9

A Logical Approach to Model a Multilevel Object Oriented Database

Frédéric Cuppens
ONERA-CERT
2 Av. E. Belin
31055, Toulouse Cedex
France
email: cuppens@tls-cs.cert.fr

Alban Gabillon
Eastern Mediterranean University
Computer Engineering Department
Famagouste
Mersin 10, Turkey
alban@compenet.compe.emu.edu.tr

Abstract

In the context of OODB (Object-Oriented DataBases), several multilevel security models appeared in the literature. In this paper, we are mainly interested in the MultiView model [BCCGY93, BCCGY94a]. Our first objective is not to propose a new security model but rather to present the MultiView model in a *formal* way. Another objective is to *extend* the MultiView model to include new security functionalities, in particular the protection of the database schema. Our approach can be sum up as follows. We propose a language based on first-order logic to represent an OODB content and specify integrity constraints which must be enforced in an OODB. A first security model called Single-View is then defined. In this security model, every atomic formula of the language used to represent the OODB is a piece of information we may associate with a classification level. We also derive general theorems which must be enforced when classifying each piece of information. Finally, we show how to refine the Single-View model to obtain the MultiView model. We give a sketch of implementation of the MultiView model.

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

INTRODUCTION

One major area in DBMS is the security of shared data stored and manipulated via various operations in the database. Database security aims to maintain both the confidentiality and integrity of information in the database by restricting access to authorized persons and operations only. In this paper, we only pay attention to the multilevel confidentiality policy. Intuitively, a given multilevel confidentiality policy assigns a clearance level to subjects and a classification level to sentences of the language used to represent the database content. It is generally assumed that the set of levels is a lattice. According to a multilevel policy a subject is permitted to know sentences stored in the database whose classification is lower or equal to his clearance, and he is forbidden to know sentences whose classification is greater than or not comparable with his clearance. In the context

P. Samarati et al. (eds.), *Database Security*

of object oriented databases, several multilevel security models appeared in the literature. Our objective here is actually not to propose a *new* model but rather a more *complete* model. Our model, briefly stated, includes the following features:

- Management of multi-level objects. This means that each object attribute is associated with its own classification level.
- Protection of object existence. This means that each object is associated with a classification level used to hide the object existence.
- Management of attribute value polyinstantiation. This functionality refers to the simultaneous existence of multiple values for the same object attribute, where the multiple values are distinguished by their classification levels.
- Protection of class existence as well as existence of class attributes and methods.
- Protection of the sub-class and super-class hierarchy. This means that a link of type *isa* may be associated with a classification level.
- Protection of the fact that an object is instance of a given class. This means that a link of type *instance_of* may be also associated with a classification level.

This model is a *formal* model based on first order logic. This formal approach allows us to analyze the effects of classifying a given piece of information upon the classification of other pieces of information stored in the database. General constraints that must be enforced when classifying the database content are formally derived as theorems.

However, our point of view is that directly implementing this model would be impractical as such. This is mainly due to the fact that there are many problems associated with a direct implementation of the so called multilevel objects (see [Lun90] for a discussion). Therefore, we suggest refining this model. Our objective in refining is to decompose the multilevel object oriented database into a collection of single level databases whose behaviors are quite similar to classical non-protected object oriented databases. The main difference is that we suggest using dynamic links between each single level database. Intuitively, these dynamic links avoid replication of the same information at several security levels and make automatic the propagation of a low level update to the higher levels.

The remainder of this paper is organized as follows. Section 1 proposes a formalization of the main elements of the object-oriented data model without dealing with security. We do not consider that this model is complete but it is sufficient for the definition of a formal security model for OODB. In particular, we give in section 1 a number of axioms from which theorems regarding the multilevel security will be derived. In section 2, we develop a model for a secure multilevel object oriented database system. We call this model Single-View because all the information related to a given real-world entity is encapsulated into a single multilevel object. The axioms given in section 1 are used to derive the implications of classifying a given piece of information in an object oriented database. Section 3 proposes a refinement of the model described in section 2. This leads to another formal model called MultiView. The Single-View and MultiView models provide the same means of protection but we guess that MultiView is easier to implement without using trusted enforcement mechanisms in the object layer. Section 3 also informally shows through an example how the MultiView model deals with object creation, consultation, updating and deletion and how to activate a method. In section 4, we compare our approach with related work and section 5 is a brief summary of the approach proposed in this paper.

1 NON PROTECTED MODEL

The aim of this section is to propose a language to specify the content of an object-oriented database. This language is based on first-order logic. First order logic has been used to formalize relational databases in two main ways usually called the proof theoretic approach and the model theoretic approach. The former considers a relational database as a first order theory, the latter considers a relational database as an interpretation of a first order theory. In both cases, the logic approach introduces capabilities of general rule representation which is useful either for database integrity checking or for automatic deduction of data. In the context of object oriented databases, it is less obvious to use first order logic as a formal language. It seems that several concepts such as methods require higher order logics or modal logics (dynamic logic for instance) to be properly formalized [Wie91]. In this section, our goal is not to develop such a complete formalism for object-oriented database. The formalism we use simply introduces all the concepts we need to formally define our model for a multilevel object-oriented database. For a more complete formalization, see for instance [MQ93].

1.1 Language

The language we consider is based on a first order logic with equality. In order to represent an object oriented database using this language, we shall use eight predicates:

- Two one place predicates $Object$ and $Class$.
- Five two place predicates CA, OA, $Method$, $Instance$, Isa.
- One three place predicate Val.

Intuitively, these predicates are respectively to be read:

- $Object(o)$: "o is an object".
- $Class(c)$: "c is a class".
- $CA(c, a)$: "a is an attribute of the class c".
- $OA(o, a)$: "a is an attribute of the object o".
- $Method(c, m)$: "m is a method of the class c".
- $Instance(o, c)$: "o is an instance of the class c".
- $Isa(c, c')$: "c is a sub-class of the class c'".
- $Val(a, o, v)$: "v is the value of the attribute a in the object o".

1.2 Axiomatics

The axiomatics of language L is classical axiom schemas of first order logic with equality, plus the proper axioms of our theory. These axioms may be viewed as a set of integrity constraints to be enforced by the object-oriented database:

- If a is an attribute of the class c then c is a class.
  $\forall a \forall c, CA(c, a) \rightarrow Class(c)$ (1)
- If a is an attribute of the object o then o is an object.
  $\forall a \forall o, OA(o, a) \rightarrow Object(o)$ (2)
- If m is a method of the class c then c is a class.
  $\forall m \forall c, Method(c, m) \rightarrow Class(c)$ (3)

- Any object attribute has a value.
  $\forall a \forall o, OA(o,a) \leftrightarrow \exists v, Val(o,a,v)$ (4)
- The value of an object attribute is unique.
  $\forall a \forall o \forall v \forall v', Val(o,a,v) \wedge Val(o,a,v') \rightarrow (v = v')$ (5)
- If o is an instance of c then o is an object and c is a class.
  $\forall o \forall c, Instance(o,c) \rightarrow Object(o) \wedge Class(c)$ (6)
- The two arguments of the Isa predicate must denote a class.
  $\forall c \forall c', Isa(c,c') \rightarrow Class(c) \wedge Class(c')$ (7)
- Isa is respectively antisymetric, antireflexive and transitive.
  $\forall c \forall c', Isa(c,c') \rightarrow \neg Isa(c',c)$ (8)
  $\forall c, \neg Isa(c,c)$ (9)
  $\forall c \forall c' \forall c'', Isa(c,c') \wedge Isa(c',c'') \rightarrow Isa(c,c'')$ (10)
- Inheritance property 1. Any attribute a which belongs to a class c' is inherited in any sub-class c of c'.
  $\forall c \forall c' \forall a, Isa(c,c') \wedge CA(c',a) \rightarrow CA(c,a)$ (11)
- Inheritance property 2. Any method m which belongs to a class c' is inherited in any sub-class c of c'.
  $\forall c \forall c' \forall m, Isa(c,c') \wedge Method(c',m) \rightarrow Method(c,m)$ (12)
- Inheritance property 3. Any attribute a of an object o is inherited from a class the object o is instance of. Conversely, any attribute a of a class c is inherited by the objects which are instance of the class c.
  $\forall o \forall a, OA(o,a) \leftrightarrow \exists c, CA(c,a) \wedge Instance(o,c)$ (13)
- Any object o which is instance of a class c is also instance of all the super-classes of c.
  $\forall o \forall c \forall c', Instance(o,c) \wedge Isa(c,c') \rightarrow Instance(o,c')$ (14)
- Any object is instance of at least one class.
  $\forall o, Object(o) \rightarrow \exists c, Instance(o,c)$ (15)

Notice that we introduce the concept of method in a very simple syntactical definition. This definition will be sufficient to include in our security policy the protection of method existence. However, as the program associated with a method is not represented, our model does not enable this program to be explicitly protected.

