Formalisation de la Confidentialité dans une Base de Données à Objets

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

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

Résumé

Plusieurs modèles de sécurité multi-niveaux pour les bases de données à objets sont récemment apparus dans la littérature. Dans cet article, nous nous intéressons au modèle Multi-Vues [BCGY93b][BCGY94]. Notre premier objectif n'est pas de présenter un nouveau modèle de sécurité, mais plutôt de présenter le modèle Multi-Vues de façon formelle. Un second objectif est d'étendre le modèle Multi-Vues en y incluant de nouvelles fonctionnalités, telles que la protection du schéma. Notre approche est la suivante. Nous proposons tout d'abord un langage basé sur la logique du premier ordre afin de pouvoir représenter tout type d'information contenue dans une base de données à objets ainsi qu'un certain nombre de contraintes d'intégrité inhérentes au paradigme objet. Un premier modèle de sécurité appelé le modèle Vue Unique est alors défini. Dans ce modèle de sécurité, toute formule atomique du langage utilisé pour représenter une base de données à objets peut recevoir un niveau de classification. Nous dérivons également un certain nombre de théorèmes régissant l'assignation des niveaux de sécurité. Finalement, nous montrons comment raffiner le modèle Vue Unique afin d'obtenir le modèle Multi-Vues.

1 Introduction

Une fonction importante des sytèmes de gestion de bases de données est la fonction de sécurité. La sécurité dans les bases de données a pour objectif de maintenir à la fois la confidentialité et l'intégrité des informations. Dans cet article, nous nous intéressons à une politique de sécurité multi-niveaux assurant la confidentialité des informations. Dans le cadre d'un règlement multi-niveaux, tout utilisateur reçoit un niveau de sécurité appelé habilitation alors que toute information reçoit un niveau de sécurité appelé classification. L'ensemble des niveaux de sécurité forme un treillis. Dans ce contexte, la propriété de confidentialité s'exprime par le fait qu'un utilisateur a la permission de connaître les informations dont la classification est inférieure ou égale à son niveau d'habilitation, mais a l'interdiction de connaître les informations dont la classification est supérieure à son niveau d'habilitation. Notre objectif, dans cet article, n'est pas de définir un nouveau modèle de sécurité multi-niveaux pour les bases de données à objets, mais de présenter une version *formelle* et particulièrement *complète* du modèle Multi-Vues [BCGY93b][BCGY94]. Brièvement, notre modèle de sécurité offre les fonctionnalités suivantes :

- Gestion des objets multi-niveaux. Chaque valeur d'attribut de chaque objet reçoit un niveau de classification.
- Gestion des attributs polyinstantiés. Un attribut d'objet peut avoir plusieurs valeurs classifiée chacune séparément.
- Protection de l'existence des objets. Chaque objet reçoit un niveau de classification destiné à protéger son existence.
- Protection du schéma. Chaque classe, chaque attribut, chaque méthode mais aussi chaque lien d'héritage reçoit un niveau de classification destiné à protéger son existence.

L'outil de formalisation que nous utilisons est la logique du premier ordre. Cette approche formelle nous permet en particulier d'étudier les conséquences de la classification d'une information sur les niveaux de sécurité des autres informations. La suite de l'article est organisée de la façon suivante :

Dans la Section 2, nous définissons tout d'abord un langage du premier ordre. Ce langage permet de formaliser aussi bien le contenu d'une base de données à objets que des contraintes d'intégrité devant être respectées dans une base de données à objets. Dans la Section 3, nous étendons ce langage afin de pouvoir classifier toute formule atomique représentant une information contenue dans la base de données. A partir des contraintes d'intégrité, des théorèmes de contrôle des flux d'information sont dérivés. Ces théorèmes doivent être respectés dans toute base de données à objets multi-niveaux. Un premier modèle de sécurité est ainsi obtenu. Ce modèle est appelé le modèle Vue Unique. Dans le modèle Vue Unique, les attributs d'un objet ou d'une classe peuvent être classifiés de façon indépendante. Le modèle Vue Unique permet donc de gèrer des entités multi-niveaux. Cependant, de ce fait, le modèle Vue Unique semble difficilement implantable. Un raffinement du modèle Vue Unique est alors proposé dans la Section 4. Ce nouveau modèle de sécurité est appelé le modèle Multi-Vues. Dans le modèle Multi-Vues, chaque entité multi-niveaux est décomposée en plusieurs entités mono-niveau. Enfin, en Section 5, nous présentons les principes d'implantation du modèle Multi-Vues.

2 Modèle non protégé

L'objectif de cette section est de proposer une formalisation du contenu d'une base de données à objets non protégée à l'aide de la logique du premier ordre. La logique du premier ordre a souvent été utilisée pour formaliser les bases de données (essentiellement les bases de données relationnelles) selon deux approches possibles :

- Approche théorie de la preuve. Dans cette approche, la base de donnée est assimilée à une théorie logique.
- Approche théorie des modèles. Dans cette approche, la base de données est assimilée à une interprétation particulière d'une théorie logique.

Dans ce chapitre nous adoptons l'approche théorie de la preuve pour représenter le contenu d'une base de données à objets.

Il n'est pas évident a priori de pouvoir formellement définir le modèle objet à l'aide d'uniquement la logique du premier ordre. Il semble, en effet, que certains concepts, tels que les méthodes (cf [Wie91] par exemple) requièrent des logiques d'ordre supérieur pour pouvoir être complètement formalisés. Notre but n'est pas de proposer une formalisation complète du modèle objet, mais une formalisation *suffisante* pouvant servir de support à la définition d'un modèle complet de sécurité multi-niveaux pour les bases de données à objets. Pour une description complète et exhaustive du modèle objet (adapté aux bases de données à objets), nous renvoyons le lecteur à [ODMG93] ou [BDK92] par exemple.

2.1 Langage

Le langage *L* que nous utilisons est basé sur la logique du premier ordre avec égalité. Pour pouvoir représenter une base de données à objets, nous considérons huit prédicats particuliers :

- Deux prédicats unaires *Object* et *Class*.
- Cinq prédicats binaires *CA*, *OA*, *Method*, *Instance*, *Isa*.
- Un prédicat ternaire *Val*.

Intuitivement, ces prédicats sont interprétés de la façon suivante :

- *Object(o)* doit être lu «o identifie un objet existant».
- *Class(c)* doit être lu «c identifie une classe existante».
- *CA(c,a)* doit être lu «a est un attribut de la classe c[1]».

