SPECIFYING AND COMPUTING HIERARCHIES OF TEMPORAL AUTHORIZATIONS

E. Bertino, P.A. Bonatti, E.Ferrari and M.L. Sapino

Abstract

We present a powerful authorization mechanism which supports: (1) positive and negative temporal authorizations; (2) user-defined temporal rules, by which authorizations can be derived; (3) the hierarchical organization of subjects and objects, with an associated mechanism for authorization inheritance.

The resulting model is very flexible in terms of the protection requirements it can represent. Such flexibility requires a non trivial underlying formal model. In particular, when inheritance and derivation rules are used simultaneously, there is need for conditions ensuring that the authorization base is free from ambiguities. For this purpose, we introduce a notion of *safeness*, and prove that it guarantees the absence of ambiguities and inconsistencies in the specification.

Keywords: Hierarchical Temporal Authorization Model,

1. INTRODUCTION

In many application domains, the need of restricting access permissions to specific time intervals or periods arises naturally. Authorizations often need to be tailored to the pattern of users' activities; as an example, consider part-time staff that should be authorized for accesses only on working days between 9 a.m. and 1 p.m. Temporal authorizations are also crucial for optimizing resource usage. For instance, execution of resource-expensive programs might be restricted to specific time periods.

Another crucial requirement is the possibility of expressing the relationships that usually exist among authorization subjects and objects. In most application domains, subjects and objects are *hierarchically* organized. The semantics of these hierarchies depends on the considered domain. For instance, the object hierarchy usually defines the composition of an object in terms of other objects

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: [10.1007/978-0-387-35508-5_22](https://doi.org/10.1007/978-0-387-35508-5_22)

V. Atluri et al. (eds.), *Research Advances in Database and Information Systems Security*

© IFIP International Federation for Information Processing 2000

(e.g., a directory immediately precedes in the hierarchy the files it contains), whereas the subject hierarchy usually reflects the relative position of subjects within the organization.

In this paper, we present an authorization model that provides integrated support for all the aforementioned features. In the model, *temporal* authorizations can be specified: positive authorizations, for granting a privilege; negative authorizations, for explicitly denying a privilege. A temporal authorization is an authorization that holds for specific periods of time. Subjects and objects are organized into hierarchies, supporting a more adequate representation of their semantics. Authorizations automatically propagate along these hierarchies in that a subject may inherit authorizations specified for more general subjects and objects, *unless* different specific authorizations are explicitly provided. Through inheritance, many protection requirements can be expressed concisely. For instance, a user may authorize all the employees to access the files of a given directory with only one authorization, having as subject the class "employee" and as object the directory. Such an authorization automatically propagates to all the employees and all the files in the directory. Furthermore, owners can always specify positive and negative exceptions to such general authorizations, by issuing authorizations at the lowest levels in the hierarchies. Thus, the resulting model provides a high degree of flexibility coupled with the possibility of enforcing stricter controls on crucial data items.

Our model supports also temporal derivation rules, by which many protection requirements can be concisely and clearly specified. For example, it is possible to state that two users, working on the same project, must receive the same authorizations on certain types of objects, or that a user can access an object in certain periods, provided that nobody else is allowed to access the same object in the same periods.

The work reported in this paper is based on the model presented in [1], that supports both temporal authorizations and derivation rules. The current paper extends [1] with the possibility of hierarchically organizing authorization subjects and objects, and with the related inheritance mechanism. This extension is particularly important when dealing with the interoperability of heterogeneous, distributed systems. In this framework, different local security policies have to be represented and integrated at a global level, to guarantee interoperability and to detect possible inconsistencies and security breaches [9]. Such global representation and integration involves (among other issues) establishing mappings among different object types and subjects. Such mappings, as when dealing with integration of heterogeneous data models, require the notion of hierarchy for both the object types and the subject types. We have therefore extended our previous model with hierarchies in view of supporting temporal authorizations in heterogeneous, distributed systems.

The introduction of hierarchies for authorization objects and subjects raises several theoretical and performance issues. When inheritance and authorization rules are used simultaneously, subtle ambiguities may easily arise, even in very simple specifications.

Example 3 Consider the following rules: (R_1) if subject s has write permission on object o , then s has also read permission on o ; (R_2) if s does not have read permission on object o , then s must not have write permission on o , either. Assume also that s' is the head of the administration department, and that an authorization A_1 stating that s' can write all documents of his/her department therefore exists. Now consider a classified administration document o' . The security administrator tries to protect this document through a general authorization A_2 stating that no user can read o' . Unfortunately, the resulting security specification is ambiguous. In fact, there are two mutually incompatible possibilities:

1. an authorization A'_1 , stating that s' can write o' , is inherited from A_1 , because o' is an instance of the class of documents of the administration department. By R_1 , it follows that s' can read o' . Such inferred authorization blocks inheritance from A_2 ; that is, it is *not* derived that s' cannot read o' ;
2. an authorization A'_2 , stating that s' cannot read o' , is inherited from A_2 , because s' is an instance of the class of all users. By R_2 , it follows that s' cannot write o' . This blocks inheritance from A_1 ; that is, A'_1 cannot be derived, and s' cannot write o' .