1.3 Example of object-oriented database

Figure 1 presents the example we shall use in the remainder of this paper. There are three classes *Employee*, *Employee_to_Dismiss*, *Employee_Company_A*. We assume that *Employee_Company_A* is a sub-class of *Employee_to_Dismiss*; this means that every employee of company A is an employee to be dismissed. There are also two objects O_1 and O_2. O_1 is an instance of *Employee_Company_A* and O_2 is an instance of *Employee_to_Dismiss*. Both of them have four attributes: *Name*, *Salary* and *Religion* inherited from *Employee* and *Dismissal_Date* inherited from *Employee_to_Dismiss*.

2 SINGLE-VIEW MODEL

2.1 Principles

The Single-View model is a multilevel object oriented model in which all the information related to a given real-world entity is encapsulated in a single multilevel object. The central

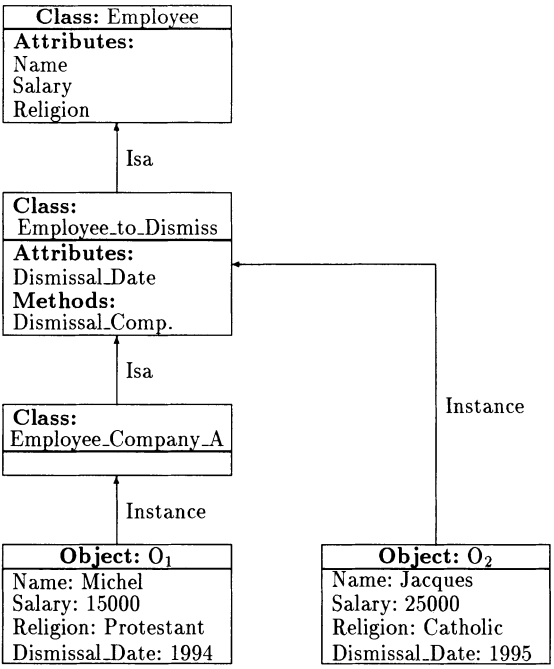

Figure 1 Example of object-oriented database

idea of our approach is that any piece of information represented by an atomic formula in the non-protected object oriented model described in section 1 may be associated with a classification level. Another feature of this model is to support cover stories in using the polyinstantiation technique. Therefore, we need to perform several extensions of the non-protected model so that sensitive information can be protected.

2.2 Levels

We first extend the language developed in section 1 to represent a finite set of classification levels. For this purpose, we shall use a one-place predicate *Level*. The formula *Level*(l) is to be read "l is a classification level". We assume that the set of levels is a lattice associated with a partial order relation denoted $\leq$. Therefore, the *least upper bound* and *greatest lower bound* are defined. For this purpose, we shall use two functions *lub* and *glb*. If l_1 and l_2 are two security levels, then $lub(l_1, l_2)$ and $glb(l_1, l_2)$ are respectively the least upper bound and greatest lower bound of l_1 and l_2. There is also a level which is lower than all other levels, we denote it $\bot$ and a level which is higher than all other levels, we denote it $\top$.

It would be straightforward to write the axioms associated with the predicates *Level* and $\leq$ which correspond to these assumptions. As we shall not use these axioms in the following, we do not give them here.

2.3 Multilevel database

Language

The language used to represent the multilevel database is an extension of the language developed in section 1. For each predicate P of arity n used to represent the non-protected database content, there is a predicate P' of arity $(n+1)$ used to represent the multilevel database. Therefore, we obtain eight multilevel predicates $Object'$, OA', Val', $Class'$, CA', $Method'$, Isa' and $Instance'$. Intuitively, if l is a classification level, then $P'(t_1, ..., t_n, l)$ is to be read "the piece of information $P(t_1, ..., t_n)$ is classified at level l".

Notice that we might also apply the classification process to the predicates $Level$ and $\leq$ introduced in section 2.2. For instance, this would lead to consider a two-place predicate $Level'$ with sentences $Level'(l, l')$ to be read "the fact that l is a classification level is itself classified at level l'". This extension is perfectly acceptable and enables information related to the lattice of classification levels to be protected. However, for the sake of simplicity, we do not include this possibility in our model. This implicitly means that we assume that the extension of the predicates $Level$ and $\leq$ are classified at the lowest level $\bot$.

Axiomatics of the extended logic

We add to the list of axioms (1)–(15) presented in section 1 the following axioms used to specify the links between the non–protected database and the multilevel database:

- For each n-place predicate P, we have the following axiom:
  $\forall t_1 ... \forall t_n, P(t_1, ..., t_n) \rightarrow \exists l, Level(l) \wedge P'(t_1, ..., t_n, l)$ (16)
  This axiom says that any fact of the non-protected database is classified in the multilevel database.
- For each n-place predicate $P' \neq Val'$, we have the following axiom:
  $\forall t_1 ... \forall t_n \forall l, P'(t_1, ..., t_n, l) \rightarrow Level(l) \wedge P(t_1, ..., t_n)$ (17)
  This axiom says that, if $P' \neq Val'$, then the extension of the predicate P' in the multilevel database is obtained by classifying facts belonging to the extension of the predicate P in the non-protected database.
- For the predicate Val', the axiom is different. This comes from the fact that we accept this predicate to be polyinstantiated.
  $\forall a \forall o \forall v \forall l, Val'(a, o, v, l) \rightarrow \exists v', Val(a, o, v') \wedge Level(l)$ (18)
  This axiom says that if the sentence $Val(a, o, v)$ is classified at level l in the multilevel database then there must exist a corresponding value v' associated with the attribute a of object o in the non-protected database. However, due to polyinstantiation, we may have $v \neq v'$.
- Moreover, to restrict the effect of polyinstantiation the following axiom must be stated.
  $\forall a \forall o \forall v \forall v' \forall l, Val'(a, o, v, l) \wedge Val'(a, o, v', l) \rightarrow (v = v')$ (19)
  This axioms says that, within a given level, the attribute value of an object must be unique, i.e. within a given security level, database consistency must be enforced.

Axiom (19) is the multilevel counterpart of the non-protected axiom (5). Notice that, we can also use the axiom (16) to derive multilevel counterpart to all other non-protected axioms in the list (1)–(15). For instance, there is the following multilevel counterpart to axiom (1):

$\forall a \forall c \forall l, CA'(c, a, l) \rightarrow \exists l', Class(c, l')$

Our next objective is to derive general constraints that must be enforced when classifying the database content. For instance, which condition must be satisfied when classifying $CA(c,a)$ at level l and $Class(c)$ at level l'? We now present these theorems.