1. Rigoureusement, nous devrions dire a est un attribut de la classe c. Dans la suite, nous confondrons la classe ou l'objet avec son identificateur.

- *OA(o,a)* doit être lu «a est un attribut de l'objet o».
- *Method(m,c)* doit être lu «m est une méthode de la classe c».
- *Instance(o,c)* doit être lu «o est un objet instance de la classe c».
- *Isa(c,c')* doit être lu «c est une sous-classe de la classe c'».
- *Val(a,o,v)* doit être lu «v est la valeur de l'attribut a de l'objet o».

2.2 Axiomatique

L'axiomatique de notre théorie est constituée des schémas d'axiomes classiques de la logique du premier ordre avec égalité, auquel nous ajoutons les axiomes propres de notre théorie. Ces axiomes propres sont constitués par un ensemble de *contraintes d'intégrité* devant être respectées dans n'importe quelle base de données à objets. Notons que toutes les contraintes d'intégrité que nous proposons sont compatibles avec les spécifications de [ODMG93].

- Si *a* est un attribut de la classe *c*, alors *c* est une classe existante.

$$\forall a \forall c, CA(c, a) \rightarrow Class(c) \quad (1)$$

- Si *a* est un attribut de l'objet *o*, alors *o* est un objet existant.

$$\forall a \forall o, OA(o, a) \rightarrow Object(c) \quad (2)$$

- Si *m* est une méthode de la classe *c*, alors *c* est une classe existante.

$$\forall m \forall c, Method(c, m) \rightarrow Class(c) \quad (3)$$

- Tout attribut d'objet a une valeur.

$$\forall a \forall o, OA(o, a) \leftrightarrow \exists v, Val(a, o, v) \quad (4)$$

- La valeur d'un attribut est unique.

$$\forall a \forall o \forall v \forall v', Val(a, o, v) \wedge Val(a, o, v') \rightarrow v = v' \quad (5)$$

- Si *o* est une instance de *c*, alors *o* est un objet et *c* est une classe.

$$\forall o \forall c, Instance(o, c) \rightarrow Object(o) \wedge Class(c) \quad (6)$$

- Les deux arguments du prédicat *Isa* doivent identifier chacun une classe.

$$\forall c \forall c', Isa(c, c') \rightarrow Class(c) \wedge Class(c') \quad (7)$$

- *Isa* est antisymétrique, antiréflexif et transitif.

$$\forall c \forall c', Isa(c, c') \rightarrow \neg Isa(c', c) \quad (8)$$

$$\forall c, \neg Isa(c, c) \quad (9)$$

$$\forall c \forall c' \forall c'', Isa(c, c') \wedge Isa(c', c'') \rightarrow Isa(c, c'') \quad (10)$$

- Propriété d'héritage 1 : Tout attribut *a* appartenant à une classe *c'* est hérité dans toute sous-classe *c* de *c'*.

$$\forall c \forall c' \forall a, Isa(c, c') \wedge CA(c', a) \rightarrow CA(c, a) \quad (11)$$

- Propriété d'héritage 2 : Toute méthode *m* appartenant à une classe *c'* est héritée dans toute sous-classe *c* de *c'*.

$$\forall c \forall c' \forall m, Isa(c, c') \wedge Method(c', m) \rightarrow Method(c, m) \quad (12)$$

- Propriété d'héritage 3 : Tout attribut *a* d'un objet *o* est hérité d'une classe dont l'objet *o* est instance. Réciproquement tout attribut *a* d'une classe *c* est hérité par tous les objets qui sont instance de la classe *c*.

$$\forall o \forall a, OA(o, a) \leftrightarrow \exists c, Instance(o, c) \wedge CA(c, a) . \quad (13)$$

- Tout objet instance d'une classe *c* est aussi instance de toutes les super-classes de *c*.

$$\forall o \forall c \forall c', Instance(o, c) \wedge Isa(c, c') \rightarrow Instance(o, c') \quad (14)$$

- Tout objet est instance d'au moins une classe.

$$\forall o, Object(o) \rightarrow \exists c, Instance(o, c) \quad (15)$$

Nous avons pleinement conscience que ces axiomes ne suffisent pas à établir formellement tous les concepts relatifs à une base de données à objets. Nous formalisons par exemple les méthodes à l'aide d'une définition syntaxique très simple. Cette définition sera toutefois suffisante pour pouvoir inclure dans notre modèle de sécurité, la possibilité de protéger l'existence d'une méthode. A contrario, la protection du code d'une méthode ne pourra être explicitement protégé dans la mesure où ce code n'est pas représenté dans la théorie ci-dessus.

Pour des raisons de simplification, nous ne formalisons pas également les concepts de valeur d'attribut ensemble ou tuple, permettant de représenter des attributs structurés. La formalisation de ces concepts compliqueraient la présentation de notre modèle de sécurité. Une formalisation de tels attributs est toutefois presentée dans [GAB95].

2.3 Exemple de base de données à objets non protégée

La Figure 1 présente un exemple de base de données à objets. Cet exemple inclut trois classes *Employé*, *Employé_à_licencier* et *Employé_entreprise_A*. Nous supposons que la classe *Employé_entreprise_A* est sous-classe de la classe *Employé_à_licencier*. Cela signifie que tout employé de l'entreprise A doit être licencié. Cet exemple inclut également deux objets *O1* et *O2*. *O1* est une instance de la classe *Employé_entreprise_A*, alors que *O2* est instance de la classe *Employé_à_licencier*. Ces deux objets ont trois attributs identiques *Nom*, *Salaire* et *Religion* hérités de la classe *Employé* ainsi que le même attribut *Date_licenciement* hérité de la classe *Employé_à_licencier*.

D'un point de vue théorique, cette base de données est représentée par la théorie que nous avons définie à la section précédente à laquelle s'ajoutent un certain nombre d'axiomes permettant de dériver toutes les informations représentées graphiquement sur la Figure 1.