Clearly, it would be hard to detect such ambiguities in specifications of realistic size, without the help of some automated tool. This task is further complicated by the fact that authorizations can be independently formulated by different users, and, generally, they are not aware of every single authorization and rule in the system. The standard solution to this problem (adopted in [1]) consists in requiring the rules to satisfy so-called *stratifiability* conditions. Unfortunately, stratifiability does not solve the problem; in Example 3 above, rules R_1 and R_2 can be encoded in a specification which is stratified, according to [1]. Technically speaking, the problem is that inheritance introduces “hidden rules” that make authorization bases *always* equivalent to *non-stratified* logic programs [5]. Such programs, in general, may be ambiguous (in the above sense) or inconsistent. Moreover no general, efficient way of computing their models is known (the problem is NP-hard). Thus, dealing simultaneously with rules and inheritance requires the development of new techniques.

For this purpose, in this paper, we introduce a notion of *safeness*, and prove that it guarantees the absence of ambiguities and inconsistencies in the speci-

fication. Moreover, we define an efficient algorithm for computing authorizations from safe specifications, by dynamically refining a sort of stratification during the computation.

The introduction of temporal features in access control on one side, and the use of logical formalisms for specifying authorizations, on the other side, have been addressed by other research efforts. In particular, the Kerberos [10] system provides a notion of *ticket*, with an associated validity time. The purpose of temporal tickets is very different from our access control model. The former record only the fact that a client has been authenticated by the authentication server; they cannot be used to grant access to specific documents or resources managed by the server.

From the side of logical formalisms for security specifications, Woo and Lam in [12] propose a very general formalism for expressing authorization rules. Their language does not support temporal constraints explicitly, but it has almost the same expressive power as first order logic; this makes it possible to model temporal constraints. The main drawback is that the trade-off between expressiveness and efficiency seems to be strongly unbalanced in their approach. Jajodia et al. in [7] propose a logic language for expressing authorization rules. Their language can express most of the policies definable through the access control mechanisms proposed so far in the literature; however, temporal authorizations cannot be expressed. The development of flexible authorization models has been addressed in other papers [3, 11]. However none of these proposals support a complete set of features as our model, nor they provide a formal foundation to their models.

The remainder of this paper is organized as follows. Section 2. introduces the formalism we use to represent periodic time. Section 3. illustrates the basic components of the authorization model. Section 4. gives their formal semantic. Section 5. introduces the notion of *safe* authorization base and gives a mechanism to check whether an authorization base satisfies this condition. Section 6. concludes the paper and outlines future work.

2. PERIODIC EXPRESSIONS

A symbolic (user friendly) formalism to represent periodic time is provided. The formalism consists of a pair $\langle [\text{begin}, \text{end}], P \rangle$ where P is a *periodic expression* denoting an infinite set of time intervals, and $[\text{begin}, \text{end}]$ denotes the lower and upper bounds that are imposed on time intervals in P .

The formalism for periodic expressions is essentially the one proposed in [8], based on the notion of *calendars*. A calendar is defined as a countable set of contiguous intervals,¹ numbered by integers called *index* of the intervals.

A subcalendar relationship can be established between calendars. Given two calendars C_1 and C_2 , we say that C_1 is a subcalendar of C_2 , (written $C_1 \sqsubseteq C_2$), if each interval of C_2 is exactly covered by a finite number of intervals of C_1 . New calendars can be dynamically constructed from the existing ones, by means of a function *generate()* (cf. [1] for a formal definition), a *reference time instant*, and a *basic calendar* (the tick of the system), denoted by $\perp$. In the following, we postulate the existence of a set of calendars containing at least *Hours*, *Days*, *Weeks*, *Months*, and *Years*, where *Hours* is the calendar with finest granularity.

Calendars can be combined to represent more general sets of periodic intervals, not necessarily contiguous, like, for instance, the set of *Mondays* or the set of *The third hour of the first day of each month*. Complex sets of periodic intervals, like the ones above, are represented by means of *periodic expressions*, formally defined as follows.

Definition 5 [1] Given calendars $C_d, C_1, \dots, C_n$, a *periodic expression* is defined as $P = \sum_{i=1}^n O_i.C_i \triangleright r.C_d$, where $O_1 = all$, $O_i \in 2^{\mathbb{N}} \cup \{all\}$ and $C_i \sqsubseteq C_{i-1}$ for $i = 2, \dots, n$, $C_d \sqsubseteq C_n$, and $r \in \mathbb{N}$.

The left-hand side of $\triangleright$ identifies the set of starting points of the intervals, whereas the right-hand side specifies the duration. For example, $all.Years + \{3, 7\}.Months \triangleright 2.Months$ represents the set of intervals starting at the same instant as the third and seventh month of every year, and having a duration of 2 months. In practice, O_i is omitted when its value is *all*, whereas it is represented by its unique element when it is a singleton. $r.C_d$ is omitted when it is equal to $1.C_n$.

The infinite set of time intervals corresponding to a periodic expression P is denoted by $\Pi(P)$. Function $\Pi()$ is formally defined as follows.