Inference control theorems

When classifying any piece of information at a given classification level, the following inference control rule must be enforced:

Rule 1 Let $x_1,...,x_n$ be tuples of variables respectively compatible with the arity of predicates $P_1,...,P_n$. Let y be another tuple of variables compatible with the arity of Q. We assume that each variable in tuple y appears in at least one of the tuples $x_1,...,x_n$. If:

$$\forall x_1...\forall x_n, P_1(x_1) \wedge ... \wedge P_n(x_n) \rightarrow Q(y) \quad (20)$$

is an axiom of the non-protected object oriented database, then we can derive the following theorem in the multilevel object oriented database:

$$\forall x_1...\forall x_n \forall l_1...\forall l_n \forall l, P'_1(x_1,l_1) \wedge ... \wedge P'_n(x_n,l_n) \wedge Q'(y,l) \rightarrow l \leq lub(l_1,l_2,...,l_n)$$

If rule 1 is not satisfied, then a subject cleared at level $lub(l_1,...,l_n)$ can access every $P_i(x_i)$ and use the axiom* 20 to derive $Q(y)$. If the classification of $Q(y)$ is not lower or equal to $lub(l_1,...,l_n)$, then an inference channel which enables prohibited information to be disclosed is created. By combining rule 1 with some axioms given in section 1, we can derive the following theorems†:

- The sensitivity of the fact "a is an attribute of class c" dominates the sensitivity of the fact "c is a class" (Combine Rule 1 with axiom 1):
  $$\forall a \forall c \forall l \forall l', CA'(c,a,l) \wedge Class'(c,l') \rightarrow (l' \leq l) \quad (21)$$
- The sensitivity of "a is an attribute of object o" dominates the sensitivity of "o is an object" (Combine Rule 1 with axiom 2):
  $$\forall a \forall o \forall l \forall l', OA'(o,a,l) \wedge Object'(o,l') \rightarrow (l' \leq l) \quad (22)$$
- The sensitivity of "m is a method of class c" dominates the sensitivity of "c is a class" (Combine Rule 1 with axiom 3):
  $$\forall m \forall c \forall l \forall l', Method'(c,m,l) \wedge Class'(c,l') \rightarrow (l' \leq l) \quad (23)$$
- The sensitivity of "v is a value of the attribute a in object o" dominates the sensitivity of "a is an attribute of object o" (Combine Rule 1 with axiom 4. Notice that a second theorem can be derived from axiom 4. See Section 2.3.4):
  $$\forall a \forall o \forall v \forall l \forall l', Val'(o,a,v,l) \wedge OA'(o,a,l') \rightarrow (l' \leq l) \quad (24)$$
- The sensitivity of "object o is an instance of class c" dominates the least upper bound of "o is an object" and "c is a class" (Combine Rule 1 with axiom 6):
  $$\forall o \forall c \forall l \forall l', Instance'(o,c,l) \wedge Object'(o,l') \wedge Class'(c,l'') \rightarrow (lub(l',l'') \leq l) \quad (25)$$
- The sensitivity of "c is a sub-class of c'" dominates the least upper bound of "c is a class" and "c' is a class" (Combine Rule 1 with axiom 7):
  $$\forall c \forall c' \forall l \forall l', Isa'(c,c',l) \wedge Class'(c,l') \wedge Class'(c',l'') \rightarrow (lub(l',l'') \leq l) \quad (26)$$
- The sensitivity of "c is a sub-class of c''" is dominated by the least upper bound of "c is a sub-class of c'" and "c' is a sub-class of c''" (Combine Rule 1 with axiom 10):
  $$\forall c \forall c' \forall c'' \forall l \forall l' \forall l'', Isa'(c,c',l) \wedge Isa'(c',c'',l') \wedge Isa'(c,c'',l'') \rightarrow (l'' \leq lub(l,l')) \quad (27)$$

*we implicitly assume that this axiom is classified at the lowest level $\perp$

†Detailed demonstrations for all these theorems can be found in [Gab95].

- Similar to axiom (27) for *Instance* (Combine Rule 1 with axiom 14):
  $\forall c \forall c' \forall o \forall l \forall l' \forall l'',$
  $$Instance'(o,c,l) \wedge Isa'(c,c',l') \wedge Instance'(o,c',l'') \rightarrow (l'' \leq lub(l,l')) \quad (28)$$
- The sensitivity of "a is an attribute of class c" is dominated by the least upper bound of "c is a sub-class of c'" and "a is an attribute of class c'" (Combine Rule 1 with axiom 11):
  $$\forall c \forall c' \forall a \forall l \forall l' \forall l'', Isa'(c,c',l) \wedge CA'(c',a,l') \wedge CA'(c,a,l'') \rightarrow (l'' \leq lub(l,l')) \quad (29)$$
- Similar to axiom (29) for *Method* (Combine Rule 1 with axiom 12):
  $\forall c \forall c' \forall m \forall l \forall l' \forall l'',$
  $$Isa'(c,c',l) \wedge Method'(c',m,l') \wedge Method'(c,m,l'') \rightarrow (l'' \leq lub(l,l')) \quad (30)$$
- The sensitivity of "a is an attribute of object o" is dominated by the least upper bound of "o is an instance of c" and "a is an attribute of class c" (Combine Rule 1 with axiom 13. A second theorem can be derived from axiom 13. See Section 2.3.4):
  $$\forall c \forall a \forall o \forall l \forall l' \forall l'', Instance'(o,c,l) \wedge AC'(a,c,l') \wedge AO'(a,o,l'') \rightarrow (l'' \leq lub(l,l')) \quad (31)$$

Signaling control theorems

When classifying any piece of information at a given classification level, the following signaling control rule must be enforced if one wants to protect the existence of sensitive information:

Rule 2 Let $x_1, ..., x_n$ and $y_1, ..., y_p$ be tuples of variables respectively compatible with the arity of predicates $P_1, ..., P_n$ and $Q_1, ..., Q_p$ and let y be another tuple of variables. We assume that each variable in tuple y appears in at least one of the tuples $y_1, ..., y_p$ and each variable in tuples $y_1, ..., y_p$ appears in at least one of the tuples $x_1, ..., x_n, y$. If:

$$\forall x_1 ... \forall x_n, P_1(x_1) \wedge ... \wedge P_n(x_n) \rightarrow \exists y, Q(y_1) \wedge ... \wedge Q(y_p) \quad (32)$$

is an axiom of the non-protected object oriented database, then, we can derive the following theorem in the multilevel object oriented database:

$\forall x_1 ... \forall x_n \forall l_1 ... \forall l_n,$
$P'_1(x_1,l_1) \wedge ... \wedge P'_n(x_n,l_n)$
$\rightarrow \exists y \exists l'_1 ... \exists l'_p, Q'(y_1,l'_1) \wedge ... \wedge Q'(y_p,l'_p) \wedge lub(l'_1,...,l'_p) \leq lub(l_1,...,l_n)$

If rule 2 is not satisfied, then a subject cleared at level $lub(l_1, ..., l_n)$ can access every $P_i(x_i)$ and use the axiom 32 to derive the existence of the facts $Q(y_1), ..., Q(y_p)$ some of them being classified higher than $lub(l_1, ..., l_n)$. Therefore, a signaling channel which enables the existence of prohibited information to be disclosed is created. By combining rule 2 with the axioms given in section 1, we can derive the following theorems:

- This theorem states that if the sensitivity of the fact "a is an attribute of object o" is l, then it exists a value for this object attribute whose classification level l' is dominated by l. (Combine Rule 2 and axiom (4)):
  $$\forall a \forall o \forall l, OA'(o,a,l) \rightarrow \exists v, \exists l', Val'(o,a,v,l') \wedge (l' \leq l) \quad (33)$$
- By combining theorem (33) with theorem (24), we can derive another interesting theorem which says that if the fact that "a is an attribute of object o" is classified at level l, then a value for this attribute must be provided at level l.
  $$\forall a \forall o \forall l, OA'(o,a,l) \rightarrow \exists v, Val'(o,a,v,l) \quad (34)$$

- Similarly, by applying rule 2 to axiom (15) and by combining this result with theorem (25), we can derive the following interesting theorem which says that if o is an object whose existence is classified at level l, then a class c such that o is instance of c must be provided at level l.
  $\forall o \forall l, Object'(o,l) \rightarrow, \exists c, Instance'(o,c,l)$ (35)
- Finally, by also applying rule 2 to axiom (13) and by combining this result with theorem (31), we can derive the following theorem which says that if the fact that "a is an attribute of object o" is classified at level l, then there is a class c such that "o is instance of c" and "a is an attribute of c" are two facts classified at level l.
  $\forall o \forall a \forall l, OA'(o,a,l) \rightarrow \exists c, CA'(a,c,l) \wedge Instance'(o,c,l)$ (36)