3 Le modèle Vue Unique

Notre objectif dans cette section est de présenter le modèle Vue Unique. Le modèle Vue Unique est un modèle complet et défini de façon formelle. Il présente les caractéristiques suivantes :

- Gestion des objets multi-niveaux. Chaque valeur d'attribut d'un objet *o* reçoit un niveau de classification. Ce niveau protège le fait «la valeur de l'attribut *a* de l'objet *o* est *v*».
- Protection de l'existence des attributs d'objet. Chaque attribut d'objet reçoit un niveau de classification destiné à protéger son existence.
- Gestion des attributs polyinstantiés. Un attribut d'objet peut avoir plusieurs valeurs classifiées chacune séparément.
- Protection de l'existence des objets. Chaque objet reçoit un niveau de classification destiné à protéger son existence.
- Protection des attributs de classe. Chaque attribut de classe reçoit un niveau de classification destiné à protéger son existence.
- Protection des méthodes. Chaque méthode reçoit un niveau de classification destiné à protéger son existence.
- Protection de la hiérarchie des classes. Chaque lien de type *Isa* reçoit un niveau de classification destiné à protéger le fait que telle classe est sous-classe de telle autre classe.
- Protection du fait qu'un objet est instance d'une certaine classe. Chaque lien de type *Instance* reçoit un niveau de classification destiné à protéger le fait que tel objet est instance de telle classe.

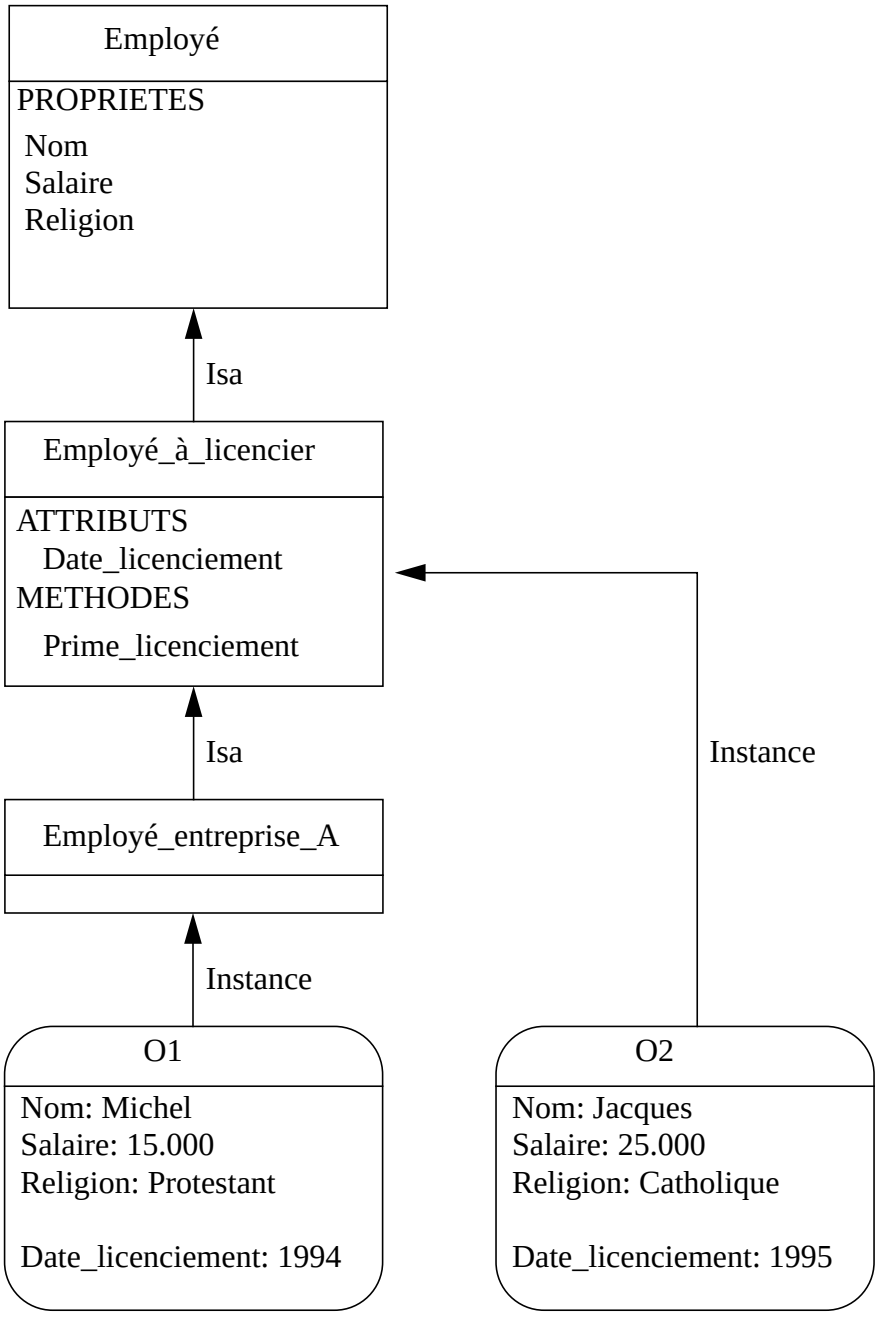

FIGURE 1 Exemple d'une base de données à objets

3.1 Niveaux de sécurité

Nous étendons tout d'abord le langage *L* défini à la section précédente avec le prédicat unaire *Level*. Intuitivement, la formule *Level(x)* doit être lu «*x* est un niveau de sécurité». Nous supposons que l'ensemble des niveaux de sécurité est muni d'une relation d'ordre partiel «domine» que nous représentons par le prédicat ≥. De ce fait, les notions de borne inférieure et de borne supérieure peuvent être introduites dans notre langage. Nous utilisons pour cela deux fonctions *lub* (least upper bound) et *glb* (greatest lower bound). Si l_1, l_2, ..., l_n sont des niveaux de sécurité, alors *lub*(l_1, l_2, ..., l_n) et *glb*(l_1, l_2, ..., l_n) sont respectivement la borne supérieure et la borne inférieure des niveaux de sécurité l_1, l_2, ..., l_n. Nous supposons également qu'il existe un niveau de sécurité noté *LOW* qui est

strictement inférieur à tous les autres et un niveau de sécurité noté *HIGH* qui est strictement supérieur à tous les autres. Les axiomes associés aux prédicats *Level* et ≥ correspondant aux hypothèses précédentes sont triviaux. Nous ne les fournissons pas car nous ne les utilisons pas par la suite.

