Modelling a Multilevel Database with Temporal Downgrading Functionalities

F. Cuppens and A. Gabillon
ONERA-CERT
2 Av. E. Belin, 31055, Toulouse Cedex, France. Fax: +33 62 25 25 93.
email: {cuppens,gabillon}@tls-cs.cert.fr

Abstract
In many cases, it would be useful to automatically downgrade obsolete information. Most of the multilevel databases do not provide mechanisms which enable such downgrading to be specified and managed. In this paper, we propose a general language to cope with this problem. This language is based on first order logic with various extensions to represent concepts such as dating, classification and downgrading. This language enables the content of a temporal multilevel database to be specified. We also show that this language can be used to specify general and various downgrading rules. We conclude this paper by providing several examples of downgrading rules in the context of the entity-relationship model.

Keyword Codes: D.4.6; F.4.1; H.2.0
Keywords: Security and Protection; Database Management, General; Mathematical Logic

INTRODUCTION

Most of the multilevel database management systems (DBMS) only deal with static data classification and do not provide general mechanisms to manage the evolution of the classifications during the course of time. However, in many applications, we need functions to automatically downgrade pieces of information when these pieces of information become obsolete.

This paper proposes a preliminary analysis of this problem in order to define the general framework and functions necessary for a DBMS to deal with downgrading requirements. For this purpose, we adopt a logical approach. A database is represented by a logical first order theory and we include in this formalism several extensions to represent both concepts of temporal and classified data. The example we shall use in this paper are then presented using concepts similar to the entity-relationship model. However, our formalism may apply to a relational database as well.

This paper is organized as follows. Section 1 is a general overview of the downgrading problem; it informally describes several types of downgrading, namely on an order, an event, a specific time or after a delay. Section 2 sums up the main problems we have to solve. Section 3 proposes a theoretical analysis of the downgrading problem. Our objective in this section is to define a general language which enables the multilevel database content and general downgrading rules to be specified. Sections 4 and 5 show how to use this language by providing several examples of downgrading rules. Finally, section 6 concludes this paper and describes several continuations we plan for this work.

1 VARIOUS TYPES OF DOWNGRADING

1.1 Downgrading on an order

The already available multilevel systems generally implement a function which enables the content of an object (in the sense of the Bell and LaPadula notion of object [BL75]) to be downgraded **on an order**. The ability to play a special role, namely the role of downgrader, is generally required to execute this downgrading function. We may also assume that, for instance, the system security administrator is the only person cleared to play the role of downgrader.

1.2 Downgrading on an event

Downgrading on an order allows us to represent interactive downgrading requirements. However, in many applications, we need means to specify general rules which enable information to be automatically downgraded. Downgrading on an event is an example of automatic downgrading which is triggered off when an event external to the database occurs. The following rule is an example of downgrading on an event:

Rule 1: When John retires, his salary is to be downgraded

To deal with this kind of downgrading rules, the DBMS must provide means to filter and recognize some specific events, for instance a database update, in order to automatically trigger off the downgrading rule.

1.3 Downgrading on a time

This type of downgrading corresponds to a decision to downgrade a given piece of information on a specific time. The following rule is an example of downgrading on a time:

Rule 2: On January 1, 1995, John' salary is to be downgraded

1.4 Downgrading after a delay

Many organizations define security policies to protect their data, but generally consider that the sensitivity associated with a given data is not a static but dynamic attribute.

Therefore, this data may be downgraded after a given delay making the old sensitivity obsolete. The following rule is an example:

Rule 3: After ten years, Secret data are to be downgraded

2 PROBLEMS TO SOLVE

2.1 Specifying downgrading rules

A first problem on which this paper focuses is that we need a language to specify general rules for automatic downgrading. As far as we know, such a language has never been proposed before.

The general downgrading rules are to be managed by the multilevel DBMS and only specific users are to be cleared to write the downgrading rules. Users cleared at this position must be very accurate when writing the downgrading rules. For instance, let us reconsider the natural language specification of rule 1:

Rule 1: When John retires, his salary is to be downgraded

When inserting this rule in the DBMS, this user must accurately specify the condition that will trigger off the downgrading of John's secret salary. This is to avoid that a Trojan Horse, by observing this downgrading rule, maliciously inserts in the database that John retires, when actually he does not, in order to disclose his secret salary. For this purpose, this user must specify, in the pre-condition of the rule, that the piece of information saying that John retires is to be inserted on behalf of a trusted user.

On the other hand, the DBMS must be able to recognize the situation in which a given piece of information is to be downgraded. Therefore, downgrading seems to be related to several concepts used in active databases [DBBa88],[WCL91], in particular the concepts of action and event. We shall further discuss this point in the conclusion. In this paper, we actually follow a logical approach and shall take our inspiration in [AF94] in order to represent these concepts of events and actions in our formalism. The aim of section 3 will be to develop such a language and propose a theoretical analysis of the logical properties of this language.

2.2 Downgrading consequences

Another theoretical problem we have to consider is how to propagate data downgrading. As a matter of fact, the following rule must be enforced when data downgrading is performed:

Rule 4: If p is a fact to be downgraded at level l,
If $p \rightarrow q$ is a general rule classified at level l,
If q is classified at level l' with $\neg(l' \leq l)$,
Then q is also to be downgraded at level l

If this rule is not enforced, then an inferential channel is created in the multilevel database: a user cleared at level l can use information stored in the database to which he can legally have an access - namely p and $p \rightarrow q$ - to derive prohibited information - namely q.

However, applying the rule 4 may lead to downgrade some pieces of information whereas the user in charge of controlling the downgrading process does not plan to do so. Therefore, we have to be careful when applying this rule. A good practice is to ask the user to confirm the downgrading decision in this case. Moreover, how to deal with the consequences of a downgrading decision is actually related to a bigger problem, namely the inference problem in multilevel databases. This is a complex problem and we do not plan to solve it in this paper.