2.4 Polyinstantiation

Attribute value polyinstantiation

Let us consider the theorem (34). It says that a subject which is permitted to observe the existence of an attribute of an object must be provided with a value for this attribute. Polyinstantiation is a classical technique generally used to enforce this requirement in the database. For instance, let us consider our example of object oriented database presented in section 1.3. Let us assume that:

$OA'(O_1, Salary, \bot) \wedge Val'(O_1, Salary, 15000, \top)$

i.e. the fact "$Salary$ is an attribute of object O_1" is classified at level $\bot$ whereas the fact "15000 is the value of attribute $Salary$ in the object O_1" is classified at level $\top$. Theorem (34) says that a value for this attribute must also be provided at level $\bot$, for instance $Val'(O_1, Salary, 10000, \bot)$. From a semantical point of view, the value 10000 may be viewed as a cover story, i.e a lie provided to subjects whose clearances are $\bot$ to hide the existence of the more sensitive value 15000.

Entity polyinstantiation

In [Lun91], another type of polyinstantiation, called entity polyinstantiation is described. [Lun91] focuses on relational database. In this context, entity polyinstantiation may occur when a relation contains multiple tuples with the same apparent primary key values, but having different classification for this apparent primary key. In the context of object oriented database, a similar problem occurs when a given object o is associated with at least two different classifications, i.e:

$Object'(o,l) \wedge Object'(o,l') \wedge l \neq l'$

A possible interpretation, suggested by [Lun91], would be to consider that the two facts $Object'(o,l)$ and $Object'(o,l')$ actually refer to two distinct entities in the external world. However, this would be against the philosophy of the object oriented model where the object identifier is usually used to uniquely identify a real world entity. Therefore, we claim that it is better to prevent entity polyinstantiation in object oriented database. In section 2.5, we discuss how this may be achieved.

Other types of polyinstantiation

A similar problem may arise with the predicate $Class$, i.e. we may have:

$Class'(c,l) \wedge Class'(c,l') \wedge l \neq l'$

For similar reasons, we claim that this should also be prevented.

Finally, the predicates CA and $Method$ may also suffer from polyinstantiation, i.e we may have:

$CA'(c, a, l) \wedge CA'(c, a, l') \wedge l \neq l'$

$Method'(c, m, l) \wedge Method'(c, m, l') \wedge l \neq l'$

In both cases, we may interpret this as follows. Let us assume that $l \leq l'$. In this case, there is a first definition of the attribute a and method m at level l and a second definition at level l' which *overrides* the definition at level l. However, as our model does not enable these definitions to be completely specified (this is because specification of typing for a class and program for a method are not included in our model), it is better in our model to also prevent this type of polyinstantiation.

2.5 Preventing polyinstantiation

Preventing attribute value polyinstantiation

The basic idea to prevent attribute value polyinstantiation was suggested in [SJ92]. It consists in introducing a special symbol denoted by *Restricted* as the possible value for an attribute. Intuitively, an attribute assigned with the symbol *Restricted* means that the value exists but is higher classified. This is captured in the following axiom:

$$\forall o \forall a \forall l, Val'(o, a, Restricted, l) \rightarrow \exists v \exists l', Val'(a, o, v, l') \wedge v \neq Restricted \wedge l < l' \quad (37)$$

Therefore, if the existence of an attribute of an object is classified at a given level l and if one wants to assign to this attribute a value classified at level l' with $(l' > l)$, then one has two possibilities:

1. Assign a cover story to the attribute value at level l if one wants to hide the existence of the higher classified value.
2. Assign *Restricted* to the attribute value at level l in the opposite case.

Notice that, in both cases, theorem (34) is satisfied.

Preventing other types of polyinstantiation

If P is an n-place predicate different from Val, then polyinstantiation of this predicate is prevented if one has:

$$\forall x_1 ... \forall x_n \forall l \forall l', P'(x_1, ..., x_n, l) \wedge P'(x_1, ..., x_n, l') \rightarrow l = l' \quad (38)$$

i.e. each fact $P(x_1, .., x_n)$ is uniquely classified.

From a more practical point of view, [SJ92] suggest that there are three basic techniques for enforcing this axiom.

1. *Classify the predicate at the lowest level* $\perp$. For instance, if we add:
   $\forall o \forall l, Object'(o, l) \rightarrow l = \perp$
   then entity polyinstantiation is eliminated. This axiom says that the existence of every object is classified at $\perp$.
2. *Partition the set of symbols used to represent the database content.* For this purpose, we can add to our language a two-place predicate *Sensitivity*, with sentence $Sensitivity(x, y)$ to be read "the sensitivity of symbol x is equal to y". Axioms associated with *Sensitivity* are the following:

   - The second argument of *Sensitivity* is a level.
     $\forall i \forall l, Sensitivity(i, l) \rightarrow Level(l)$ (39)

- The sensitivity of a symbol is unique.
  $\forall i \forall l \forall l', Sensitivity(i,l) \wedge Sensitivity(i,l') \rightarrow (l = l')$ (40)

Using this predicate, we can, for instance, specify that the symbols beginning with "S-" are secret symbol. This technique is an effective means to eliminate polyinstantiation if we add the following axiom:
$\forall i \forall l, Sensitivity(i,l) \rightarrow Sensitivity'(i,l,\bot)$ (41)
This axiom says that the fact "the sensitivity of symbol i is equal to l" is itself classified at level $\bot$. In this case, the DBMS can reject any attempt by a subject cleared at a given level to insert a sentence containing higher sensitive symbol. This enables polyinstantiation to be prevented. For this purpose, we have to add the following axiom:

- Axiom to eliminate polyinstantiation of predicates P' where P' is a n-place predicate different from Val.

$\forall t_1 ... \forall t_n \forall l, P'(t_1, ..., t_n, l)$
$\rightarrow \exists l_1 ... \exists l_n, Sensitivity(t_1, l_1) \wedge ... \wedge Sensitivity(t_n, l_n) \wedge (l = lub(l_1, ..., l_n))$ (42)
From axioms (40) and (42), it is easy to derive (38) as a theorem.

3. *Limit insertions to be done by trusted subjects.* A third way to eliminate polyinstantiation is to require that insertions are only done by a user cleared at level $\top$, i.e. a user to whom all the database content is visible. For instance, if we require that the database schema is created and modify by trusted subjects only, then this eliminates polyinstantiation of predicates $Class$, CA and $Method$.

As noticed in [SJ92], the best approach will depend upon the characteristics of the DBMS and the application.

2.6 Example

Figure 2 presents a multilevel database derived from the non-protected database proposed in section 1.3 and consistent with the axioms and theorems (16)–(36). We appeal the reader's attention on the following assumptions:

- There are three security levels $U = \bot$ (Unclassified), C (Confidential) and $S = \top$ (Secret) with $U < C < S$.
- The existence of *Emp_to_Dismiss* is confidential.
- The fact that *Emp_Company_A* is a sub-class of *Emp_to_Dismiss* is secret.
- The existence of the attribute *Religion* is secret in every class.
- The existence of object O_2 is confidential.
- The fact that O_1 is an instance of *Emp_to_Dismiss* is secret.
- The value of the attribute *Salary* is polyinstantiated in object O_2. 25000 is the actual value and 10000 stands for a cover story. In figure 2, this is represented by using a polyinstantiated set [KTT89], i.e. the value of *Salary* is assigned to a set of pair (salary value, classification level).
- In object O_1, the value of the attribute *Salary* is assigned to *Restricted* at level U. This means that unclassified users are permitted to know that a higher classified salary exists.
- Other types of polyinstantiation are managed as follows. It is a mixture of the three solutions described in section 2.5. We assume that axiom 42 is enforced to eliminate