Definition 6 [1] Let $P = \sum_{i=1}^n O_i.C_i \triangleright r.C_d$ be a periodic expression, then $\Pi(P)$ is a set of time intervals whose common duration is $r \cdot C_d$, and whose set S of starting points is computed as follows:

- if $n=1$, S contains all the starting points of the intervals of calendar C_1 ;
- if $n > 1$, and $O_n = \{n_1, \dots, n_k\}$, then S contains the starting points of the $n_1^{th}, \dots, n_k^{th}$ intervals (all intervals if $O_n = all$) of calendar C_n included in each interval of $\Pi(\sum_{i=1}^{n-1} O_i.C_i \triangleright 1.C_{n-1})$.

¹Two intervals are contiguous if they can be collapsed into a single one (e.g., $[1, 2]$ and $[3, 4]$).

Figure 19.1 An example of subject and object hierarchies.

For simplicity, in this paper the bounds begin and end will be denoted by *date expressions* of the form mm/dd/yy:hh, with the obvious intended meaning; end can also be ∞ . The set of time instants denoted by $\langle [\text{begin}, \text{end}], P \rangle$ is formally defined as follows.

Definition 7 Let t be a time instant. $t \in \text{Sol}([\text{begin}, \text{end}], P)$ iff $t \in \Pi(P)$ and $t_b \leq t \leq t_e$, where t_b , t_e are the instants denoted by begin and end, respectively.

3. THE HIERARCHICAL TEMPORAL AUTHORIZATION MODEL (HTAM)

In this section we illustrate the basic components of our hierarchical temporal authorization model (HTAM for short).

In the following, S denotes the set of subjects, O is the set of objects, and M is the set of access modes. Members of S are the identifiers with which users connect to the system. We suppose identifiers may refer to single users (e.g., Ann or Bob) or roles (e.g., staff or manager). Subjects and objects are structured into hierarchies. Figure 19.1 shows an example of subject and object hierarchies.

HTAM supports positive and negative temporal authorizations with the following structure.

Definition 8 [1] A *temporal authorization* is a triple $([\text{begin}, \text{end}], \text{period}, \text{auth})$, where the pair $\langle [\text{begin}, \text{end}], \text{period} \rangle$ represents a set of time periods, and auth is a 5-tuple (s, o, m, pn, g) , with $s, g \in S$, $o \in O$, $m \in M$, and $\text{pn} \in \{+, -\}$.

Triple $([\text{begin}, \text{end}], P, (s, o, m, \text{pn}, g))$ states that subject s has been authorized (if $\text{pn} = '+'$) or denied (if $\text{pn} = '-'$) for access mode m on object o by subject g for *each* instant in $\text{Sol}([\text{begin}, \text{end}], P)$. For example, the temporal

```

(A1) ([95,5/20/98], ⊥, (managers, documents, read, +, Sam))
(A2) ([10/1/98, ∞], Fridays, (admin-staff, admin, write, +, Sam))
(A3) ([95, 98], Working-days, (Cleo, p1, read, -, Sam))
(R1) ([96, 99], Working-days, (tech-staff, admin, read, +, Sam) ASLONGAS
    ¬(admin-staff, admin, write, +, Sam))
(R2) ([98, ∞], Mondays and Fridays, (tech-staff, papers, write, +, Sam) UPON
    ¬((managers, papers, read, +, Sam) ∨ (admin-staff, papers, read, +, Sam)))
(R3) ([98, 99], Thursday, (Cleo, d1, read, -, Sam) WHENEVER
    (John, d2, read, +, Sam) ∧ (Ann, d2, read, +, Sam))
    
```

Figure 19.2 An example of TAB.

authorization $A_1 = ([1/1/96, \infty], \text{Mondays},^2 (\text{Matt}, o_1, \text{read}, +, \text{Bob}))$, specified by Bob, states that Matt has the authorization to read o_1 each Monday starting from 1/1/96.

We assume that conflicting temporal authorizations may be simultaneously specified. This is a natural assumption both in discretionary frameworks (where different grantors may concurrently grant authorizations on the same objects) and in federated databases (where independently developed local security policies need to be merged to obtain secure interoperation). Conflicts are dealt with according to the denials-take-precedence principle [6], unless one of the conflicting authorizations is more specific than the others with respect to the specified hierarchies. In that case, less specific authorizations are overridden.

Example 4 Consider the hierarchies in Figure 19.1. Intuitively, a temporal authorization such as $([1/1/96, \infty], \perp, (\text{staff}, \text{admin}, \text{read}, +, \text{Ann}))$ means that all the staff members can read, starting from 1/1/96, each administration document, *unless explicitly stated otherwise*. So for instance, in the absence of further authorizations, $(\text{John}, d_3, \text{read}, +, \text{Ann})$ and $(\text{Bob}, d_7, \text{read}, +, \text{Ann})$ hold, starting from 1/1/96. On the contrary, in the presence of a temporal authorization such as $([1/1/97, \infty], \perp, (\text{Bob}, d_7, \text{read}, -, \text{Ann}))$, Bob would be allowed to read document d_7 only during 1996. More precisely, the authorization $(\text{Bob}, d_7, \text{read}, +, \text{Ann})$ would not be inherited after the end of 1996.