2.3 Data sealing

When dealing with a downgrading rule after a delay, we are faced to the problem of sealing the information to be downgraded. For instance, let us assume that it is decided to downgrade at time t_3 a fact dated at time t_1; let us also assume that the decision to downgrade is taken at time t_2 with $t_1 \leq t_2 < t_3$. In this case, it may be possible to update the fact to be downgraded between t_2 and t_3. In our approach, this update would cancel the decision to downgrade. Therefore, if one really wants to downgrade this information, the DBMS must provide mechanisms to guarantee that the data to be downgraded is not modified between t_2 and t_3. We call sealing this type of trusted mechanisms. When a given piece of information is sealed, then all attempt to modify this piece of information must be prohibited. On the other hand, the sealing procedure does not enforce any additional restriction when a read access is requested. Thus, implementing a sealing procedure simply requires specific access controls to be effective.

3 A LANGUAGE TO SPECIFY DOWNGRADING RULES

3.1 Principles

The aim of this section is to propose a language to specify various types of downgrading rules, namely on order, event or time. This language is based on a first-order logic with several extensions whose purpose is to enable downgrading rules to be specified.

First order logic has been used to formalize databases in two main ways usually called the proof theoretic approach and the model theoretic approach. The former considers a database as a first order theory, the latter considers a database as an interpretation of a first order theory. The logical approach has been first applied to the relational model [Rei83]. In this paper, we adapt these ideas to the case of the entity-relationship model [Che76].

The remainder of this section is organized as follows: In section 3.2, we propose a logical interpretation of several entity-relationship concepts using first order logic. Then, in the subsequent sections we show how to extend this language to include several other concepts such as concepts of dating, classification and downgrading. We propose a syntax as well as an axiomatics for these concepts. Due to space limitation, we do not develop a general semantic for this language; it will be presented in a forthcoming paper.

All these concepts may seem to be unnecessarily complicated, however we claim that all of them are necessary to formally deal with the various types of downgrading we consider in this paper. Thus, section 3.3 starts by including a representation of temporal data suited to the downgrading problem. Section 3.4 then extends this language to associate formulas with a multilevel classification. Section 3.5 shows how to represent a database whose purpose is to store both temporal and multilevel data. Section 3.6 includes the concept of event we shall use to specify downgrading rules on an event. Finally, section 3.7 shows how to represent the fact that a given data must be downgraded at a given date and how to use this language to define various types of downgrading.

3.2 Entity-Relationship Model

Language and Axiomatics

The language we use to represent the database content is based on a first order logic with equality. This language, we denote it L_1 is defined by a pair (A_1, W_1) where A_1 is a set of symbols and W_1 is a set of well formed formulas. A_1 includes the following symbols:

- A set of variables V and a set of constants C. A term is a symbol which belongs to $V \cup C$.
- A set of predicates $\mathcal{P}$. $\mathcal{P}$ must include the binary predicate $=$ (equality).
- The logical symbols $\rightarrow$ (implication), $\wedge$ (conjunction), $\vee$ (disjunction) and $\neg$ (negation), $\leftrightarrow$ (equivalence).
- The universal quantifier $\forall$ and existential quantifier $\exists$.

The construction rules of the well formed formulas of language L_1 are the following:

- If P is a n-ary predicate and if $t_1, ..., t_n$ are terms then $P(t_1, ..., t_n)$ is a (atomic) formula.
- If P_1 and P_2 are formulas then $(P_1 \wedge P_2)$, $(P_1 \vee P_2)$, $(P_1 \rightarrow P_2)$, $(P_1 \leftrightarrow P_2)$ and $\neg P_1$ are formulas.
- If x is a variable and P a formula, then $\forall x, P$ and $\exists x, P$ are formulas.

The axiomatics of language L_1 are classical axiom schemas of first order logic with equality.

Entities and Classes

We now show how to use this logical framework to represent the main concepts of the entity-relationship model. We need first to model the concept of entity (or object). For this purpose, the set of constants C must include a set of object identifiers. An identifier is used to uniquely identify an object. The set of predicates $\mathcal{P}$ must also include a set of unary class predicates. Objects belong to these classes. For instance, the data "the object whose identifier is O_1 is an instance of the class Employee" is represented in our logical theory by the fact $Employee(O_1)$.

The set of class predicates is structured in a sub-class and super-class hierarchy. For instance, the fact that the class C_1 is a sub-class of class C_2 is represented in our logic by the following rule:

$$\forall x, C_1(x) \rightarrow C_2(x)$$

Finally, there is a class which is a super-class of all others classes. For this purpose, we

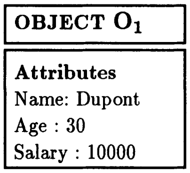

Figure 1 Example of object

introduce a special unary class predicate denoted *Object*. Every existing object is an instance of the class *Object*.

Values and Attributes

The data related to an object are represented by a set of attributes. Hence, the set of constants C must include a set of attribute values and the set $\mathcal{P}$ must include a set of attribute predicates. For instance, let us assume that an employee is associated with the attributes *Name*, *Age* and *Salary*. Therefore, to represent an employee in our logic, we shall use the three following predicates: $Name(x,y)$, $Age(x,y)$ and $Salary(x,y)$. For each class C and for each attribute $A_1, ..., A_n$ associated with this class, there is a rule to specify that each object of this class has a value associated with the attributes $A_1, ..., A_n$:

$$\forall x, C(x) \rightarrow \exists y_1, ..., \exists y_n, A_1(x,y_1) \wedge ... \wedge A_n(x,y_n)$$

Finally, we can use this language to represent the following employee O_1 (Figure 1). It is represented in our logic by the following axiom:

$$Employee(O_1) \wedge Name(O_1, Dupont) \wedge Age(O_1, 30) \wedge Salary(O_1, 10000)$$

The language we have just described is sufficient for the purpose of this paper. We refer to [CD89] for a more detailed presentation which includes the concept of types, complex object values and inheritance. It is somewhat straightforward to specify all these concepts in first order logic but it is not necessary for the purpose of this paper.