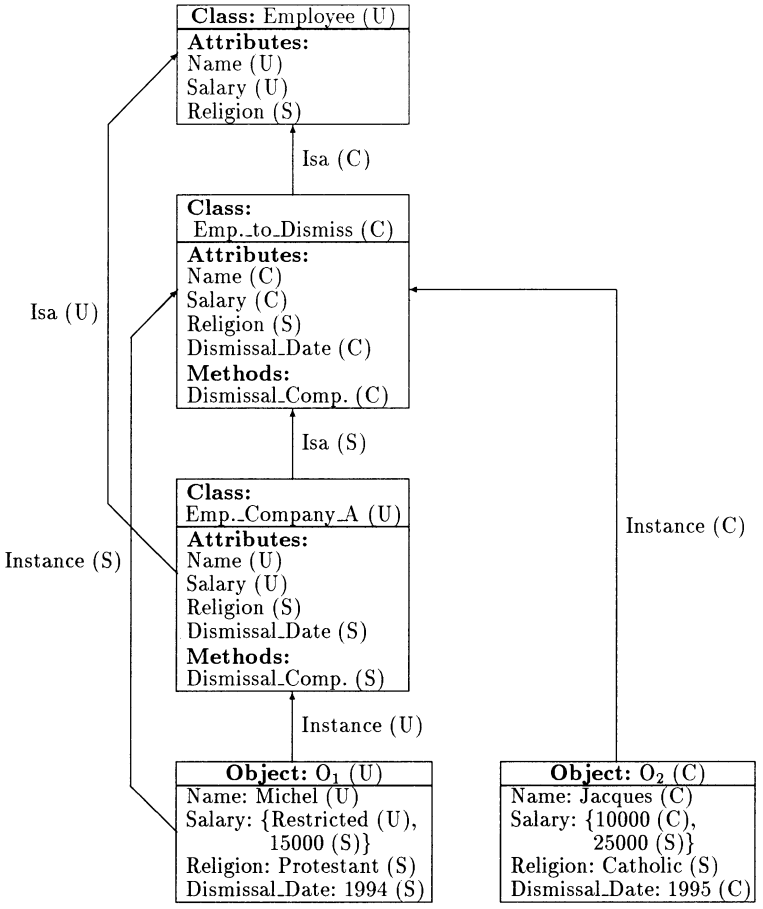

Figure 2 Example of multilevel object-oriented database

polyinstantiation of *Object* and we assume that the database schema is managed by trusted subjects to eliminate polyinstantiation of predicates *Class*, *CA* and *Method*.

We also provide the following comments to figure 2.

- Every attribute and method of *Emp_to_Dismiss* is classified at the level C or higher. This is because *Emp_to_Dismiss* is classified at C (Theorem 21 and 23).
- Similarly, every attribute of O_2 is at least C because O_2 is a confidential object (Theorem 22).
- $Instance(O_1, Emp\_to\_Dismiss)$ and $Instance(O_2, Emp\_to\_Dismiss)$ are at least C (Theorem 25).
- Similarly, *Isa(Emp_to_Dismiss,Employee)* and *Isa(Emp_Company_A,Emp_to_Dismiss)* are at least C (Theorem 26).

- As $Isa(Employee, Emp\_Company\_A)$ is U, the two attributes $Name$ and $Salary$ are also U in $Emp\_Company\_A$ (Theorem 29).
- For every object o and attribute a, the classification of $OA(o,a)$ is equal to the classification of $Val(o,a,v)$ if v is not polyinstantiated and to the lowest classification appearing in the polyinstantiated set otherwise (Theorem 34).
- As O_1 and O_2 are respectively U and C, Theorem (35) says that there must exist a class c and c' such that $Instance'(O_1, c, U)$ and $Instance'(O_2, c', C)$. In our example, this is true for $c = Emp\_Company\_A.$ and $c' = Emp\_to\_Dismiss.$

3 MULTIVIEW

3.1 Principles

The Single-View model is a complete but complex model and, as suggested in the introduction, we guess that it would be quite difficult to directly implement this model. In this section, we propose the MultiView model which may be viewed as a refinement of Single-View. In particular, MultiView should be easier to implement (see section 3.3 for an animation of this model).

Intuitively, this model is based on the decomposition of a n-level object oriented database into n views. Each view is associated with a given level of classification and contains all data whose classifications are lower or equal to the level of the view. This means that a data classified at a given level l in the Single-View model is replicated in every higher classified view. However, we shall show in section 3.3 how to avoid unnecessary replications.

In this section, we make the assumption that the set of levels is associated with a *total* order. The case of a partial order is discussed in section 3.7.

3.2 Language

The language used to represent the MultiView model is an extension of the language developed in section 2. For each n-place predicate P' used to represent the Single View database content, there is a n-place predicate P'' used to represent the MultiView database. Therefore, we obtain eight predicates $Object''$, OA'', Val'', $Class''$, CA'', $Method''$, $Instance''$ and Isa''. Intuitively, if l is a classification level, then $P''(t_1, ..., t_n, l)$ is to be read "the piece of information $P(t_1, ..., t_n)$ belongs to the view at level l" [‡].

Axiomatics of the extended logic

We add to the list of axioms (1)-(42), presented in section 1 and section 2, the following axioms used to specify the links between the Single-View and MultiView models:

- For each predicate P'' (except Val''), we have the two following axioms:
  $\forall t_1 ... \forall t_n \forall l \forall l', P'(t_1, ..., t_n, l) \wedge (l \leq l') \rightarrow P''(t_1, ..., t_n, l')$ (43)
  This axiom says that, if $P(t_1, ..., t_n)$ is a piece of information classified at level l, then this piece of information belongs to any view l' with $l' \geq l$.
  $\forall t_1 ... \forall t_n \forall l, P''(t_1, ..., t_n, l) \rightarrow \exists l', P'(t_1, ..., t_n, l') \wedge (l' \leq l)$ (44)

[‡]Notice the difference of interpretation between Single-View and MultiView. For instance $Object'(o, l)$ is to be read "the existence of object o is classified at level l" whereas $Object''(o, l)$ is to be read "the object o belongs to the view at level l".

This axiom says that, if $P(t_1, ..., t_n)$ belongs to the view at level l, then $P(t_1, ..., t_n)$ is classified at a level l' with $l' \leq l$.

- For the predicate Val'', axioms are slightly different. This is because we accept the predicate Val' to be polyinstantiated.
  $\forall a \forall o \forall v \forall l, Val'(a, o, v, l) \rightarrow Val''(a, o, v, l)$ (45)
  This axiom says that if the fact that $Val(a, o, v)$ is classified at level l, then there is a view of this fact at level l. We will see below (dynamic link axioms), how to also create the view at a given level l when there is no value v such that $Val'(a, o, v, l)$.
  $\forall a \forall o \forall v \forall l, Val''(a, o, v, l) \rightarrow \exists l', Val'(a, o, v, l') \wedge (l' \leq l)$ (46)
  This axiom says that, if a view $Val''(o, a, v, l)$ exists, then a corresponding piece of information $Val'(o, a, v, l')$ (with $l' \leq l$) must exist.
  $\forall o \forall a \forall v \forall v \forall l \forall l'$,
  $$Val''(a, o, v', l') \wedge Justbelow(l', l) \wedge \neg Val'(a, o, v, l) \rightarrow Val''(a, o, v', l)$$ (47)
  where $Justbelow$ is defined as follows:
  $\forall l \forall l', Justbelow(l', l) \leftrightarrow l' < l \wedge \neg(\exists l'', l' < l'' \wedge l'' < l)$
  Axiom (47) (Let us call it Dynamic link Axiom) says that if there is no value classified at level l for the attribute a of object o in the Single-View database, then the view at level l of the value of the attribute a is equal to the view of this value at level l' which is just below level l. Notice that, as we assume that the lattice of levels is associated with a total order, level l' is unique.