Similarly, a positive authorization may block the inheritance of a negative authorization. On the other hand, in case of multiple inheritance, conflicts are resolved in a conservative fashion, according to the denials-take-precedence principle [6].

²Here and in the following we use intuitive names for periodic expressions.

Example 5 Consider the following authorizations:

```
(([1/1/96,∞],⊥,(staff,d6,write,-,Cleo)),
([1/1/96,∞],⊥,(Bob,admin,write,+,Cleo)))
```

none is more specific than the other; in this case, the negative authorization is preferred, and only (Bob,d₆,write,-,Cleo) is inherited.

Furthermore, HTAM supports the specification of user-defined *derivation rules*, that make it possible to derive further authorizations from those explicitly given. Rule applicability can be constrained to specific time periods. Derivation rules are formally defined as follows.

Definition 9 [1] A *derivation rule* is a triple $([begin, end], period, A \text{ op } F)$, where the pair $\langle [begin, end], period \rangle$ represents a set of time periods, A is an authorization, F is a boolean combination of authorizations, and op is one of the operators: *whenever*, *aslongas*, *upon*. The authorization A is called the *head* of the rule; F is called *body*.

Definition 10 [1] A *temporal authorization base* (TAB) is a set of temporal authorizations and derivation rules.

Roughly speaking, rule $([begin, end], P, A \text{ op } F)$ states that for each instant in $Sol([begin, end], P)$, A can be derived provided that F is valid at the same instant, or in a certain set of past instants (depending on op). The grantor of A should be the user who specifies the rule.

The following is an intuitive account of the semantics of derivation rules. It will be assumed that all authorizations are granted by the same subject; therefore, the grantor will not be considered in the discussion.

- $([begin, end], P, A \text{ WHENEVER } F)$. We can derive A for each instant in $Sol([begin, end], P)$ where F is valid.
- $([begin, end], P, A \text{ ASLONGAS } F)$. Authorization A can be derived at instant t in $Sol([begin, end], P)$ if F is valid for each instant in $Sol([begin, end], P)$ from the first one up to t . For instance, rule R_1 in Figure 19.2 states that tech-staff can read admin each working day in $[1/1/96, 12/31/99]$ until the first working day in which admin-staff will be allowed to write admin.
- $([begin, end], P, A \text{ UPON } F)$. We can derive A for each instant t in $Sol([begin, end], P)$ such that there exists an instant t' in $Sol([begin, end], P)$ $t' \leq t$ such that F is valid at time t' . For instance, rule R_2 in

Figure 19.2 states that tech-staff can write papers each Monday and Friday starting from the first in 1998 in which both managers and admin-staff are not authorized to read papers.

Example 6 Consider the TAB in Figure 19.2 and the subject and object hierarchies in Figure 19.1. The following authorizations can be derived:

- (tech-staff, admin, read, +, Sam) for each Working day in [1/1/96, 10/2/98], from rule R_1 and authorization A_2 . Note that 10/2/98 is the first Friday after 10/1/98.
- (tech-staff, papers, write, +, Sam) for each Monday and Friday from 5/22/98, from rule R_2 and authorization A_1 . Note that authorization A_1 implies an authorization for managers to read papers for each day in [1/1/95, 5/20/98]. 5/22/98 is the first Friday in [1/1/98, ∞] in which neither managers nor staff can read papers.
- (Cleo, d_1 , read, -, Sam) for each Thursday from 1/1/98 to 5/20/98 from rule R_3 and authorization A_1 . Note that A_1 implies an authorization for Ann and John to read documents for each day from 1/1/95 to 5/20/98 and this authorization implies that Ann and John can read document d_2 for the same days.

4. FORMAL SEMANTICS OF TABS

We start by introducing some notation. The hierarchies of subjects and objects are formalized by two partially ordered sets $(S, \sqsubseteq_S)$ and $(O, \sqsubseteq_O)$, respectively. For example, according to the hierarchies illustrated in Fig. 19.1, we have (among other relationships) $\text{Ann} \sqsubseteq_S \text{managers}$, $\text{John} \sqsubseteq_S \text{staff}$, $p_i \sqsubseteq_O \text{documents}$, and $p_i \not\sqsubseteq_O \text{admin}$ ($1 \leq i \leq k$).

Let $\mathcal{A}$ be the set of all the tuples (s, o, m, pn, g) . Given an authorization $A = (s, o, m, pn, g)$ we use the notation $s(A)$, $o(A)$, $m(A)$, $pn(A)$, $g(A)$ to denote the corresponding components of A . Let A^+ (resp. A^-) denote the authorization obtained from A by replacing $pn(A)$ with '+' (resp. '-'). For all $A \in \mathcal{A}$, $\bar{A}$ denotes the authorization *complementary to* A , that is, $\bar{A}^+ = A^-$ and $\bar{A}^- = A^+$. The relation ' $=_{-g}$ ' defined below is the equality of authorizations modulo grantors; the set $\text{Conflicts}(A)$ captures the set of authorizations conflicting with A :

$$A =_{-g} B \equiv s(A) = s(B) \wedge o(A) = o(B) \wedge m(A) = m(B) \wedge pn(A) = pn(B),$$