3.3 Temporal Data

Topological logic

We shall say that a statement is temporally definite if its truth or falsity is independent of the time at which it is asserted. For instance, the following statements are temporally definite:

- It sometimes rains in Paris.
- It always rains in Brest.
- It was raining in Paris on January 1, 1980.

By contrast, let us consider the statements:

- It is now raining in Toulouse.
- It was raining in Paris yesterday.

These statements are all temporally indefinite in that their truth or falsity is not independent of their time of assertion. After these preliminary remarks it is useful to distinguish between dates and pseudo-dates. A definite date is a time specification that is chronologically stable (such as "January 1, 1980"). A pseudo-date is a time specification that is chronologically unstable (such as "today" or "yesterday"). In the following, we shall consider a temporal dating procedure based on a set of chronological stable dates, but it is also possible to establish a correspondance with one unstable pseudo-date: *now*. We assume that the set of stable dates is a discrete set associated with a total order denoted $\leq$. This means that we consider a linear time structure rather than a branching time structure. The temporal logic we obtain is closely related to the so-called "topological" temporal logic [RU71]. This logic does not include the two classical temporal operators *since* and *until* [GM91]. We are now presenting the syntax and axiomatics of this logic.

Let P be some temporally indefinite statement. Then, we can in general form another statement asserting that P holds at the particular time t. Therefore, we introduce the modal operator $[\ ]$ of temporal realization. We shall write $[P]_t$ to be read "P is realized at the time t". If t is a proper date (not a pseudo date), then $[P]_t$ is always temporally definite. We have supposed up to now that P is a temporally indefinite statement. We can drop this restriction by means of the following convention:

> If P is a temporally definite statement, then $[P]_t$ is to be taken simply as equivalent with P itself. Therefore, if P is temporally definite, then $P \leftrightarrow \forall t, [P]_t$.

We are now interesting in the iteration of the $[\ ]$ operator. What is the meaning of statement such as $[[P]_t]_{t'}$, i.e "it is the case at time t' that it is the case at time t that P". There are actually two cases depending on whether t is or not equal to n (i.e *"now"*):

$$[[P]_t]_{t'} \leftrightarrow \begin{cases} [P]_{t'} & \text{if } t = n \\ [P]_t & \text{if } t \neq n \end{cases}$$

Syntax and axiomatics

The language $L_2 = (A_2, W_2)$ we consider is an extension of the language L_1 previously defined. A_2 must include the following symbols:

- All the symbols of language L_1.
- The set C must contain a set of definite dates and the pseudo date n (*"now"*).
- The set V must contain a set of temporal variables denoted t, t', etc... A temporal term is a date or a temporal variable.
- A binary predicate $\leq$ of chronological ordering.
- The operator $<>$ to form interval of time.
- The modal operator $[\ ]$ of temporal realization.

The construction rules of the formulas of language L_2 are the following:

- All the construction rules of language L_1.
- If P is a formula and t_1 and t_2 are two temporal terms then $[P]_{t_1}$ and $[P]_{<t_1,t_2>}$ are formulas.

The axiomatics of language L_2 is the following axiom schemas*:

- All axioms of language L_1.
- Axioms to specify that $\leq$ is a total order.
- $[\neg P]_t \leftrightarrow \neg[P]_t$.
- $[P_1 \wedge P_2]_t \leftrightarrow [P_1]_t \wedge [P_2]_t$.
- $[P]_n \leftrightarrow [P]$.
- $[\forall t' P]_t \leftrightarrow \forall t'[P]_t$, if t and t' are two distinct temporal variables.
- $[[P]_t]_{t'} \wedge \neg(t = n) \leftrightarrow [P]_t$.
- $[n = t']_t \leftrightarrow t = t'$.
- $[t' = t'']_t \leftrightarrow t' = t''$.
- $\forall t P \rightarrow P^{t/t_0}$ where t_0 must be a date or a pseudo date and P^{t/t_0} designates the result of substituting t_0 for every free occurence of t in P. Furthermore, in this last axiom, t must not occur within the scope of the operator [ ] in P.

The inference rules of language L_2 are the modus ponens plus:

- $\frac{\vdash P}{\vdash \forall t, [P]_t}$

Finally, $[P]_{<t_1,t_2>}$ is simply an abbreviation for:

$$[P]_{<t_1,t_2>} \leftrightarrow \forall t, (t_1 \leq t \wedge t \leq t_2) \rightarrow [P]_t$$

i.e. the truth of P upon the interval $< t_1, t_2 >$ is equivalent to the truth of P in every time t belonging to the closed interval $< t_1, t_2 >$.

3.4 Data classification

For the sake of simplicity, we only consider two classification levels: the unclassified (or public) level and the secret level. Defining a classification procedure actually consists in labelling some formulas of a given language. In our case, we consider, at every time t, two sets U_t and S_t of sentences of L_2 which respectively represent the set of unclassified and secret sentences. At every time t, it is assumed that U_t and S_t are disjoint sets, that is $U_t \cap S_t = \emptyset$ and it is also assumed that some sentences of L_2 may be not classified†, that is, we do not necessarily have: $U_t \cup S_t = L_2$. The labelling procedure of some sentences of L_2 with a classification level is a priori independent from the truth or falsity of this sentence. To specify in our logic which sentences are classified, we shall consider the language $L_3 = (A_3, W_3)$ which is an extension of language L_2. A_3 must include the following symbols:

- All the symbols of language L_2.
- The modal operators $[U]$ and $[S]$.

