authorizations whose end-time has passed and UNLESS/ASLONGAS rules from which no authorizations can be derived anymore.[4]

Active Temporal authorizations and derivation rules not withdrawn nor expired.

Expired or withdrawn authorizations and derivation rules cannot always be deleted from the TAB. The actual deletion of these authorizations or rules may imply the incorrect evaluation of other rules at a later time, as illustrated by the following example.

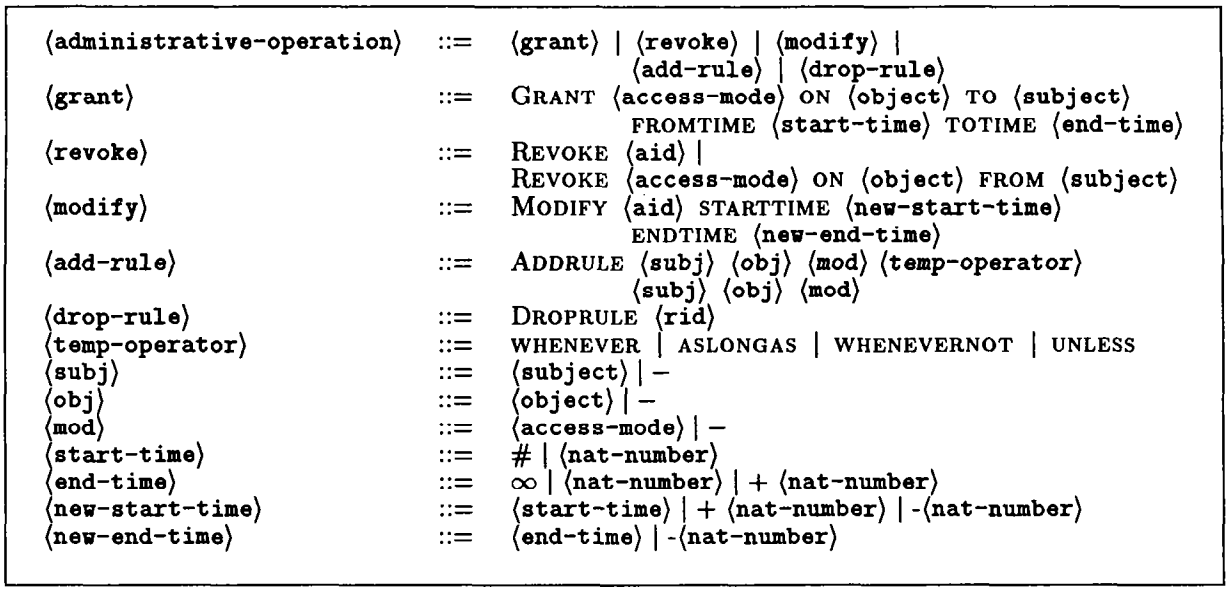
Example 4.1 Consider the TAB reported in Figure 2. Authorization A_1 expires at time 16, however it cannot be deleted at that time. Otherwise rule R_3 would not allow to derive authorization (Alice,o_2,write) after time 15. This would contradict the semantics of the considered set of rules.

We consider that, periodically, the TAB is examined to determine the temporal authorizations and derivation rules which have expired and those, among the expired and withdrawn ones, which can be deleted. More precisely, at each time period τ a cleaning process on the TAB is executed. Supposing the cleaning process is activated at time t_c, we can schematically describe its operation as follows.

1. Determine all authorizations whose end time is smaller than time t_c and mark them as expired.
2. Determine all UNLESS/ASLONGAS rules from which no authorization can be derived anymore and mark them as expired.
3. Delete from the TAB all expired and withdrawn authorizations which cannot contribute to firing of active rules.

Example 4.2 Consider the TAB reported in Figure 2 and suppose that the cleaning process is executed at time 40. No authorization nor rule has been withdrawn. Authorizations A_1 and A_2 are marked as expired. No rule has expired. Hence, the only candidates to be deleted are A_1 and A_2. For the same reasons illustrated

[4] The semantics of WHENEVER and WHENEVERNOT rules implies that they cannot expire.