3.2 Langage

Le langage *L'* que nous utilisons pour formaliser une base de données à objets multi-niveaux est une extension du langage *L*. A chaque prédicat *P* d'arité *n* utilisé pour représenter une base de données à objets non protégé est associé un prédicat *P'* d'arité *n+1* utilisé pour représenter une base de données à objets multi-niveaux. Nous obtenons ainsi huit prédicats multi-niveaux *Object'*, *Class'*, *OA'*, *CA'*, *Method'*, *Val'*, *Isa'*, *Instance'*. Intuitivement, si *l* est un niveau de sécurité, alors $P'(t_1,...,t_n, l)$ doit être lu «l'information $P(t_1,...,t_n)$ est classifiée au niveau *l*».

Notons que nous pourrions appliquer ce processus de classification aux prédicats *Level* et ≥ présentés en Section 3.1. Ainsi, nous pourrions considérer un prédicat binaire *Level'*. *Level'(l,l')* devrait être lu «le fait que *l* soit un niveau de sécurité est classifié au niveau *l'*». Nous pourrions également considérer un prédicat ternaire ≥'. *≥'(l,l',l")* devrait être lu «le fait que le niveau *l* domine le niveau *l'* est classifié au niveau *l"*». Ces extensions seraient parfaitement acceptables et permettraient de protéger des informations relatives à l'ensemble des niveaux de sécurité. Toutefois, dans un but de simplification, nous n'ajouterons pas ces deux prédicats *Level'* et ≥' à notre langage. Ceci signifie implicitement que nous supposons que tout fait appartenant à l'extension des prédicats *Level* et ≥ est implicitement classifié au niveau *LOW*.

3.3 Axiomatique

Nous ajoutons à la liste des axiomes (1)-(15) les axiomes spécifiant les liens entre une base de données à objets non protégée et une base de données à objets multi-niveaux:

Pour chaque prédicat *P*, nous avons l'axiome suivant:

- Cet axiome stipule que l'extension du prédicat *P'* dans la base de données à objets multi-niveaux est obtenue en classifiant tous les faits appartenant à l'extension du prédicat *P* dans la base de données à objets non protégée.

$$\forall t_1 \ldots \forall t_n, P(t_1, \ldots, t_n) \rightarrow \exists l, P'(t_1, \ldots, t_n, l) \quad (16)$$

Réciproquement, pour chaque prédicat *P'* différent de *Val'*, nous avons l'axiome suivant:

- Cet axiome stipule que l'extension du prédicat *P* dans la base de données non protégée est obtenue en supprimant les niveaux de sécurité associés aux faits appartenant à l'extension du prédicat *P'* dans la base de données à objets multi-niveaux.

$$\forall t_1 \ldots \forall t_n \forall l, P'(t_1, \ldots, t_n, l) \rightarrow Level(l) \wedge P(t_1, \ldots, t_n) \quad (17)$$

Pour le prédicat *Val'*, la réciproque est différente car nous acceptons que ce prédicat puisse être polyinstantié.

- Cet axiome stipule que si l'information *Val(a,o,v)* est classifiée au niveau *l* dans la base de données multi-niveaux, alors il existe une valeur *v'* telle que l'information *Val(a,o,v')* appartient à l'extension du prédicat *Val* dans la base de données non protégée. Comme la polyinstantiation est possible, nous pouvons avoir $v \neq v'$.

$$\forall a \forall o \forall v \forall l, Val'(a, o, v, l) \rightarrow \exists v', Level(l) \wedge Val(a, o, v') \quad (18)$$

- Cet axiome indique que pour un niveau de sécurité donné, la valeur d'un attribut d'objet doit être unique. Cet axiome stipule donc que, pour un niveau de sécurité donné, la polyinstantiation d'un attribut d'objet est impossible.

$$\forall a \forall o \forall v \forall v' \forall l, Val'(a, o, v, l) \wedge Val'(a, o, v', l) \rightarrow v = v' \quad (19)$$

Notons que l'axiome (19) est la contrepartie de l'axiome (5) de la base de données à objets non protégée. Notons également que nous pouvons utiliser l'axiome (16) pour dériver toutes les contreparties multi-niveaux des autres axiomes (1)-(15). Par exemple, le théorème contrepartie multi-niveaux de l'axiome (1) est le suivant:

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

Notre objectif dans la Section 3.4 suivante est de dériver des contraintes devant être respectées lors de l'assignation des différents niveaux de sécurité aux informations de la base de données à objets non protégée.

3.4 Théorèmes de contrôle des flux

Afin de pouvoir contrôler les flux d'information existants dans une base de données à objets multi-niveaux, nous ajoutons les deux règles d'inférence ci-dessous à notre théorie. Ces deux règles d'inférence permettent de dériver des théorèmes devant régir l'assignation des niveaux de sécurité, afin qu'il n'y ait pas dans la base de données à objets multi-niveaux de flux d'information illégal.

Règle 1 Soient $x_1, ..., x_n$ et y des tuples de variables respectivement compatibles avec l'arité des prédicats $P_1, ..., P_n$ et Q. Nous supposons que chaque variable de y apparait au moins dans un des tuples $x_1, ..., x_n$:

SI

$$\forall x_1 \ldots \forall x_n, P_1(x_1) \wedge \ldots \wedge P_n(x_n) \rightarrow Q(y)$$

est *axiome* de la base de données à objets non protégée

ALORS

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

est un *théorème* de la base de données à objets multi-niveaux

Si la règle 1 n'est pas satisfaite, alors un sujet habilité au niveau $lub(l_1, ..., l_n)$ pourra accéder à tous les $P_i(x_i)$ et utiliser l'axiome en prémisse de la règle 1 afin de dériver $Q(y)$. Si le niveau de classification protégeant $Q(y)$ n'est pas dominé par $lub(l_1, ..., l_n)$, alors un canal caché, permettant l'inférence d'une information ne devant pas être dévoilée, est créé.