$$\text{Conflicts}(A) = \{ B \mid B =_{-g} \bar{A} \}.$$

Note that $B \in \text{Conflicts}(A) \iff A \in \text{Conflicts}(B)$, and $A =_{-g} B \Rightarrow \text{Conflicts}(A) = \text{Conflicts}(B)$. A *conflicting pair* consists of two authorizations A and B such that $A \in \text{Conflicts}(B)$ (equivalently, $B \in \text{Conflicts}(A)$). The subject and object hierarchies induce the following natural partial ordering $\sqsubseteq_{\mathcal{A}}$ over $\mathcal{A}$:

$$\begin{aligned} A \sqsubseteq_{\mathcal{A}} B &\equiv s(A) \sqsubseteq_S s(B) \wedge o(A) \sqsubseteq_O o(B) \wedge \\ &m(A) = m(B) \wedge pn(A) = pn(B) \wedge g(A) = g(B). \end{aligned}$$

As usual, $A \sqsubset_{\mathcal{A}} B \equiv A \sqsubseteq_{\mathcal{A}} B \wedge A \neq B$. Finally, define the set:

$$\text{Parents}_{\sqsubseteq_{\mathcal{A}}}(A) = \{ B \mid A \sqsubset_{\mathcal{A}} B \text{ and } \nexists C. A \sqsubset_{\mathcal{A}} C \sqsubset_{\mathcal{A}} B \}.$$

The subscripts of $\sqsubseteq_{\mathcal{A}}$ and $\text{Parents}_{\sqsubseteq_{\mathcal{A}}}$ will often be dropped to enhance readability.

Access control is based on the set of *time-stamped authorizations* that can be derived from the given TAB T . A time-stamped authorization is a pair (t, A) where t is a non-negative integer (a “tick” in the basic calendar) and $A \in \mathcal{A}$ (the intuitive meaning is “ A holds at time t ”).

Formalizing derivability involves delicate technicalities due to the presence of *negation as failure*. Rules such as $R = ([t, t], \perp, A \text{ whenever } \neg B^-)$ are triggered only if (t, B^-) is *not derivable*, and (t, A) should be inherited only if $(t, \bar{A})$ is *not derivable* (according to the overriding mechanism outlined in the previous section). Consequently, derivable authorizations cannot be defined incrementally through a classical inductive definition; for example, the above rule R may be applicable at a certain stage of the derivation process, but as deduction goes on, (t, B^-) might become derivable, thereby invalidating the conclusion of R . Informally speaking, in order to decide whether R can be applied we should *guess* whether (t, B^-) will ever be derived in the future. Fortunately, a similar form of negation has already been formalized in logic programming, through the *stable model semantics* [4]; here we adopt the same idea.

Intuitively, the definition of derivable authorizations models the following three steps:³ (i) First, the set of all derivable time-stamped authorizations, I , is guessed; (ii) the consequences of the given TAB T are derived incrementally, by recursively applying all the applicable rules of T and inheritance; the applicability of inheritance and rules with negative literals in the body—like R —is checked using I to “guess the future”; (iii) clearly, the initial guess is correct if it coincides with what can be actually derived with the given rules, i.e. if I equals the set of authorizations derived in step (ii).

³The efficient algorithms introduced in the next section apply only to *safe* TABs and do not match the generate-and-test nature of the following steps.

Then, the formal definition corresponding to the above steps is just a fix-point equation $I = \Phi_T^I \uparrow \omega$, where I is the initial guess and $\Phi_T^I \uparrow \omega$ is the set of authorizations obtained in step (ii). In particular, Φ_T^I models a single application (in parallel) of all the applicable rules of T ,⁴ and the operator $\uparrow \omega$ iterates Φ_T^I .⁵

To formalize Φ_T^I in a compact way, we introduce a notion of *validity relative to the initial guess I* . Recall from the previous section that conflicting authorizations may be simultaneously derivable from T ; in this case the denials-take-precedence principle is applied; *validity* captures this idea, and it can be extended to arbitrary boolean combinations of authorizations in negation normal form (NNF).⁶ Intuitively, in the following, J stands for the set of authorizations that have been derived at some intermediate step of the derivation (I is the initial guess).

Definition 11 Let I and J be sets of time-stamped authorizations, t be a time point, $A \in \mathcal{A}$ and F, G_i be boolean combinations of authorizations from $\mathcal{A}$ in NNF. The *validity* of F in J at time t w.r.t. I , in symbols $J, I \models_t F$, is recursively defined as follows:

1. $J, I \models_t A^+$ if $(t, A^+) \in J$ and for all $B^- \in \text{Conflicts}(A^+)$, $(t, B^-) \notin I$;
2. $J, I \models_t A^-$ if $(t, A^-) \in J$;
3. $J, I \models_t \neg A^+$ if $(t, A^+) \notin J$ or there is $B^- \in \text{Conflicts}(A^+)$ s.t. $(t, B^-) \in J$;
4. $J, I \models_t \neg A^-$ if $(t, A^-) \notin J$;
5. $J, I \models_t G_1 \wedge G_2$ if $J, I \models_t G_1$ and $J, I \models_t G_2$;
6. $J, I \models_t G_1 \vee G_2$ if $J, I \models_t G_1$ or $J, I \models_t G_2$.