Notice that we can use axioms (43)–(47) to derive a MultiView counterpart to each axiom in the list (1)–(15)[§]. For instance, there is the following MultiView counterpart to axiom (1):
$\forall a \forall c \forall l, CA''(c, a, l) \rightarrow Class''(c, l)$
These theorems allow us to conclude that each single level view behaves like a classical non-protected database.

3.3 Animation

This section proposes a sketch of implementation of the MultiView model. We adopt the following implementation principles:

- We propose to decompose the MultiView database into several single level databases. Each l level view of the MultiView Model is implemented as a l level database.
- In MultiView, a multilevel object and a multilevel class are respectively represented by several object and class views. As a major principle of the object paradigm is the unicity of the object identifier, each object view or class view must be uniquely identified. We propose to implement this principle as follows. Each view at level l of a multilevel object o (respectively a multilevel class c) is uniquely identified by the pair (o, l) (respectively (c, l)).
- All single level class and object views having the same security level are stored into a single level object-oriented database. Each single level database can be managed as a non protected object-oriented database. The only restriction is that axiom 47 creates some dynamic links between the different single level databases.

[§]Demonstrations of these theorems can be found in [Gab95]

- We propose to implement the axiom 47 as follows. Any attribute value of an object view (o, l) such that there is no attribute value at level l in the polyinstantiated attribute of the corresponding multilevel object o, is equal to a pointer value. This pointer value is a syntactical expression which enables system to automatically retrieve the value of the same attribute in the view (o, l') such as l' is the security level which is just below the l security level.
- In [Gab95], we show that for all predicates P'' (except Val'') the following theorem can be easily derived from axioms (43) and (44):

  $$\forall t_1 ... \forall t_n \forall l \forall l', P''(t_1, ..., t_n, l) \wedge (l < l') \rightarrow P''(t_1, ..., t_n, l') \quad (48)$$

  It means in particular, that every class attribute and every method belonging to a low level view is replicated in all higher classified views. In order to implement dynamically this replication mechanism, we create an *Isa* Link from a high level view of any given class to the immediately lower view of this same class. Such *Isa* links enable low level attributes and low level methods in the high level views not to be replicated, but merely to be *inherited* by high level views.

Creation

Let us consider our example of section 2.6. As we consider three security levels (Unclassified, Confidential and Secret), the creation process of this example in the MultiView model is done in three phases. These three phases are respectively performed within unclassified, confidential and secret transactions. Figure 3 presents the database obtained after executing the three phases. In this figure, all pieces of information created by the unclassified, confidential and secret transactions are respectively written with a regular, *italic* and **bold** type style.

- Unclassified Transaction: The unclassified view of our example must be created within an unclassified transaction. We assume that the user can choose to perform a transaction at any level which is below his clearance level. Therefore, any user may create an unclassified transaction. The user may then create the unclassified view of our example in the unclassified database (see figure 3). Class view $(Employee, U)$ is inserted in the unclassified database and class view $(Emp\_Company\_A, U)$ is defined as a subclass of $(Employee, U)$. Object view $(O1, U)$ is created as an instance of $(Emp\_Company\_A, U)$. Object $O2$ and class *Emp_to_dismiss* do not appear in this unclassified database since their existence is confidential. Attribute *Religion* of the class *Employee* is not inserted in the unclassified database since its existence is secret. The *Salary* attribute of the object view $(O1, U)$ is equal to the special value *Restricted*. The unclassified view is then automatically replicated in higher classified databases to create higher classified views. This replication mechanism is performed as follows. Once it is committed, the unclassified transaction calls a confidential process and a secret process which respectively replicate this view at the confidential and secret levels¶. Actually, this unclassified view is not fully replicated in higher classified databases. Class attributes are not replicated but are merely inherited thanks to the four *Isa* links

¶We agree that such replication could be performed by several write-up during the commit of the unclassified transaction. These write-up are allowed by the multilevel security policy. However, we want to define a general mechanism of creation, and we shall see in the updating section that write-up are not convenient for object creation.

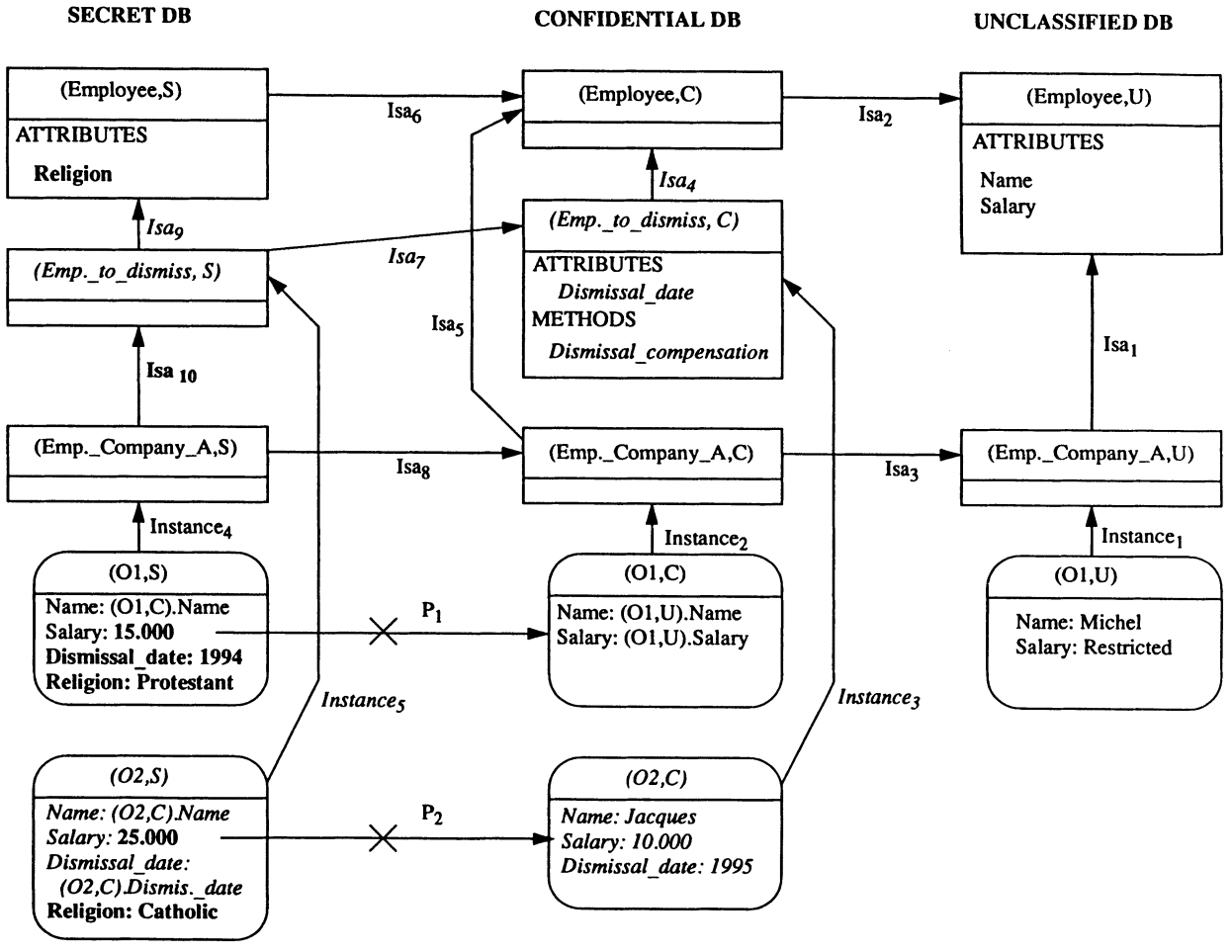

Figure 3: Example of MultiView DataBase

Isa_2, Isa_3, Isa_6 and Isa_8. The two links Isa_2 and Isa_3 are created by the confidential replication process whereas the two links Isa_6 and Isa_8 are created by the secret replication process. In the same way, attributes values of the object view $(O1, U)$ are directly pointed out by the object view $(O1, C)$ through the two dynamic pointers $(O1, U).Name$ and $(O1, U).Salary$ and are indirectly pointed out by the object view $(O1, S)$ through the two dynamic pointers $(O1, C).Name$ and $(O1, C).Salary$. Notice, this last pointer is represented in figure 3 by the crossed arrow P_1. This is because pointer $(O1, C).Salary$ will be removed by an update performed at the secret level (see below).