The construction rules of language L_3 are the following:

*We assume that the temporal variables are distinct from other variables. This assumption associated with these construction rules actually leads to a two-sorted logic. This is to avoid the existence of strange and undesirable formulas in our language

†i.e. are not explicitly associated with a classification level

- All the construction rules of language L_2.
- If P is a formula of L_2 then $[U]P$ and $[S]P$ are formulas of L_3.

Intuitively, $[U]P$ means that the formula P is unclassified and $[S]P$ means that the formula P is secret. Formulas having the form $[U]P$ and $[S]P$ are temporally indefinite statements (even though P is temporally definite). By contrast, if t is a date, then $[[U]P]_t$ and $[[S]P]_t$ are two temporally definite statements and we have:

- If $P \in U_t$ then $[[U]P]_t$
- If $P \in S_t$ then $[[S]P]_t$

There are other axiom schemas (see also [CD94] for a more detailed presentation) to define general properties of the classification procedure, namely:

- $[U](true)$ where *true* is an abbreviation for $P \vee \neg P$
- $\frac{\vdash P \rightarrow Q}{\vdash [U]P \rightarrow [U]Q}$

The first axiom says that tautologies should be unclassified; the second axiom says that if $P \rightarrow Q$ is a tautology then P is unclassified should imply that Q is also unclassified. For $[S]$, it is simply the converse which holds, namely:

- $\neg[S](true)$
- $\frac{\vdash P \rightarrow Q}{\vdash [S]Q \rightarrow [S]P}$

The first axiom says that tautologies should not be secret; the second axiom says that if $P \rightarrow Q$ is a tautology then Q is secret should imply that P is also secret.

Finally, let us consider the following formula:

- $\neg([U]P \wedge [S]P)$

i.e. any formula should not be both unclassified and secret. We consider that this constraint is a part of the multilevel security policy. Therefore, in the following, we shall assume that this is a theorem of our logic.

3.5 Temporal databases

Assumptions

We consider a database denoted *db* and assume that the database content is a set of formulas of the language L_2. This means that the database may contain temporal data. We also consider that the database content changes in the course of time. We note db_t the database content at time t. We shall use the following definitions (see also [Sri93, J^+94]):

Definition 1 We call classical database a database *db* which only contains temporally indefinite data.

Definition 2 We call valid-time database or historical time database a database which may contain temporally definite data, i.e the database content db_t of the database at time t is represented by statements having the form $[P]_{t'}$ with possibly $t \neq t'$. t is called instant valid-time or instant historical time and t' is called instant transaction-time or instant belief time. Notice that we may have $t' > t$, that is to say the database may contain future statements.

Definition 3 We call transaction-time database or belief time database a database which may contain at time t all the database contents db_{t_i} with $t_i \leq t$. This means that a transaction-time database records all its past beliefs.

We can actually consider that a classical database (i.e a database which does not contain temporal data) is a special case of valid-time database whose content at time t is uniformly dated at t. It is assumed that db_t is a consistent set of formulas. On the other hand, it is not assumed that all the data stored in db are true of the world, that means that db_t is considered as a set of beliefs. It is assumed that the global database db is actually separated into two sub-databases, the unclassified database denoted udb and secret database denoted sdb. We shall call "unclassified database" at time t a consistent set of sentences udb_t and "secret database" at time t a consistent set of sentences sdb_t.

Language extension

We want to represent in our language all the sentences derivable from the (deductive) database db, udb and sdb. For this purpose, we consider an extension $L_4 = (A_4, W_4)$ of language L_3 which must contain three modal operators $[DB]$, $[UDB]$, $[SDB]$ and we consider the following construction rules:

- All construction rules of language L_3.
- If P is a formula then $[DB]P$, $[UDB]P$ and $[SDB]P$ are formulas.

Intuitively, $[DB]P$, $[UDB]P$ and $[SDB]P$ respectively means that the global database, the unclassified database and the secret database believes P. We assume that $[B]P$ (where $[B]$ may be any of the modalities $[DB]$, $[UDB]$ and $[SDB]$) is temporally indefinite. By contrast, if t is a date, then the formula $[[B]P]_t$ is temporally definite and means that P is derivable from the consistent set of sentences db_t (or udb_t, or sdb_t). Actually, we have:

$$[B]P \leftrightarrow [[B]P]_n$$

So, $[B]P$ represents that P is derivable from the set of sentences db_n (or udb_n, or sdb_n).

Axiomatics

The modalities $[DB]$, $[UDB]$ and $[SDB]$ are three doxastic modalities whose axiomatics is the KD logic [Che80]. Therefore, we get the following axiomatics (where $[B]$ represents any of the modalities $[DB]$, $[UDB]$ or $[SDB]$):

- $[B]P \wedge [B](P \rightarrow Q) \rightarrow [B]Q$
- $\neg[B](false)$
- $\frac{\vdash P}{\vdash [B]P}$

The first axiom says that the database beliefs are closed under logical derivation. The second axiom says that each database content is a consistent set of beliefs and the third axiom says that each database believes all the tautologies.

Moreover, there are two axiom schemas to derive the global database from the unclassified and secret databases (see [CC95] for a more detailed presentation)[‡]:

- $[SDB]P \rightarrow [DB]P$
- $[UDB]P \wedge \neg[SDB]\neg P \rightarrow [DB]P$

The first axiom says that the secret database is a sub-part of the global database. On the other hand, the unclassified database may contain cover stories; a cover story is a lie provided to unclassified users to hide the existence of a more secret fact. Therefore, the second axiom says that the global database will believe in an unclassified fact only if this fact is not contradicted by an otherwise secret fact.