Règle 2 Soient $x_1, ..., x_n$ et $y_1, ..., y_p$ des tuples de variables respectivement compatibles avec l'arité des prédicats $P_1, ..., P_n$ et $Q_1, ..., Q_p$ et soit y un autre tuple de variables. Nous supposons que chaque variable de y apparait dans au moins un des tuples $y_1, ..., y_p$. Nous supposons aussi que chaque variable des *tuples* $y_1, ..., y_p$ apparait dans au moins un des tuples $x_1, ..., x_n, y$:

SI

$$\forall x_1 \ldots \forall x_n, P_1(x_1) \wedge \ldots \wedge P_n(x_n) \rightarrow \exists y, Q_1(y_1) \wedge \ldots \wedge Q_p(y_p)$$

est *axiome* de la base de données à objets non protégée

ALORS

$$\forall x_1 \ldots \forall x_n \forall l_1 \ldots \forall l_n, P'_1(x_1, l_1) \wedge \ldots \wedge P'_n(x_n, l_n)$$

$$\rightarrow \exists y \exists l'_1 \ldots \exists l'_p, Q'_1(y_1, l'_1) \wedge \ldots \wedge Q'_p(y_p, l'_p) \wedge lub(l_1, \ldots, l_n) \geq lub(l'_1, \ldots, l'_p)$$

est un *théorème* de la base de données à objets multi-niveaux

Si la règle 2 n'est pas satisfaite, alors un sujet habilité au niveau $lub(l_1, ..., l_n)$ pourra accéder à tous les $P_i(x_i)$ et utiliser l'axiome prémisse de la règle 2 afin de dériver l'existence des faits $Q_1(y_1), ..., Q_p(y_p)$ sachant que certains d'entre eux ont un niveau de sécurité dominant strictement $lub(l_1, ..., l_n)$. De ce fait, un canal caché, permettant de signaler l'existence d'informations ne devant pas être dévoilée, est créé.

Les théorèmes[1] de contrôle des flux dérivés à partir de ces deux règles sont répertoriés dans les deux sous sections suivantes.

3.4.1 Théorèmes dérivés à l'aide de la règle 1

- Le niveau de protection du fait «*a* est un attribut de la classe *c*» doit dominer le niveau de protection du fait «*c* est une classe»:

$$\forall a \forall c \forall l \forall l', CA'(c, a, l) \wedge Class'(c, l') \rightarrow l \geq l' \quad (20)$$

- Le niveau de protection du fait «*a* est un attribut de l'objet *o*» doit dominer le niveau de protection du fait «*o* est un objet»:

$$\forall a \forall o \forall l \forall l', OA'(o, a, l) \wedge Object'(o, l') \rightarrow l \geq l' \quad (21)$$

- Le niveau de protection du fait «*m* est une méthode de la classe *c*» doit dominer le niveau de protection du fait «*c* est une classe»:

$$\forall m \forall c \forall l \forall l', Method'(c, m, l) \wedge Class'(c, l') \rightarrow l \geq l' \quad (22)$$

- Le niveau de protection du fait «*v e*st une valeur de l'attribut *a* de l'objet *o*» doit dominer le niveau de protection du fait «*a* est un attribut de l'objet *o*»:

$$\forall a \forall o \forall v \forall l \forall l', Val'(o, a, v, l) \wedge OA'(o, a, l') \rightarrow l \geq l' \quad (23)$$

- Le niveau de protection du fait «l'objet *o* est instance de la classe *c*» doit dominer la borne supérieure des niveaux de protection des faits «*o* est un objet» et «*c* est une classe» :

$$\forall c \forall o \forall l \forall l' \forall l'', Instance'(o, c, l) \wedge Object'(o, l') \wedge Class'(c, l'') \rightarrow l \geq lub(l', l'') \quad (24)$$

- Le niveau de protection du fait «la classe *c* est sous-classe de la classe *c'*» doit dominer la borne supérieure des niveaux de protection des faits «*c* est une classe» et «*c'* est une classe» :

$$\forall c \forall c' \forall l \forall l' \forall l'', Isa'(c, c', l) \wedge Class'(c, l') \wedge Class'(c', l'') \rightarrow l \geq lub(l', l'') \quad (25)$$

- Le niveau de protection du fait «la classe *c* est sous-classe de la classe *c''*» doit être dominé par la borne supérieure des niveaux de protection des faits «*c* est sous-classe de *c'*» et «*c'* est sous-classe de *c''*» :

$$\forall c \forall c' \forall c'' \forall l \forall l' \forall l'', Isa'(c, c', l) \wedge Isa'(c', c'', l') \wedge Isa'(c, c'', l'') \rightarrow lub(l, l') \geq l'' \quad (26)$$

- Le niveau de protection du fait «l'objet *o* est instance de la classe *c'*» doit être dominé par la borne supérieure des niveaux de protection des faits «*c* est sous-classe de *c'*» et «*o* est instance de *c*» :

$$\forall o \forall c \forall c' \forall l \forall l' \forall l'', Instance'(o, c, l) \wedge Isa'(c, c', l') \wedge Instance'(o, c', l) \rightarrow lub(l, l') \geq l'' \quad (27)$$

- Le niveau de protection du fait «*a* est un attribut de la classe *c*» doit être dominé par la borne supérieure des niveaux de protection des faits «*c* est sous-classe de *c'*» et «*a* est un attribut de *c'*» :

$$\forall c \forall c' \forall a \forall l \forall l' \forall l'', Isa'(c, c', l) \wedge CA'(c', a, l') \wedge CA'(c, a, l'') \rightarrow lub(l, l') \geq l'' \quad (28)$$

- Le niveau de protection du fait «*m* est une méthode de la classe *c*» doit être dominé par la borne supérieure des niveaux de protection des faits «*c* est sous-classe de *c'*» et «*m* est une méthode de *c'*» :

$$\forall c \forall c' \forall m \forall l \forall l' \forall l'', Isa'(c, c', l) \wedge Method'(c', m, l') \wedge Method'(c, m, l'') \rightarrow lub(l, l') \geq l'' \quad (29)$$

- Le niveau de protection du fait «*a* est un attribut de l'objet *o*» doit être dominé par la borne supérieure des niveaux de protection des faits «*o* est instance de *c*» et «*a* est un attribut de *c*» :

1. Voir [GAB95] pour les démonstrations détaillées de tous ces théorèmes

$$\forall c \forall o \forall a \forall l \forall l' \forall l'', Instance'(o, c, l) \wedge CA'(c, a, l') \wedge OA'(o, a, l'') \rightarrow lub(l, l') \geq l'' \quad (30)$$

3.4.2 Théorèmes dérivés à l'aide de la règle 2

- Ce théorème stipule que si «*a* est un attribut de l'objet *o*» est classifié au niveau *l*, alors il doit exister une valeur de cet attribut dont le niveau *l'* de classification est dominé par le niveau *l*.