- Confidential Transaction: The confidential view of our example must be updated within a confidential transaction. This confidential transaction may be created by a user cleared at the confidential or secret level who chooses to work at the confidential level. Class view $(Emp\_to\_dismiss, C)$ is inserted in the confidential database as a subclass of the class view $(Employee, C)$. Object view $(O2, C)$ is created as an instance of $(Emp\_to\_dismiss, C)$. The value of the $Salary$ attribute of the object view $(O2, C)$ is equal to 10.000. This value actually stands for a cover story.
  These confidential updates are then automatically replicated in the secret database by a secret replication process called during the commit of the confidential transaction. This is quite similar to the replication processes called during the commit of the unclassified transaction.
- Secret Transaction: The secret view of our example must be updated within a secret transaction. This secret transaction may be created by a user cleared at the secret level only. Secret link Isa_{10} is introduced in the secret database. This link means that class view $(Emp\_Company\_A, S)$ is a subclass of $(Emp\_to\_dismiss, S)$. Thus, $(Emp\_Company\_A, S)$ inherits the *Dismissal_date* attribute and the *Dismissal_compensation* method. $(O1, S)$ becomes an instance of the $(Emp\_to\_dismiss, S)$ and $(Employee, S)$ class views. Thus attribute *Dismissal_date* appears in the secret view of the object $O1$. Secret attribute *Religion* is inserted in the secret $(Employee, S)$ class view. This attribute is inherited by all instances ($(O1, S)$ and $(O2, S)$) of this class view. Notice, the $Salary$ pointer values of the two object views $(O1, S)$ and $(O2, S)$ are removed and updated by a secret value.
  At the end of the secret transaction, creation of our example is complete.

3.4 Consultation

Let us consider a user who wants to query a view at level l of our example. Let us see the consultation process thanks to the MultiView model. To be able to consult the l single level database, the user must first set his working level to l and create a transaction at level l. Then the user can act on the l single level database and in particular see the l view of our example. The user sees the database schema which is evaluated with the l classified schema and the Isa links to lower classified schemas. Pointer attribute values are also evaluated and pointed values are then retrieved. Notice, we can imagine a mechanism which would inform the user of the classification level associated with any retrieved piece of information. A general technique would be to check all the lower classified databases to detect the lowest classified database in which a given piece of information is stored. The security level associated with this single level database corresponds to the security level associated with the given piece of information.

3.5 Updating and Deletion

In this section, we present updating and deletion processes in a non formal way (see [Gab95] for a formal presentation of these processes). So, let us consider a user who wants to update a view at level l of our example. Let us see the updating process thanks to the MultiView model. To be able to update this view, the user must first set his working level to l and create a transaction at level l. Updates which can be performed by this user are then the following:

- Schema updating: Updates of the schema at level l can be performed without any security restriction except that the policy used to eliminate polyinstantiation, if any, must be enforced.
- Attribute values updating: Updates of attribute values can also be performed without any security restriction. In the case where the updated value is a pointer, then the pointer is removed and a new value at level l is inserted. Notice any update performed on an attribute value is automatically propagated to highers levels thanks to the pointers from the high levels to the l level.
- Object creation: A new object can be inserted in our example by using the creation process described in section 3.3. We can now explain why a newly created object view at a low level cannot be replicated to higher levels by using the write-up technique. Let us consider for example an unclassified user who wants to create an instance of the class *Employee*. The secret view of this object could not be created by writing up since this view must be created with the attribute *Religion* which is not visible at the unclassified level. Therefore, the replication of a low level creation to a high level is to be performed by a high level process called by the low level transaction.
- Deletion: Let us show how to deal with object deletion. Similar comments apply to class deletion. We guess we can adopt two solutions depending on the required degree of integrity. Intuitively, with the first solution the user is obliged to delete all the views of a multilevel object whereas with the second solution, the user may delete the lowest level view of a multilevel object only. The two solution are investigated in [Gab95]. Whatever the solution adopted, to delete an object a user must first set his working level to the l security level protecting the object existence i.e to the l security level of the lowest level view of this object:

  - With the first solution the l level deletion process deletes the view at level l of the object and successively all the higher level views. This may be implemented by writing up the higher level views. However, high classified information concerning the deleted object are removed during this deletion process. This may be viewed as a threat regarding integrity.
  - In the second solution, the l level process only deletes the l level view of the object. However, due to this deletion, pointer values from the view at level $l+1$ to the view at level l become dangling references. This problem was first noticed in [BMJ94]. To avoid this problem, the l level deletion process must call a process at level $l+1$. This higher level process will replicate the pointed values of the view at level l to the view at level $l+1$. Notice that, as the deletion process at level l does not know which values are actually pointed at level $l+1$, this deletion process cannot perform the replication at level $l+1$ by writing up the view at level $l+1$.

Notice deletion of an object view which is different from the lowest level view would be conflicting with axiom 43.

3.6 Method Activation

Let consider a user who creates a transaction at a given security level l. This user can activate any method whose existence is classified at a level which is dominated by l. This method is always executed within the single-level transaction at level l and therefore may be viewed as a single level process. Hence, classical access controls such as the "no read up" and "no write down" of the Bell and LaPadula model apply to control the execution of a method.

Notice if a method tries to read a pointer attribute value, the corresponding pointed value stored in a lower classified database must be retrieved. This pointer may be directly evaluated if we accept to violate the encapsulation principle. If this principle is to be enforced, this value is retrieved by invoking the corresponding method at level $l-1$ but executed at level l. However, during this retrieve operation, a lock on the lower classified database may be used to protect the integrity of the retrieved value. This lock may be used by a method trapped with a Trojan horse to build a covert channel. This is a well known problem of multilevel database which is not specific to our approach. Many solutions were proposed to solve this problem (see for instance [KT90a, AJJ92, MJ93]). As an example, the retrieve operation may be performed on a snapshot of the lower classified database and not on the lower classified database itself [AJJ92].

3.7 Partial Order

We now analyze how to adapt the MultiView Model in the case of a partial order on the security levels. The creation process described in section 3.3 remains almost unchanged. The only restriction is explained in the following example. Let us consider the four security levels: U,$C1$,$C2$,S with $U < C1 < S$, $U < C2 < S$. $C1$ and $C2$ are not comparable. Let us also consider a multilevel object O whose existence is unclassified. We assume this object has one attribute whose existence and value are both unclassified. We denote this attribute a. Let us now consider a user who creates an unclassified transaction to insert this object in the unclassified view. At the end of this transaction, high level processes are initiated to replicate the unclassified view of the object in higher level databases. According to the axiom (47), the value of attribute a in views $(O, C1)$ and $(O, C2)$ is equal to the pointer $(O, U).a$. A problem now arises when creating the pointer of the attribute a in the secret view (O, S) since two security levels are *just below* the secret security level. Should this pointer be equal to $(O, C1).a$ or $(O, C2).a$? At this stage, this is not very important since in both cases, we would actually obtain the unclassified value. However, we are in trouble if the view of the attribute a at level $C1$ or $C2$ is updated. In this case, the simplest solution is to consider that the choice of the correct pointer is to be made by the user. Another more elaborated solution proposed in [CC95] would be to use the concept of topic (see [CD89]) to automatically determine which view is to be pointed out.

4 COMPARISON WITH RELATED WORKS

It is interesting to compare our approach in this paper with other models proposed in the literature.

SODA [KTT89, KT90b] is one of the earliest attempt to develop an object oriented model. SODA provides means to support multilevel objects and attribute value polyinstantiation by using polyinstantiated sets. Roughly speaking, SODA may be viewed as a direct implementation of a simplified version of the Single-View model.