⟨administrative-operation⟩ ::= ⟨grant⟩ | ⟨revoke⟩ | ⟨modify⟩ | ⟨add-rule⟩ | ⟨drop-rule⟩

⟨grant⟩ ::= GRANT ⟨access-mode⟩ ON ⟨object⟩ TO ⟨subject⟩ FROMTIME ⟨start-time⟩ TOTIME ⟨end-time⟩

⟨revoke⟩ ::= REVOKE ⟨aid⟩ | REVOKE ⟨access-mode⟩ ON ⟨object⟩ FROM ⟨subject⟩

⟨modify⟩ ::= MODIFY ⟨aid⟩ STARTTIME ⟨new-start-time⟩ ENDTIME ⟨new-end-time⟩

⟨add-rule⟩ ::= ADDRULE ⟨subj⟩ ⟨obj⟩ ⟨mod⟩ ⟨temp-operator⟩ ⟨subj⟩ ⟨obj⟩ ⟨mod⟩

⟨drop-rule⟩ ::= DROPRULE ⟨rid⟩

⟨temp-operator⟩ ::= WHENEVER | ASLONGAS | WHENEVERNOT | UNLESS

⟨subj⟩ ::= ⟨subject⟩ | −

⟨obj⟩ ::= ⟨object⟩ | −

⟨mod⟩ ::= ⟨access-mode⟩ | −

⟨start-time⟩ ::= # | ⟨nat-number⟩

⟨end-time⟩ ::= ∞ | ⟨nat-number⟩ | + ⟨nat-number⟩

⟨new-start-time⟩ ::= ⟨start-time⟩ | + ⟨nat-number⟩ | -⟨nat-number⟩

⟨new-end-time⟩ ::= ⟨end-time⟩ | -⟨nat-number⟩

Figure 4: Syntax of administrative operations

in Example 4.1, A_1 cannot be deleted, in fact it can still be used by rule R_3. By contrast, A_2 is deleted, since the only rule which could use it is R_3, but its application at time t, with $t \geq 40$, would require authorization (Ann,o_2,read) to be valid at each instant of interval [11,t], which is not the case for this TAB.

5 Conclusions

In this paper we have presented a formal definition of a temporal authorization model. Our model provides two new concepts with respect to previously proposed authorization models: temporal intervals of validity for authorizations and temporal dependencies among authorizations. Both those concepts are crucial to meet the articulate security requirements deriving from advanced applications, such as office automation, CAD, CSCW. To our knowledge, no previous model has dealt with temporal authorizations. However, given the increasing relevance of time in database systems, it is important to take into account time in all operational aspects of database systems. A qualifying aspect of our model is that it is independent from a specific data model. Indeed, it can be applied to relational database systems, object-oriented database systems, and deductive database systems.

The work reported in this paper can be extended in several directions, some of which we are currently investigating. First, the proposed model can be extended to consider a decentralized administration of authorizations. Under decentralized administration, several users are responsible for granting and revoking authorizations. In most models, the creator of an object is the owner of the object and is entitled to administrate authorizations on the object as well as to grant other users administration rights on the object. This policy can be imported in our model and revisited taking time into account.

A second direction concerns the extension of derivation rules. A straightforward extension consists in associating temporal intervals of validity with derivation rules. In this paper, we have made the assumption that a rule is valid from the time of its insertion until the time it is deleted or infinity. The model can be extended to admit rules with explicitly specified time intervals. After an interval elapses, the associated derivation rule is revoked. Another interesting extension can be obtained considering temporal operators on intervals as, for example, the ones used in [2].

A third direction deals with negative authorizations. Negative authorizations are given to explicitly forbid access to users on specific objects. Negative authorizations are particularly useful for modeling exceptions and to limit, in some cases, user discretionality in decentralized authorization administration. The introduction of negative authorizations in our temporal authorization model will lead to several interesting questions concerning both theory and implementation.

Finally, a fourth direction concerns implementation issues and authorization administration tools. The main complexity of our authorization model derives from the need to use an inference mechanism to derive authorizations from the authorizations stored into the authorization base. Therefore, this inference process must be enhanced by using techniques similar to those proposed for view materialization in relational databases and deductive databases. Administration tools are particularly crucial when dealing with sophisticated authorization models. In our model, for example, it is important to develop a tool providing information about derivation rules involved in critical sets. The area of administration tools has not, however, been so far widely

investigated. We plan to invest a major effort in this direction.

References

[1] M. Abadi, M. Burrows, B.W. Lampson, and G. Plotkin. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems*, 15(4):706–734, September 1993.

[2] J. F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154, 1984.

[3] V. Atluri, E. Bertino, and S. Jajodia. Achieving stricter correctness requirements in multilevel secure databases. In *Proc. IEEE Symposium on Security and Privacy*, pages 135–147, Oakland, California, May 1993.