Let us now consider the following formula:

- $[UDB]P \rightarrow [U]P$

This formula says that the unclassified database only contains data labelled unclassified. We shall consider that this constraint is a part of the multilevel security policy. Therefore, in the following, we shall assume that this is a theorem of our logic[§]. Notice also that we do not assume that the following formulas:

- $[U]P \rightarrow [UDB]P$
- $[S]P \rightarrow [SDB]P$

are theorems of our logic because we consider that the labelling procedure may be a priori independant from what the databases believe. This allows us to represent general classification rules such as:

$\forall x, \forall y, Employee(x) \wedge Sal(y) \rightarrow [S]Salary(x, y)$

This rule classifies at secret the set of formulas having the form $Salary(x, y)$ for each x instance of class $Employee$ and for each y of type Sal. On the other hand, for a given employee, only one secret formula having the form $Salary(x, y)$ can be stored in the secret database. This is because we can assume that the salary of an employee is unique.

Finally, to reduce length of further formulas, we shall use the following abbreviation:

$[SECRET]P \leftrightarrow [S]P \wedge [DB]P$

If $[SECRET]P$ is true, we say that P is a *true secret*. This last definition is more general than the following one:

$[SECRET]P \leftrightarrow [S]P \wedge [SDB]P$

For instance, let us consider that we have $[UDB]P$ and $[SDB](P \rightarrow Q)$ and $\neg[SDB]\neg P$. Then, using the above axioms we shall derive that $[DB]P$ and $[DB](P \rightarrow Q)$ and therefore $[DB]Q$ because the global database is closed under logical derivation. In this case, using our definition of true secret, we shall consider that Q is a true secret if we have $[S]Q$, even though Q is not stored in the secret database.

[‡]Notice that the second axiom is only available in the case of atomic formulas. In the general case, there are problems with this axiom (see [GLQS92] for a discussion)

[§]Notice that from this last theorem and from the theorem $\neg([U]P \wedge [S]P)$, we can also derive $[S]P \rightarrow \neg[UDB]P$

3.6 Events

Language extension

We want to include in our language the possibility to specify that some event occurs, for instance updating some data in the database or giving a downgrading order. For this purpose, we shall consider the following extension $L_5 = (A_5, W_5)$ of language L_4. A_5 must contain the following symbols:

- All the symbols of language L_4.
- A set of action symbols $\mathcal{A}$.
- A set of subjects $\mathcal{S}$.
- The modal operator $Perform$.

The construction rules of language L_5 are the following:

- All the construction rules of language L_4.
- If s is a subject and α an action then $Perform(s, \alpha)$ is a formula.

Intuitively $Perform(s, \alpha)$ means that the subject s performs the action α. Examples of actions we shall consider are: *Insert*, *Downgrade*, etc... For the sake of simplicity, we only consider atomic actions. We also assume that $Perform(s, \alpha)$ is a temporally indefinite statement.

Axiomatics

The theory must include specific axioms associated with each action to describe the effects of the action on the database content. For instance:

$$[Perform(s, Insert(P))]_t \rightarrow [[DB]P]_{t+1}$$

This axiom says that the effect of inserting a sentence P in the database at time t is that the database will believe P at time $t+1$. Actually, it would be more correct to consider that the database believes P at time $t+d$ where d represents the execution time of the *Insert* operation. However, for the sake of simplicity, we shall consider, in the following, that execution time of every action is equal to $d = 1$.

3.7 Downgrading rules

Finally, we want to include in our language the possibility to specify downgrading rules. For this purpose, we consider the extension $L_6 = (A_6, W_6)$ of language L_5 which must include the modal operator $[D]$ and the following construction rule:

- If P is a formula of language L_2, then $[D]P$ is a formula.

$[D]P$ means that the formula P is labelled with a *downgrading marking.* This label means that the formula P is to be downgraded to the unclassified level. We only consider two security levels, the secret and the unclassified levels. To consider more than two levels and to be able to specify downgrading to level unclassified, confidential, etc, we would have simply to parameterize the modal operator $[D]$ with a classification level. This extension is straightforward.

Of course, we consider that $[D]P$ is temporally indefinite, but if t is a date then $[[D]P]_t$ is temporally definite. $[[D]P]_t$ means that the formula P is labelled with a *t-downgrading marking.* This label means that the formula P is to be downgraded to the unclassified level at the time t.

General format of downgrading rules

General format of a downgrading rule is the following:

$$Condition \rightarrow [[D][Info]_{t_1}]_{t_2}$$

where *Condition* is any formula of language L_5 and *Info* is any formula of language L_2.

Such a rule means that once decision to downgrade is taken (i.e once *Condition* becomes true), *Info* dated at time t_1 is labelled with a t_2-downgrading marking. Consequence of such a downgrading rule is the following:

$$[[SECRET][Info]_{t_1}]_{t_2} \wedge [[D][Info]_{t_1}]_{t_2} \rightarrow [[UDB][Info]_{t_1}]_{t_2+1}$$

This rule says that downgrading a piece of information $[Info]_{t_1}$ at a date t_2 means that the unclassified database will believe in $[Info]_{t_1}$ at the date $t_2 + 1$[¶]. Notice also that, due to the theorem $[UDB]P \rightarrow [U]P$, the fact that $[Info]_{t_1}$ becomes unclassified at time $t_2 + 1$ is also a consequence of donwgrading $[Info]_{t_1}$.

This rule also means that to be effectively downgraded at the date t_2, $[Info]_{t_1}$ must be a true secret at the date t_2. This is because we consider that the labelling procedure (with secret, unclassified or downgrading markings) is a priori independant from what the databases believe.

Different types of downgrading rules

Definition 4 Let us consider the following downgrading rule :

$Condition \rightarrow [[D][Info]_{t_1}]_{t_2}$

- If $t_1 = t_2$ then this rule is a downgrading rule **on a specific time**.
- If $t_1 < t_2$ then this rule is a downgrading rule **after a delay**.
- If condition of the rule includes a term like $Perform(s, downgrade([Info]_{t_1}, t_2))$, then this rule is a downgrading rule **on an order**[||].
- If condition of the rule includes a term like $Perform(s, \alpha)$, where α is an action performed by user s then this rule is a downgrading rule **on an event**. Notice downgrading on an order is a special case of downgrading on an event.
- If t_2 is a constant then this rule is a **constant time** downgrading rule.
- If t_2 is a universally quantified variable then this rule is a **variable time** downgrading rule.