As noticed in the beginning of this paper, we guess that approaches based on the decomposition of multilevels objects into single level objects leads to models easier to implement. There are several proposals in this direction [JK90, ML92, BJ93]. [BJ93] suggests representing multilevel objects using aggregation of single-level objects. However, a drawback of this approach is that it requires to modify the schema and to rewrite the methods. In [JK90], Jajodia and Kogan try also to demonstrate that it is possible to represent multilevel objects by using single level objects. In particular, they introduce the concept of *security inheritance*. This mechanism is identical to classical inheritance but is used solely for the purpose of representing multilevel objects by a set of single-level objects. Their motivations in this case are quite similar to ours in section 3.3.

In [BCCGY94b], we suggest combining an approach based on the decomposition objects with a virtual view machanism. This leads to a two-level architecture. At the bottom level, there are single-level objects. At the second level, there are virtual views which references to objects of different security levels. Such a view machanism could be used on top of other approaches based on single-level objects.

A comparison of several existing object oriented is proposed in [OS94]. For this purpose, [OS94] develops a formal model and some axioms of the object-oriented model are given from which theorems regarding secure database models are derived. This approach is actually close to ours. However, our model is more complete. In particular, [OS94] does not include the possibility to protect the links *Isa* and *Instance* and considers that an object attribute is simply a facet. Therefore, the value of an object attribute is not explicitly protected and the effect of attribute value polyinstantiation is not investigated. It is also not the purpose of [OS94] to develop a possible implementation for its model.

There are also some connections between our concept of "dynamic links" and the "derive option" proposed in [HOT91] to design a multilevel relational database. The derive option permits user to require that the value of a lower tuple is automatically copied up into the attribute of a higher tuple. However, using this option in the relational model deeply complicates the retrieval of data from the database. In the object-oriented-model, down-pointing data references may be specified using an access path which enables lower classified attribute values to be directly retrieved.

Finally, we can notice that most of the concepts that we first described in [BCCGY93, BCCGY94a] and that we formalize in this paper were reused by Schaefer *et al.* [SMKL95] in their implementation of a trusted ONTOS DBMS.

5 CONCLUSION

A first objective in this paper is to develop a *formal* model for multilevel OODB. Definition of this formal model is done in three phases:

1. *Non-Protected Model.* We define a language based on first order logic which allows us to represent an OODB content as well as integrity constraints which must be enforced in any OODB.

2. *Single-View Model.* We extend our language to be able to protect each piece of information represented by an atomic formula. We also derive general theorems that must be enforced when classifying the database content.
3. *MultiView Model.* We refine the Single-View model by decomposing the multilevel database into a collection of single level views. We also show how to implement the MultiView Model by associating each view with a single level database.

A second objective of this paper is to develop a *complete* model for multilevel OODB. This is mainly achieved as follows:

1. We include in our language, predicates that enable an OODB content to be fully represented. A Class, an object, a class attribute, an object attribute, an attribute value, a method, an isa link, an instance link are pieces of information which are represented in our language and therefore can be protected. In particular, this enables the database schema to be protected.
2. We introduce in our security model the possibility to manage cover stories in using the so called polyinstantiation technique.

One limitation to the expressive power of our model is that, for the sake of simplicity, we do not deal in this paper with the notion of *grain of classification* we introduced in [BCCGY94a]. This notion was used to represent attribute tuple values as well as attribute set values. However, we guess that it would not cause insuperable problems to also include this notion in our formal model.

ACKNOWLEDGEMENTS

This work was carried out with the support of the STSIE (Convention No. 9373584). Also, comments by Nora Boulahia-Cuppens on earlier draft of this paper have resulted in a number of improvements.

REFERENCES

[AJJ92] P. Ammann, F. Jaeckle, and S. Jajodia. A Two Snapshot Algorithm for Concurrency Control in Multi-Level Secure Databases. In *IEEE Symposium on Security and Privacy*, Oakland, 1992.

[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.

[BCCGY94a] N. Boulahia-Cuppens, F. Cuppens, A. Gabillon, and K. Yazdanian. Decomposition of Multilevel Objects in an Object-Oriented Database. In *European symposium on research in computer security*, Brighton, UK, 1994. Springer Verlag.

[BCCGY94b] N. Boulahia-Cuppens, F. Cuppens, A. Gabillon, and K. Yazdanian. Virtual View Model to Design a Secure Object-Oriented Database. In *Proceedings of the 17th National Computer Security Conference*, Baltimore, USA, 1994.

[BJ93] E. Bertino and S. Jajodia. Modeling Multilevel Entities Using Single-level Objects. In *Proceedings of the Third Conference on Deductive and Object-Oriented Databases*, volume 760 of *Lecture Notes in Artificial Intelligence*. Springer Verlag, 1993.

[BMJ94] E. Bertino, L. Mancini, and S. Jajodia. Collecting Garbage in Multilevel Secure Object Stores. In *IEEE Symposium on Security and Privacy*, Oakland, 1994.

[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.

[Gab95] A. Gabillon. *Sécurité Multi-Niveaux dans les Bases de Données à Objets*. Thèse de Doctorat. ENSAE, 1995.

[HOT91] J. T. Haigh, R. C. O'Brien, and D. J. Thomsen. The LDV Secure Relational DBMS Model. In *Database Security, IV: Status and Prospects*. North-Holland, 1991. Results of the IFIP WG 11.3 Workshop on Database Security.

[JK90] S. Jajodia and B. Kogan. Integrating an Object-Oriented Data Model with Multi-Level Security. In *IEEE Symposium on Security and Privacy*, Oakland, 1990.

[KT90a] T. Keefe and W. Tsai. Multiversion Concurrency Control for Multilevel Secure Database Systems. In *IEEE Symposium on Security and Privacy*, Oakland, 1990.

[KT90b] T. Keefe and W. Tsai. Prototyping the SODA Security Model. In *Database Security, 3: Status and Prospects*. North-Holland, 1990. Results of the IFIP WG 11.3 Workshop on Database Security.

[KTT89] T. Keefe, W. Tsai, and M. Thuraisingham. SODA : A Secure Object-Oriented Database System. *Computer and Security*, 8(6), 1989.

[Lun90] T. F. Lunt. Multilevel Security for Object-Oriented Database Systems. In D. L. Spooner and C. Landwehr, editors, *Database Security, III: Status and Prospects*. North-Holland, 1990. Results of the IFIP WG 11.3 Workshop on Database Security.

[Lun91] T. F. Lunt. Polyinstantiation: an inevitable part of a multilevel world. In *Proc. of the computer security foundations workshop*, Franconia, 1991.

[MJ93] J. McDermott and S. Jajodia. Orange Locking: Channel-Free Database Concurrency Control Via Locking. In *Database Security, 6: Status and Prospects*. North-Holland, 1993. Results of the IFIP WG 11.3 Workshop on Database Security.

[ML92] J. K. Millen and T. F. Lunt. Security for Object-Oriented Database Systems. In *IEEE Symposium on Security and Privacy*, Oakland, 1992.

[MQ93] J. Meseguer and X. Qian. A Logical Semantics for Object-Oriented Databases. In *ACM SIGMOD*, Washington D.C, 1993.

[OS94] M. S. Olivier and S. H. Von Solms. A Taxonomy for Secure Object-Oriented Databases. *ACM Tansactions on Database Systems*, 19(1), March 1994.

[SJ92] R. Sandhu and S. Jajodia. Polyinstantiation for cover stories. In *European symposium on research in computer security*, Toulouse, France, 1992. Springer Verlag.

[SMKL95] M. Schaefer, P. Martel, T. Kanawan, and V. Lyons. Multilevel Data Model for the Trusted ONTOS Prototype. In *Ninth Annual IFIP WG 11.3 Working Conference on Database Security*, Rensselaerville, USA, 1995.

[Wie91] R. J. Wieringa. A Formalization of Objects Using Equational Dynamic Logic. In C. Delobel, M. Keifer, and Y. Masunaga, editors, *Second International Conference DOOD'91*, volume 566 of *Lecture Notes in Computer Science*, Munich, Germany, 1991. Springer-Verlag.