$$\forall a \forall o \forall l, OA'(o, a, l) \rightarrow \exists v \exists l', Val'(o, a, v, l') \wedge l \geq l' \quad (31)$$

- En combinant ce théorème (31) avec le théorème (23), nous obtenons alors le théorème intéressant ci-dessous qui stipule que si «*a* est un attribut de l'objet *o*» est classifié au niveau *l*, alors il doit exister une valeur de cet attribut au niveau *l*.

$$\forall a \forall o \forall l, OA'(o, a, l) \rightarrow \exists v, Val'(o, a, v, l) \quad (32)$$

- Ce théorème stipule que si «*o* est un objet» est classifié au niveau *l*, alors il doit exister une classe *c* telle que «*o* est instance de *c*» est classifié à un niveau *l'* dominé par le niveau *l*.

$$\forall o \forall l, Object'(o, l) \rightarrow \exists c \exists l', Instance'(o, c, l') \wedge l \geq l' \quad (33)$$

- En combinant ce théorème (33) avec le théorème (24), nous obtenons alors le théorème intéressant ci-dessous qui stipule que si le fait «*o* est un objet» est classifié au niveau *l*, alors il doit exister une classe *c* telle que le fait «*o* est instance de *c*» est classifié au niveau *l*.

$$\forall o \forall l, Object'(o, l) \rightarrow \exists c, Instance'(o, c, l) \quad (34)$$

- Ce théorème stipule que si «*a* est un attribut de l'objet *o*» est classifié au niveau *l*, alors il doit exister une classe *c* telle que le fait «*a* est un attribut de *c*» est classifié au niveau *l'* et le fait «*o* est instance de *c*» est classifié au niveau *l''*, avec *l* qui domine la borne supérieure des niveaux *l'* et *l''*.

$$\forall o \forall a \forall l, OA'(o, a, l) \rightarrow \exists c \exists l' \exists l'', Instance'(o, c, l') \wedge CA'(c, a, l'') \wedge l \geq lub(l', l'') \quad (35)$$

- En combinant ce théorème (35) avec le théorème (30), nous obtenons alors le théorème intéressant ci-dessous qui stipule que si le fait «*a* est un attribut de l'objet *o*» est classifié au niveau *l*, alors il doit exister une classe *c* telle que les faits «*a* est un attribut de *c*» et «*o* est instance de *c*» sont respectivement classifiés aux niveaux *l'* et *l''* avec *l = lub(l',l'')*.

$$\forall o \forall a \forall l, OA'(o, a, l) \rightarrow \exists c \exists l' \exists l'', Instance'(o, c, l') \wedge CA'(c, a, l'') \wedge l = lub(l', l'') \quad (36)$$

3.5 Polyinstantiation

Nous posons que tout prédicat *P'* d'arité *n+1* différent de *Val'* est polyinstantié s'il existe $t_1,...,t_n$, *l* et *l'* tel que

$$P'(t_1, \ldots, t_n, l) \wedge P'(t_1, \ldots, t_n, l') \wedge l \neq l'$$

Pour le prédicat *Val'*, nous avons implicitement (axiome (18)) vu que la définition était différente. Le prédicat *Val'* est polyinstantié s'il existe *o, a, v, v', l* et *l'* tel que :

$$Val'(o, a, v, l) \wedge Val'(o, a, v', l') \wedge l \neq l'$$

L'objectif de cette section est de tenter de donner une interprétation à la polyinstantiation de chaque prédicat *P'*. L'objectif de la Section 3.6 sera de proposer des techniques permettant d'éviter la polyinstantiation des prédicats *P'*.

3.5.1 Polyinstantiation de la valeur d'un attribut

Considérons le théorème (32). Ce théorème stipule qu'un sujet autorisé à connaître l'existence d'un attribut d'objet doit pouvoir accéder à une valeur associée à cet attribut. La polyinstantiation est une technique généralement

utilisée pour satisfaire ce type d'exigence. Considérons, par exemple, notre exemple de base de données à objets présenté à la Section 2.3. Supposons que :

$OA'(O_1, Salaire, LOW) \wedge Val'(O_1, Salaire, 15000, HIGH)$

Cette expression signifie que le fait «*Salaire* est un attribut de O_1» est classifié au niveau *LOW* et le fait «la valeur de l'attribut *Salaire* de l'objet O_1 est *15000*» est classifié au niveau *HIGH*. Le théorème (32) stipule qu'il doit exister une valeur associée à cet attribut et classifiée au niveau *LOW*, par exemple *Val'(O_1, Salaire, 10000, LOW)*. La valeur *10000* constitue un leurre destiné à dissimuler aux sujets habilités *LOW*, l'existence de la valeur protégée (*15000*).

3.5.2 Polyinstantiation d'un objet

Dans un article consacré à la polyinstantiation dans les BD relationnelles, Lunt [Lun91] décrit un autre type de polyinstantiation, appelé polyinstantiation d'une entité. Dans le contexte d'une BD relationnelle, il y a polyinstantiation d'une entité lorsque une relation contient plusieurs tuples ayant une même clé primaire, mais ayant chacun un niveau de sécurité associé à la clé primaire, différent. Dans le contexte de notre base de données à objets multi-niveaux, un problème similaire se produit lorsqu'un objet est associé à au moins deux niveaux de sécurité:

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

Une interprétation possible suggérée par [Lun91], serait de considérer, qu'en réalité les deux faits *Object'(o,l)* et *Object'(o,l')* font référence à deux entités distinctes du monde réel. Cette interprétation serait contraire à un principe du paradigme objet qui stipule qu'un identificateur d'objet doit faire référence à un objet unique. C'est pourquoi, nous pensons qu'il est préférable de prévenir la polyinstantiation d'un objet dans le cadre d'une base de données à objets.

3.5.3 Autres types de polyinstantiation

Le prédicat *Class'* peut également être polyinstantié. Nous pouvons avoir :

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

Nous pensons que ce type de polyinstantiation doit aussi être évité, pour la même raison que celle invoquée à la Section 3.5.2 précédente.