[¶]Indeed downgrading is complete at time $t_2 + 1$ since it requires one unity of time to insert $[Info]_{t_1}$ in the unclassified database (see section 3.6)

[||]$Perform(s, downgrade([Info]_{t_1}, t_2))$ intuitively means that the subject s requests the system to downgrade $[Info]_{t_1}$ at time t_2

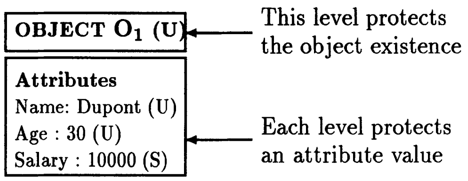
OBJECT O_1 (U)
Attributes
Name: Dupont (U)
Age : 30 (U)
Salary : 10000 (S)
This level protects the object existence
Each level protects an attribute value

Figure 2 Multilevel object

Sealing

Definition 5 Sealing information which must be downgraded, in the context of a temporal database which manages both valid and transaction time means prohibiting belief revision about this information in the secret database until the downgrading date.

We consider that sealing information labelled with a downgrading marking until its downgrading date is not mandatory. However, if we want to be sure that this information will be downgraded, then this information must be sealed until the downgrading date. This information is sealed at the downgrading decision date if it is a true secret at this date, and is sealed as soon as it is a true secret otherwise. Of course if this information is not true secret at any time between the downgrading decision date and the downgrading date then this information cannot be sealed and then is not downgraded.

Notice we say "sealing an information means *prohibiting belief revision about this information in the secret database until the downgrading date*". This does not imply "*preventing the unclassified database to believe such information before the downgrading date*". Indeed the information to downgrade at time t_2 may be downgraded before this instant time t_2 by another downgrading rule. In this case, downgrading the information at time t_2 is canceled and sealing of this information until the time t_2 is canceled as well.

In order to formally deal with sealing in our language, we would have to consider an action *Sealing* belonging to the set of actions $\mathcal{A}$ and a modal operator *Sealed*. If P is a formula of language L_2 and s a subject then $Perform(s, Sealing(P))$ is to be read "the subject s performs the action of sealing the sentence P". $[Sealed]P$ means that the formula P is labelled *Sealed*. Using this extending language, it would be easy to specify the sealing axiomatics. However, for the sake of simplicity, as we do not use this axiomatics in the remainder of this paper, we do not present it here.

4 DOWNGRADING IN AN ENTITY-RELATIONSHIP DATABASE

4.1 Multilevel entities

A *multilevel* object (see Figure 2) is an object whose identifier is associated with a security level and attributes are separately classified. Each security level assigned to an attribute is to protect the value of this attribute. The security level assigned to the object identifier is to protect the object existence. This security level must be dominated by the greatest lower bound of all the security levels assigned to the attributes of this object.

This example clearly shows that two types of information can be protected:

- Existence of an object.
- Value of an attribute.

Thus, we have to define two different mechanisms to perform downgrading of each kind of information.

4.2 Entity existence downgrading

As we assumed there are only two security levels (the secret and unclassified levels), to be able to study object existence downgrading, we must consider an object whose existence is secret. The fact that the existence of object O_1 at time t_1 is a true secret at time t_2 is expressed using the language defined in section 3 as follows:

$$[[SECRET][Object(O_1)]_{t_1}]_{t_2}$$

Suppose now, we want to downgrade at the instant time t_2 that object O_1 exists at time t_1. Such a decision is expressed as follows:

$$[[D][Object(O_1)]_{t_1}]_{t_2}$$

Consequence of such a downgrading decision is as follows:

$$[[SECRET][Object(O_1)]_{t_1}]_{t_2} \wedge [[D][Object(O_1)]_{t_1}]_{t_2} \rightarrow [[UDB][Object(O_1)]_{t_1}]_{t_2+1}$$

Notice that downgrading of the objects existence does not imply downgrading of its attribute values. Attribute values of object O_1 remain secret.

4.3 Attribute value downgrading

The sentence "the name attribute value *"Dupont"* of O_1 at time t_1 is a true secret at time t_2" is expressed as follows:

$$[[SECRET][name(O_1, Dupont)]_{t_1}]_{t_2}$$

Suppose now that we want to downgrade at instant time t_2 the attribute name *"Dupont"* of object O_1 at time t_1. Such a decision is expressed as follows:

$$[[D][name(O_1, Dupont)]_{t_1}]_{t_2}$$

Consequence of such a downgrading are the following:

$$[[SECRET][name(O_1, Dupont)]_{t_1}]_{t_2} \wedge [[D][name(O_1, Dupont)]_{t_1}]_{t_2}$$
$$\rightarrow [[UDB][name(O_1, Dupont)]_{t_1}]_{t_2+1}$$

Notice that downgrading an object attribute value also discloses the existence of this object. This is a consequence of downgrading (see section 2.2). However, from the security administrator point of view, it is perhaps safer to require the existence of the object to be first downgraded before downgrading an attribute value of this object. This is a decision we shall have to make during the implementation phase.

5 EXAMPLES OF DOWNGRADING RULES

In this section, we show how to use the language described in section 3 to specify different types of downgrading rules.