Now we can define the *immediate consequence* operator Φ_T^I that applies in parallel all applicable rules and performs all possible inheritance steps. In the following, J plays the role of the set of authorizations which have already been derived, and $\Phi_T^I(J)$ is the result of one more derivation step.

Definition 12

$$\begin{aligned} \Phi_T^I(J) = & \{ (t, A) \mid (TI, P, A) \in T \text{ and } t \in \text{Sol}(TI, P) \} \\ & \cup \{ (t, A) \mid (TI, P, A \text{ whenever } F) \in T, t \in \text{Sol}(TI, P), J, I \models_t F \} \\ & \cup \{ (t, A) \mid (TI, P, A \text{ upon } F) \in T, t \in \text{Sol}(TI, P) \text{ and} \\ & \quad \exists t' \in \text{Sol}(TI, P) \text{ such that } t' \leq t, J, I \models_{t'} F \} \\ & \cup \{ (t, A) \mid (TI, P, A \text{ as long as } F) \in T, t \in \text{Sol}(TI, P) \text{ and} \end{aligned}$$

⁴Note that this operator depends on I , as required by step (ii).

⁵We recall the recursive definition of $\uparrow$: $\Phi \uparrow 0 = \Phi(\emptyset)$; $\Phi \uparrow i + 1 = \Phi(\Phi \uparrow i)$; $\Phi \uparrow \omega = \bigcup_{i \geq 0} \Phi \uparrow i$.

⁶A formula is in negation normal form if negation is applied only to atoms. This is the form of the bodies of TAB rules.

$$\begin{aligned}
& \text{for all } t' \in \text{Sol}(TI, P) \text{ such that } t' \leq t, J, I \models_{t'} F \} \\
\cup & \{ (t, A) \mid \exists A_1 \in \text{Parents}(A). J, I \models_t A_1 \text{ and } \forall A_2 \in \text{Conflicts}(A). A_2 \notin I, \\
& \text{and if } A = A^+ \text{ then } \forall A_3 \in \text{Parents}(\text{Conflicts}(A)), J, I \models_t \neg A_3 \}.
\end{aligned}$$

The first four sets in the right-hand side formalize the semantics of TAB rules; the last set defines the semantics of inheritance. The precedence of negative authorizations over positive authorizations is obtained by inheriting A^+ only when no conflicting authorization $B \in \text{Conflicts}(A^+)$ can possibly be inherited; in turn, this is verified by checking that no parent A_3 of B is valid.

As we have already pointed out, by iterating the above operator we obtain all the facts derivable from T , given the initial guess I .

Proposition 4.1 *The operator Φ_T^I is continuous, and hence i) $\Phi_T^I \uparrow 0 \subseteq \Phi_T^I \uparrow 1 \subseteq \dots \subseteq \Phi_T^I \uparrow \omega$; ii) the least fixpoint of Φ_T^I is $\Phi_T^I \uparrow \omega$.*

Thus, the initial guess is correct iff I coincides with $\Phi_T^I \uparrow \omega$. So the notion of $\sqsubseteq$ -stable model introduced below captures the set of derivable authorizations.

Definition 13 A set of time-stamped authorizations I is a $\sqsubseteq$ -stable model of a TAB T iff $I = \Phi_T^I \uparrow \omega$.

Suppose that T has one $\sqsubseteq$ -stable model M ; then an authorization A should be enforced at time t iff A is valid in M at time t .

5. DYNAMICALLY STRATIFIED TABS

A TAB may have multiple $\sqsubseteq$ -stable models, or no $\sqsubseteq$ -stable models at all. In this section, we introduce conditions that guarantee that the TAB has exactly one $\sqsubseteq$ -stable model (and hence a consistent, non ambiguous semantics).

A *labeled dependency graph* is a graph whose nodes are authorizations from $\mathcal{A}$ and whose edges are labeled with $+$, $-$ or v (accordingly, the edges are called *positive*, *negative* or *variable*).

Let $DG_T(t)$ be the labeled dependency graph whose set of nodes is $\mathcal{A}$ and whose edges are all and only the triples $\langle A, l, B \rangle$ that satisfy some of the following conditions:

- (DG1) For some rule $(TI, P, B \langle \text{op} \rangle F) \in T$ such that $t \in \text{Sol}(TI, P)$, A occurs in F ; moreover, if A occurs in the scope of a negation sign ($\neg$), then $l = -$, otherwise $l = +$;
- (DG2) $A \in \text{Conflicts}(A_1^+)$, where A_1^+ occurs in the body F of some rule $(TI, P, B \langle \text{op} \rangle F) \in T$ such that $t \in \text{Sol}(TI, P)$; moreover, if A_1^+ occurs in the scope of a negation sign ($\neg$), then $l = +$, otherwise $l = -$;
- (DG3) $A \in \text{Parents}(B)$ and $l = -$;

(DG4) $A \in \text{Conflicts}(B)$ and $l = v$.