Les prédicats *CA'* et *Method'* peuvent être polyinstantiés. Nous pouvons avoir:

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

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

Dans les deux cas, l'interprétation suivante peut être faite. Supposons $l' \geq l$. Dans ce cas nous pouvons considérer qu'il existe une première définition de l'attribut *a* ou de la méthode *m* au niveau *l* et une seconde définition de cet attribut ou de cette méthode au niveau *l'* qui *surcharge* la définition au niveau *l*. Toutefois notre modèle ne permet pas de spécifier complètement un attribut ou une méthode. Les concepts de typage d'un attribut et de programme associé à une méthode ne sont pas pris en compte. Il est donc impossible de définir dans le cadre de notre modèle, ce concept de surcharge. Nous pensons donc que ce type de polyinstantiation doit être évité dans notre modèle.

La polyinstantiation des autres prédicats *OA'*, *Isa'*, *Instance'*, serait difficile à interpréter. C'est pourquoi, la polyinstantiation de ces prédicats doit être évitée dans notre modèle.

3.6 Prévenir la polyinstantiation

3.6.1 Prévenir la polyinstantiation de Val'

Pour prévenir la polyinstantiation du prédicat Val', nous introduisons dans notre langage un symbole spécial, *Restricted*. L'utilisation de ce symbole pour prévenir la polyinstantiation d'une valeur d'attribut est suggérée par [SJ92]. Intuitivement, *Restricted* signifie la valeur existe mais est hautement classifiée. Ceci est exprimé dans l'axiome suivant :

- Cet axiome stipule que s'il existe au niveau *l*, une valeur d'attribut égale à *Restricted*, alors il existe une valeur différente de *Restricted*, à un niveau *l'* dominant strictement *l*.

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

De ce fait, si l'existence d'un attribut d'objet est classifié à un niveau *l* et que nous voulons associer à cet attribut une valeur à un niveau *l'* dominant strictement *l*, alors nous avons deux possibilités pour assurer le théorème (32) :

- Associer un leurre au niveau *l* à l'attribut, si nous voulons protéger l'existence de la valeur classifiée au niveau *l'*. Ce choix correspond à l'option «mentir» [EY92].
- Associer *Restricted* au niveau *l* à l'attribut dans le cas contraire. Ce choix correspond à l'option «dire la vérité» [EY92].

3.6.2 Prévenir la polyinstantiation des autres prédicats P'

Si *P'* est un prédicat différent de *Val'*, *P'* ne peut être polyinstantié si nous incluons l'axiome suivant dans notre modèle:

- Cet axiome stipule que chaque fait $P(x_1,...x_n)$ est classifié de façon unique.

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

D'un point de vue pratique, [SJ92] suggère trois techniques possibles pour que cet axiome soit vérifié.

1. Classifier au niveau *LOW* tous les faits appartenant à l'extension d'un prédicat. Par exemple, l'axiome suivant élimine la polyinstantiation des objets.

- Cet axiome stipule que l'existence de chaque objet est classifié au niveau LOW. Tous les objets sont donc visibles au niveau *LOW*:

$$\forall o \forall l, Object'(o, l) \rightarrow l = LOW$$

L'axiome (38) est ainsi vérifié pour le prédicat *Object'*.

2. Partitionner l'ensemble des symboles utilisés pour représenter le contenu de la base de données. Afin de satisfaire cet objectif, nous ajoutons à notre langage un prédicat binaire *Sensitivity.* Intuitivement *Sensitivity(x,l)* doit être lu «la sensitivité associée au symbole *x* est *l*». Les axiomes associés au prédicat *Sensitivity* sont les suivants:

- Le second argument de *Sensitivity* est un niveau :

$$\forall i \forall l, Sensitivity(i, l) \rightarrow Level(l) \quad (39)$$

- La sensitivité d'un symbole est unique :

$$\forall i \forall l \forall l', Sensitivity(i, l) \wedge Sensitivity(i, l') \rightarrow l = l' \quad (40)$$

- Cet axiome permet d'éliminer la polyinstantiation des prédicats *P'*, avec *P'* prédicat d'arité *n+1* et différent de *Val'*.

$$\forall x_1 \ldots \forall x_n \forall l, P'(x_1, \ldots, x_n, l) \rightarrow \exists l_1 \ldots l_n, Sensitivity(x_1, l_1) \wedge \ldots \wedge Sensitivity(x_n, l_n) \wedge l = lub(l_1, \ldots, l) \quad (41)$$

Des axiomes (39), (40) et (41), il est alors facile de dériver (38), comme théorème (cf [GAB95]).

- Cet axiome stipule que la sensitivité du fait «la sensitivité du symbole *i* est *l*» est toujours classifiée au niveau *LOW*

$$\forall i \forall l, Sensitivity(i, l) \rightarrow Sensitivity'(i, l, LOW) \quad (42)$$

La conséquence de cet axiome est que tout utilisateur peut donc connaître la sensitivité associée à n'importe quel symbole. Dès lors le SGBD peut rejeter, sans risque de créer un canal caché, toute insertion à un niveau donné *l*, d'une information contenant un symbole de plus haute sensitivité que *l*.

Pour rendre applicable cette partition de l'ensemble des symboles, nous pouvons, par exemple, préfixer par «S-» tous les symboles dont la sensitivité est secrète.

3. Limiter les insertions. Un troisième moyen pour éliminer la polyinstantiation est d'imposer que toute insertion ne puisse être faite que par les utilisateurs habilités au niveau *HIGH*, c'est à dire les utilisateurs pouvant voir tout le contenu de la BD. Par exemple, nous pouvons imposer que le schéma de la BD ne peut être modifié que par des utilisateurs habilités *HIGH*. Ceci élimine la polyinstantiation des prédicats *Class'*, *Method'* et *CA'*.

Comme le fait remarquer [SJ92], le choix de la solution un, deux ou trois dépend essentiellement des caractéristiques du SGBD et de l'application.

3.7 Exemple de base de données à objets multi-niveaux

Cette section présente la base de données à objets multi-niveaux (Figure 2) dérivée de la base de données à objets non protégée présentée à la Section 2.3. Cette base de données à objets est cohérente avec les axiomes et théorèmes (1) à (38).