5.1 Constant time downgrading rules

Constant time downgrading rule on a specific time
Let us consider the following rule expressed in natural language:

Rule 5: On January 1, 1995, salary of O_1 on January 1, 1995 must be downgraded

This rule may be expressed as follows in using the language we defined in section 3:

$$\forall x, [[SECRET][salary(O_1, x)]_{1/1/95}]_{1/1/95} \rightarrow [[D][[salary(O_1, x)]_{1/1/95}]_{1/1/95}$$

Notice condition of this rule checks if information to downgrade is a true secret at time 1/1/95. This is not mandatory because, as we have seen in section 3.7, this is performed anyway just before downgrading the information. Thus the rule could have been equally stated as follows:

$$\forall x, [[DB][salary(O_1, x)]_{1/1/95}]_{1/1/95} \rightarrow [[D][[salary(O_1, x)]_{1/1/95}]_{1/1/95}$$

Constant time downgrading rule after a delay
Let us consider the following rule:

Rule 6: On January 1, 1995, salary of O_1 on January 1, 1994 must be downgraded

The two following rules are two possible translations of the previous rule in natural language:

$$\forall x, [[SECRET][salary(O_1, x)]_{1/1/94}]_{1/1/94} \rightarrow [[D][salary(O_1, x)]_{1/1/94}]_{1/1/95} \tag{1}$$

$$\forall x, [[SECRET][salary(O_1, x)]_{1/1/94}]_{1/1/95} \rightarrow [[D][salary(O_1, x)]_{1/1/94}]_{1/1/95} \tag{2}$$

Although the rule expressed in natural language does not seem ambiguous, it may lead to several interpretations once we try to translate it in logic. To speak truly, it is possible to have several interpretations of the *condition* of the rule. There is only one solution to express the consequence of the rule.

In rule 1, decision to downgrade is taken at time 1/1/94. Thus $[salary(O_1, x)]_{1/1/94}$ must be sealed until the date 1/1/95, if we want to be sure that it will be downgraded. In rule 2, decision to downgrade is taken at time 1/1/95, thus $[salary(O_1, x)]_{1/1/94}$ does not need to be sealed since the downgrading date is also 1/1/95.

5.2 Variable time downgrading rules

Variable time downgrading rule on an event
Let us consider the following downgrading rule:

Rule 7: When the object O retires, Downgrade his salary

This rule may be translated in our language as follows:

$$\begin{aligned}
&\forall x, \forall t_1, \forall t_2,\\
&[Perform(s, Insert([job(O, retirement)]_{t_1}))]_{t_2} \wedge [Trusted(s)]_{t_2} \wedge\\
&[[SECRET][salary(O,x)]_{t_1}]_{t_2} \wedge t_1 \leq t_2\\
&\quad \rightarrow [[D][salary(O,x)]_{t_1}]_{t_2} \qquad (3)
\end{aligned}$$

Notice that the sentence "When the object O retires" is translated in our language by the event $[Perform(s, Insert([job(O, retirement)]_{t_1}))]_{t_2}$. Notice also that the fact "object O retires" may be not immediately inserted in the database when this fact occurs in the real world, but perhaps after a delay, namely at time t_2 with $t_2 \geq t_1$. This insert must be performed on behalf of a trusted subject to prevent a Trojan Horse from disclosing a secret piece of information by maliciously inserting in the database that object O retires.

Finally, as the decision to downgrade the salary of O coincides with the downgrading date, then it is not necessary to seal the salary of O.

Variable time downgrading rule on an order

Let us consider the following rule:

Rule 8: A user cleared to play the role of downgrader can order the downgrading of any pieces of information.

This rule may be translated in our formal language as follows:

$$\begin{aligned}
&\forall t_1, \forall t_2, \forall t_3,\\
&[Perform(s, downgrade([Info]_{t_1}, t_3))]_{t_2} \wedge [Role(s, downgrader)]_{t_2} \wedge t_2 \leq t_3\\
&\quad \rightarrow [[D][Info]_{t_1}]_{t_3} \qquad (4)
\end{aligned}$$

This rule means "if an agent s cleared to play the role of downgrader orders at any instant time t_2 to downgrade at the instant time t_3 a given piece of information $Info$ dated at instant time t_1, then this piece of information is labelled with a downgrading marking at time t_3. In this rule, we actually assume that only a trusted user can play the role of downgrader in order to avoid the problem mentioned in section 5.2.1.

This rule is a variable downgrading rule since t_3 is universally quantified, it is also on an order and after a delay if $t_1 < t_3$. Moreover, we should have $t_2 \leq t_3$ so that the downgrading order is prior to the downgrading date.

Finally, if $[Info]_{t_1}$ is a true secret at time t_2, then $[Info]_{t_1}$ may be sealed between t_2 (the date of the downgrading decision) and t_3.

Variable time downgrading rule after a delay

Previous downgrading rules show us how to specify various and complex downgrading rules which may be difficult to express in natural language. To conclude this section, let us consider a rule which seems to be very clear when expressed in natural language:

Rule 9: After ten years, secret information must be downgraded

However, an ambiguity appears once we try to express it in our language. Actually, two solutions seem possible when translating this sentence into a logical rule:

$$\forall t_1, \forall t_2, [Perform(s, (Insert[Info]_{t_1}))]_{t_2} \wedge [[S][Info]_{t_1}]_{t_2} \rightarrow [[D][Info]_{t_1}]_{t_2+10} \quad (5)$$

$$\forall t_1, \forall t_2, [Perform(s, (Insert[Info]_{t_1}))]_{t_2} \wedge [[S][Info]_{t_1}]_{t_2} \rightarrow [[D][Info]_{t_1}]_{t_1+10} \quad (6)$$

The condition is the same for rules 5 and 6. It means "agent s inserts in the database, at time t_2, $[Info]_{t_1}$ which is labelled secret at time t_2". In rule 5, decision is to downgrade $[Info]_{t_1}$, ten years after having inserted it in the database. In rule 6, decision is to downgrade $[Info]_{t_1}$, ten years after time t_1.