TABs will be required to satisfy the following *safeness conditions*. We shall prove that safeness guarantees the existence of a unique stable model.

Definition 14 A TAB T is *safe* if for all $t \geq 0$: 1) no cycle in $DG(t)$ contains a negative edge; 2) if A, B is a conflicting pair, then every path from A to B in $DG(t)$ contains a variable edge (A', v, B') such that $A' =_{-g} A$ and $B' =_{-g} B$.

A *stratification* of a set of authorizations $\mathcal{A}' \subseteq \mathcal{A}$ is a mapping $\pi : \mathcal{A}' \rightarrow \{1, \dots, |\mathcal{A}'|\}$. We say that π is *compatible* with a labeled dependency graph DG if: 1) for all edges $\langle A, l, B \rangle$ in DG , $\pi(A) \leq \pi(B)$, and 2) if $l = -$ then $\pi(A) < \pi(B)$.

In the standard logic programming setting, the existence of a compatible stratification is equivalent to the “safeness” of the program. On the contrary, here safeness is stronger (the second condition is not enforced by the existence of a stratification). Intuitively, the problem is that for every conflicting pair A, B , the inheritability of A depends on the absence of B and viceversa. Fortunately, this cyclic (non-stratifiable) dependency does not affect the nice properties of stratifiable programs (i.e., the existence of a unique stable model which can be computed in polynomial time).

The model of a TAB T which satisfies the safeness conditions can be constructed as follows. Let $\pi_t : \mathcal{A} \rightarrow \{1, \dots, |\mathcal{A}|\}$ be a stratification compatible with $DG_T(t)$. We call π_t a *pre-stratification* of $\mathcal{A}$ at time t . Define $\mathcal{A}_{t,l} = \{A \mid \pi_t(A) = l\}$. A *refined stratification* of $\mathcal{A}_{t,l}$ w.r.t. an interpretation J is a mapping $\sigma_{t,l} : \mathcal{A}_{t,l} \rightarrow \{1, \dots, |\mathcal{A}_{t,l}|\}$ such that for all $A, B \in \mathcal{A}_{t,l}$:

1. if $\langle A, +, B \rangle$ belongs to $DG_T(t)$ then $\sigma_{t,l}(A) \leq \sigma_{t,l}(B)$;
2. if $\langle A^+, v, B \rangle$ is in $DG_T(t)$ and $\exists B' \in \text{Parents}(\text{Conflicts}(A)). J, J \models_t B'$ then $\sigma_{t,l}(A) < \sigma_{t,l}(B)$;
3. if $\langle A^-, v, B \rangle$ is in $DG_T(t)$ and $\nexists A' \in \text{Parents}(\text{Conflicts}(B)). J, J \models_t A'$ then $\sigma_{t,l}(A) < \sigma_{t,l}(B)$.

Note that $\sigma_{t,l}$ is a stratification compatible with a modified version of $DG_T(t)$, where the variable edges over $\mathcal{A}_{t,l}$ have been either eliminated or turned into negative edges. The idea is that the negative cycles involving conflicting pairs can be “broken” dynamically, on the basis of the part of model built so far (represented here by J). Indeed, in the full paper [2] it is shown that for each conflicting pair A, B , either $\sigma_{t,l}(A) < \sigma_{t,l}(B)$ or $\sigma_{t,l}(B) < \sigma_{t,l}(A)$.

The model M is constructed through three nested iterations, over time, pre-stratification level and refined stratification level, respectively. In order to simplify notation, we shall use τ as an abbreviation of an arbitrary triple (t, l, i) of

the form $(0,0,0)$ or such that $t \geq 0$, $1 \leq l \leq |\mathcal{A}|$ and $1 \leq i \leq |\mathcal{A}_{t,l}|$. With a slight abuse of notation, the lexicographic ordering of such triples will be denoted by $\leq$, and the immediate predecessor and successor of τ in this ordering will be denoted by $\tau - 1$ and $\tau + 1$, respectively. Each time-stamped authorization (t, A) will be associated to a *layer* $\ell(t, A) = (t, l, i)$, where $l = \pi_t(A)$ and $i = \sigma_{t,l}(A)$. Finally, define by mutual recursion:⁷

$$M = \bigcup_{\tau \geq (0,0,0)} M_\tau, \quad \text{where}$$

$$M_\tau = \emptyset \quad \text{if } \tau = (0,0,0), \text{ otherwise}$$

$$M_\tau = \Phi_{T_\tau}^{M_{\tau-1}} \uparrow \omega(M_{\tau-1}), \quad \text{where}$$

$$\begin{aligned} T_{(t,l,i)} = & \{([t_b, t], P, A \langle \text{op} \rangle F) \mid ([t_b, t_e], P, A \langle \text{op} \rangle F) \in T, \\ & t \in \text{Sol}([t_b, t_e], P), \sigma_{t,l}(A) = i\} \\ \cup & \{([t_b, t], P, A) \mid ([t_b, t_e], P, A) \in T, \\ & t \in \text{Sol}([t_b, t_e], P), \sigma_{t,l}(A) = i\} \end{aligned}$$

and where $\sigma_{t,l}$ is a refined stratification of $\mathcal{A}_{t,l}$ w.r.t. $M_{(t,l,1)-1}$. Roughly speaking, $T_{(t,l,i)}$ is the set of rules and authorizations that can possibly be used to derive time-stamped authorizations whose layer is (t, l, i) . Note that $T_{(t,l,i)}$ depends on $M_{(t,l,1)-1}$ through $\sigma_{t,l}$.