[4] E. Bertino, L. Mancini, and S. Jajodia. Collecting garbage in multilevel secure object stores. In *Proc. IEEE Symposium on Security and Privacy, Oakland, California*, May 1994.

[5] E. Bertino and P. Samarati. Research issues in discretionary authorization for object bases. In B. Thuraisingham, R. Sandhu, and T.Y. Lin, editors, *Security for object-oriented systems*. Springer-Verlag, London, 1994.

[6] E. Bertino, P. Samarati, and S. Jajodia. Authorizations in relational database management systems. In *Proc. First ACM Conference on Computer and Communications Security*, Fairfax, Virginia, November 1993.

[7] S. Chokhani. Dod: trusted computer system evaluation criteria. *Communications of the ACM*, 35(7):66–76, July 1992.

[8] D. D. Clark and D. R. Wilson. A comparison of commercial and military computer security policies. In *Proc. IEEE Symposium on Security and Privacy*, pages 184–194, Oakland, California, April 1987.

[9] O. Costich. Transaction processing using an untrusted scheduler in a multilevel secure database with replicated architecture. In C.E. Landwehr, editor, *Database Security, V: Status and Prospects*, pages 173–189. North-Holland, Amsterdam, 1992.

[10] V. Doshi and S. Jajodia. Referential integrity in multilevel secure database management systems. In G.G. Gable and W.J.Caelli, editors, *IT Security: The Need for International Cooperation*, pages 359–371. North-Holland, 1992.

[11] E. B. Fernandez, E. Gudes, and H. Song. A security model for object-oriented databases. In *Proc. IEEE Symposium on Security and Privacy*, pages 110–115, Oakland, California, May 1989.

[12] P. P. Griffiths and B. W. Wade. An authorization mechanism for a relational database system. *ACM Trans. on Database Systems*, 1(3):242–255, September 1976.

[13] S. Jajodia and B. Kogan. Integrating an object-oriented data model with multilevel security. *Proc. IEEE Symposium on Security and Privacy, Oakland, California*, pages 76–85, May 1990.

[14] W. T. Maimone and I. B. Greenberg. Single-level multiversion schedulers for multilevel secure database systems. In *Proc. 6th Annual Computer Security Applications Conf.*, pages 137–147, Tucson, Arizona, December 1990.

[15] F. Rabitti, E. Bertino, W. Kim, and D. Woelk. A model of authorization for next-generation database systems. *ACM Trans. on Database Systems*, 16(1):88–131, March 1991.

[16] R.S. Sandhu. Separation of duties in computerized information systems. In S. Jajodia and C.E. Landwehr, editors, *Database Security, IV: Status and Prospects*, pages 179–189. North-Holland, Amsterdam, 1991.

[17] W.R. Shockley, M. Heckman, R.R. Schell, D.E. Denning, and T.F. Lunt. The SeaView security model. *IEEE Transactions on Software Engineering, Vol. 16, No. 6*, pages 593–607, June 1990.

[18] K.P. Smith. *Managing rules in active databases.* PhD Thesis, December 1992.

[19] D. L. Spooner. The impact of inheritance on security in object-oriented database systems. In C.E. Landwehr, editor, *Database Security, II: Status and Prospects*, pages 141–160. North-Holland, Amsterdam, 1989.

[20] G. Steinke and M. Jarke. Support for security modeling in information systems. In B.M. Thuraisingham and C.E. Landwehr, editors, *Database Security, VI: Status and Prospects*, pages 125–141. North-Holland, Amsterdam, 1993.

[21] R.K. Thomas and R.S. Sandhu. Discretionary access control in object-oriented databases: Issues and research directions. In *Proc. 16th National Computer Security Conference*, pages 63–74, Baltimore, MD, Sept. 1993.

[22] M. B. Thuraisingham. Mandatory security in object-oriented database system. In *Proc. Conf. on Object-Oriented Programming: Systems, Languages,and Applications*, pages 203–210, October 1989.

[23] Johan van Benthem. Temporal logic. In D. Gabbay, C. Hogger, and J. Robinson, editors, *Handbook of logic in artificial intelligence and logic programming*, volume 3. Oxford University Press, 1991.

[24] T.Y.C. Woo and S.S. Lam. Authorizations in distributed systems: A new approach. *Journal of Computer Security*, 2(2 & 3):107–136, 1993.