We guess that solution proposed by rule 6 seems to best correspond to the rule expressed in natural language. Notice however that a problem may arise if $t_2 > t_1 + 10$. This means information would have to be downgraded before the database believes it ! A solution proposed by rule 5, despite that it does not correspond exactly to the rule in natural language, can however corresponds to a need.

6 CONCLUSION

In this paper, we developed a logical language which enables general automatic downgrading rules to be specified. We used this language to specify downgrading rules in the context of a multilevel entity-relationship database. We have shown in particular that this language is powerful enough to specify downgrading after a delay, on an event, an order and a specific time. There are several issues to this work:

1. Data management needs to be further investigated. In particular, we could have shown how to manage temporal objects in using time intervals [SC91], but space prohibited development of this issue in this paper. We also plan to extend the MultiView model [BCCGY93] to manage temporal objects and downgrading mechanisms. Other temporal features would also require further investigations [SA86, PMY94, P$^+$94]. In particular, the concept of event would be required to be slightly refined to include the possibility to specify actions having variable execution times.
2. We plan to translate the modal logic used to specify downgrading rules into a classical logic, and designing a tool based on PROLOG. Even though we have shown that modal logic is of interest to specify downgrading rules, we think that downgrading rules can be translated in classical logic. This would enable powerful tools such as a PROLOG interpretor to be used. Moreover, a tool based on PROLOG enables a temporal database to be easily simulated [Sri93].
3. Implementation of downgrading rules in an OODB seems to be possible but would require more work than designing a tool based on PROLOG. Indeed many problems must be solved – in particular, time management in the OODBMS, implementation of deduction mechanisms to infer on downgrading rules, implementation of sealing and downgrading mechanisms. It is however important to notice that some DBMS such as active DBMS supply some features which seem to be appropriate to implement downgrading rules. Most of the prototypes such as HiPAC [DBBa88] or Starburst [WCL91] for instance, support different kinds of events (like time events, events based

on data updates, or external events). These events are likely to fit very well to our downgrading rules.

ACKNOWLEDGEMENT

This work was supported by the STEI. We would also like to thank Nora Boulahia-Cuppens, Gérard Eizenberg and the anonymous referees for their helpful comments.

REFERENCES

[AF94] J. F. Allen and G. Fergusson. Actions and Events in Interval Temporal Logic. *Journal of Logic and Computation*, 1994.

[BCCGY93] N. Boulahia-Cuppens, F. Cuppens, A. Gabillon, and K. Yazdanian. Multi-View Model for Multilevel Object Oriented Databases. In *Ninth Annual Computer Security Applications Conference*, Orlando, Florida, 1993.

[BL75] D. Bell and L. LaPadula. Secure Computer Systems: Unified Exposition and Multics Interpretation. Technical Report ESD-TR-75-306, MTR-2997, MITRE, Bedford, Mass, 1975.

[CC95] L. Cholvy and F. Cuppens. Providing Consistent Views in a Polyinstantiated Database. In J. Biskup, M. Morgenstern, and C. Landwehr, editors, *Database Security, 8: Status and Prospects*. North-Holland, 1995. Results of the IFIP WG 11.3 Workshop on Database Security.

[CD89] F. Cuppens and R. Demolombe. How to recognize topics to provide cooperative answering. *Information Systems*, 14(2), 1989.

[CD94] F. Cuppens and R. Demolombe. Normative Conflicts in a Confidentiality Policy. In *ECAI'94 Workshop on Artificial Normative Reasoning*, Amsterdam, The Netherlands, 1994.

[Che76] P. P. Chen. The Entity / Relationship Model: toward a unified view of data. *ACM TODS*, 1(1), March 1976.

[Che80] B. F. Chellas. *Modal Logic.* Cambridge University Press, 1980.

[DBBa88] U. Dayal, B. Blaustein, A. Buchmann, and al. The HiPAC Project : Combining Active Databases and Timing Constraints. *SIGMOD RECORD*, 17(1), March 1988.

[GLQS92] T. Garvey, T. Lunt, X. Qian, and M. Stickel. Toward a Tool to Detect and Eliminate Inference Problems in the Design of Multilevel Databases. In *Proc. of the Sixth IFIP WG 11.3 Working Conference on Database Security*, Vancouver, 1992.

[GM91] D. Gabbay and P. McBrien. Temporal Logic and Historical Databases. In *Proceedings of the 17th International Conference on Very Large Data Bases*, Barcelona, Spain, 1991.

[J$^+$94] C. Jensen et al. A Consensus Glossary of Temporal Database Concepts. *SIGMOD RECORD*, 23(1), March 1994.

[P$^+$94] N. Pissinou et al. Towards an Infrastructure for Temporal Databases: Report of an Invitational ARPA/NSF Workshop. *SIGMOD RECORD*, 23(1), March 1994.

[PMY94] N. Pissinou, K. Makki, and Y. Yesha. On Temporal Modeling in the Context of Object Databases. *SIGMOD RECORD*, 22(3), September 1994.

[Rei83] R. Reiter. Towards a logical reconstruction of relational database theory. In *On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases and Programming Languages.* Springer Verlag, 1983.

[RU71] N. Resher and A. Urquhart. *Temporal Logic.* Springer Verlag, 1971.

[SA86] R. Snodgrass and I. Ahn. Temporal Databases. *IEEE Computer*, 19(9), September 1986.

[SC91] S. Su and H.-H. Chen. A Temporal Knowledge Representation Model OSAM*/T and its Query Language OQL/T. In *Proceedings of the 17th International Conference on Very Large Data Bases*, Barcelona, 1991.

[Sri93] S. M. Sripada. A Metalogic Programming Approach to Reasoning about Time in Knowledge bases. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, 1993.

[WCL91] J. Widom, R. Cochrane, and B. Lindsay. Implementing set-oriented production rules as an extension to Starbust. In *Proceedings of Int. Conf. on VLDB, Barcelona*, 1991.