First, it must be shown that the above construction is well defined, by proving that pre-stratifications π_t and refined stratifications $\sigma_{t,l}$ exist.

Theorem 5.1 *If T satisfies the safeness conditions then π_t and $\sigma_{t,l}$ exist for all $t \geq 0$ and for all l ($1 \leq l \leq |\mathcal{A}|$).*

Theorem 5.2 *M is the unique $\sqsubseteq$ -stable model of T .*

Definition 14 above gives a method to test whether a given TAB satisfies the safeness conditions. Such test is performed by checking a set of conditions, for each instant $t \geq 0$. Obviously, this test is feasible in practice only if we can safely stop it at some finite instant t . The particular form our rules allows us to ensure the existence of this finite constant, denoted in the following as *max-time*. *max-time* is determined as $\bar{t}_{max} + \bar{k} \cdot P_{max}$, where $\bar{t}_{max}$ is the greater instant corresponding to an end date expression in TAB, P_{max} is the least common multiple of all the periodicities of the periodic expressions appearing in TAB, and $\bar{k}$ is the maximum number of ASLONGAS and UPON

⁷Recall that $\Phi_T^I \uparrow 0(J) = J$ and $\Phi_T^I \uparrow i + 1(J) = J \cup \Phi_T^I(\Phi_T^I \uparrow i(J))$.

rules in TAB plus one. The important property is that, after instant *max-time*, the validity of any authorization in TAB becomes periodic. The proof of the existence of *max-time*, adapted from [1], is contained in [2].

Two different strategies can be used to enforce access control: *run-time derivation* and *materialization*. Under the latter approach, the system permanently maintains all the valid authorizations derivable from a TAB. Access control therefore becomes very efficient; as a drawback, the materialization has to be properly updated every time the TAB is modified. A detailed discussion of access control under the materialization approach is presented in [2].

6. CONCLUSIONS

In this paper we have presented a powerful authorization mechanism which provides support for temporal authorizations, user-defined derivation rules, and the hierarchical organization of subjects and objects.

We have given a solution to the problems related to the simultaneous presence of inheritance and authorization rules. Variable edges, safeness and the dynamic stratification mechanism have no counterpart in the literature. Safeness guarantees that the TAB is consistent and unambiguous.

Sometimes, in the presence of ambiguities, it may be possible to select one stable model of the TAB, by taking into account additional information, such as the grantors of the conflicting authorizations, or priorities defined over access modes. This will be the subject of future work.

References

- [1] Bertino, E., Bettini, C., Ferrari, E. and Samarati, P. (1998). An access control model supporting periodicity constraints and temporal reasoning. *ACM Transactions on Database Systems*, **23**(3), pp. 231–285.
- [2] Bertino, E., Bonatti, P., Ferrari, E. and Sapino, M.L. (1999). Specifying and Computing Hierarchies of Temporal Authorizations. Technical Report - University of Milano.
- [3] Bertino, E., Jajodia, S. and Samarati, P. (1996). Supporting multiple access control policies in database systems. *Proceedings of the IEEE Symposium on Research in Security and Privacy*.
- [4] Gelfond, M. and Lifschitz, V. (1988). The stable model semantics for logic programming. *Proceedings of the Fifth International Conference and Symposium on Logic Programming*, pp. 1070–1080.

- [5] Przymusiński, T.C. (1987). On the declarative semantics of deductive databases and logic programs. *Foundations of Deductive Databases and Logic Programming* (ed. J. Minker), Morgan Kaufmann.
- [6] Jajodia, S., Samarati, P., Subrahmanian, V.S. and Bertino, E. (1997). A unified framework for enforcing multiple access control policies. *Proceedings of the ACM SIGMOD International Conference on Management of Data*.
- [7] Jajodia, S., Samarati, P. and Subrahmanian, V.S. (1997). A logical language for expressing authorizations. *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pp. 31–42.
- [8] Niezette, M. and Stevenne, J. (1992). An efficient symbolic representation of periodic time. *Proceedings of the First International Conference on Information and Knowledge Management*.
- [9] Gong, L. and Qian, X. (1996). Computational Issues in Secure Interoperation. *IEEE Transactions on Software Engineering*, **22**(1), pp. 43–52.
- [10] Steiner, J.G., Neuman, C. and Schiller, J.I. (1988). Kerberos: An authentication service for open network systems. *Proceedings of the USENIX Conference*, pp. 191–202.
- [11] Rabitti, F., Bertino, E., Kim, W. and Woelk, D. (1991). A model of authorization for next-generation database systems. *ACM Transactions on Database Systems*, **16**(1), pp. 88–131.
- [12] Woo, T.Y.C. and Lam, S.S. (1993). Authorizations in distributed systems: A new approach. *Journal of Computer Security*, **2**(2 & 3), pp. 107–136.