Les hypothèses faites dans cet exemple sont les suivantes :

- Nous avons trois niveaux de sécurité $P = LOW$ (Public), C (Confidentiel) et $S = HIGH$ (Secret) avec $S \geq C \geq P$.
- L'existence de *Employé_à_licencier* est confidentielle.
- Le fait que *Employé_entreprise_A* est une sous-classe de *Employé_à_licencier* est secret.
- L'existence de l'attribut *Religion* est secrète dans toutes les classes.
- L'existence de l'objet O_2 est confidentielle.
- Le fait que O_1 est une instance de *Employé_à_licencier* est secret.
- La valeur du *Salaire* de l'objet O_1 au niveau public est *Restricted*. Les utilisateurs habilités public sont donc autorisés à savoir que le salaire de O_1 est hautement classifié.
- La valeur de l'attribut *Salaire* de l'objet O_2 est polyinstantiée. *25000* est la valeur réelle, alors que *10000* est un leurre. Sur la Figure 2, cette valeur polyinstantiée est représentée par un ensemble polyinstantié [KTT89]. La valeur de l'attribut *Salaire* de l'objet O_2 est un ensemble de paires (valeur, niveau).
- Il n'y a pas d'autre cas de polyinstantiation dans notre exemple. Nous supposons que la polyinstantiation des objets est évitée grâce à la technique du partitionnement des symboles et que seuls des utilisateurs habilités *HIGH* peuvent modifier le schéma (ce qui élimine la polyinstantiation des prédicats *Class'*, *CA'* et *Method'*).
- Dans un soucis de clarté, les classifications des faits associés au prédicat *OA* ne sont pas représentées. Les niveaux de sécurité figurant à droite des attributs d'objet représentent donc les niveaux de sécurité associés aux valeurs d'attribut. Rappelons tout de même que les classifications des faits associés au prédicat *OA* sont complètement déterminées par le théorème (36).

En guise de commentaires, nous soulignons les points suivants:

- Chaque attribut et méthode de la classe *Employé_à_licencier* est classifié à un niveau dominant *C*. Ceci est dû au fait que la classe *Employé_à_licencier* est classifiée au niveau *C* (théorèmes (20) et (22)).
- Chaque attribut de O_2 est au moins *C car* O_2 est un objet confidentiel (théorème (21)). Chaque valeur d'attribut de O_2 est donc au moins *C* (théorème (23)).

- *Instance(O_1,Employé_à_licencier)* et *Instance(O_2,Employé_à_licencier)* sont au moins *C* (théorème (24)).
- *Isa(Employé_à_licencier,Employé)* et *Isa(Employé_entreprise_A,Employé_à_licencier)* sont au moins *C* (théorème (25)).
- Comme *Isa(Employé_entreprise_A,Employé)* est *P*, les deux attributs *Nom* et *Salaire* sont également *P* dans *Employé_entreprise_A* (théorème (28)).
- Pour chaque objet *o* et attribut *a*, la classification de *OA(o,a)* est égale à la classification de *Val(a,o,v)* si *v* n'est pas un ensemble polyinstantié ou est égale à la plus basse classification apparaissant dans l'ensemble polyinstantié (théorème (32)).
- Comme O_1 et O_2 sont respectivement *P* et *C*, le théorème (34) stipule qu'il doit exister une classe *c* et une classe *c'* telles que *Instance'(O_1,c,P)* et *Instance'(O_2,c',C)*. Dans notre exemple, *c=Employé_entreprise_A* et *c'=Employé_à_licencier.*

D'un point de vue théorique, cette base de données est représentée par la théorie que nous avons définie dans les sections 3.1 à 3.6 à laquelle s'ajoutent un certain nombre d'axiomes permettant de dériver toutes les informations représentées graphiquement sur la Figure 2.

4 Le modèle Multi-Vues

Le modèle Vue Unique est un modèle complet, mais semble difficile à implanter directement (voir chapitre précédent). Dans ce chapitre, nous présentons le modèle Multi-Vues [BCGY93a] [BCGY93b] [BCGY94]. Le modèle Multi-Vues est un raffinement du modèle Vue Unique. Il offre les mêmes moyens de protection que le modèle Vue Unique, mais il est plus facilement implantable que le modèle Vue Unique. Intuitivement, le principe du modèle Multi-Vues est de décomposer une base de données à objets *n*-niveaux en *n* vues. Chaque vue est associée avec un niveau donné de classification et contient toutes les données de la base de données à objets multi-niveaux dont le niveau de sécurité est dominé par le niveau de la vue. Une donnée classifiée au niveau *l* se retrouve donc répliquée dans toutes les vues dont le niveau de sécurité domine *l*. Nous verrons toutefois à la Section 5 comment implanter le modèle Multi-Vues en évitant les réplications inutiles.Dans ce chapitre, par soucis de simplicité, nous faisons l'hypothèse que la relation *domine* (≥) associée à l'ensemble des niveaux de sécurité définit un ordre total. Le cas d'un ordre partiel est discuté dans [GAB95].

4.1 Langage

Le langage *L"* que nous utilisons pour formaliser une base de données à objets Multi-Vues est une extension du langage *L'* défini au chapitre précédent. A chaque prédicat *P'* d'arité *n* utilisé pour représenter une base de données à objets multi-niveaux est associé un prédicat *P"* d'arité *n* utilisé pour représenter une base de données à objets Multi-Vues. Intuitivement, si *l* est un niveau de sécurité, alors *P"(t_1, ...,t_n, l)* doit être lu «l'information *P(t_1, ...,t_n)* appartient à la vue de niveau *l*»[1].

4.2 Axiomatique

Nous ajoutons à la liste des axiomes et des théorèmes (1) à (42) du chapitre précédent les axiomes spécifiant les liens entre une base de données à objets multi-niveaux et une base de données à objets Multi-Vues :

- Cet axiome est valable pour tous les prédicats *P'* différent de *Val'*. Il stipule que si *P(t_1,...,t_n)* est une information classifiée au niveau *l*, alors cette information appartient à toute vue de niveau *l'*, avec *l'* ≥ *l*.

$$\forall t_1 \ldots \forall t_n \forall l \forall l', P'(t_1, \ldots, t_n, l) \wedge l' \geq l \rightarrow P''(t_1, \ldots, t_n, l') \quad (43)$$

Pour le prédicat *Val'*, les axiomes sont plus complexes, car nous acceptons que le prédicat *Val'* puisse être polyinstantié.

1. Noter la différence d'interprétation entre Multi-Vues et Vue unique. Par exemple, *Object'(o,l)* doit être lu «l'objet *o* est classifié au niveau *l*», alors que *Object"(o,l)* doit être lu «l'objet *o* appartient à la vue de niveau *l*».

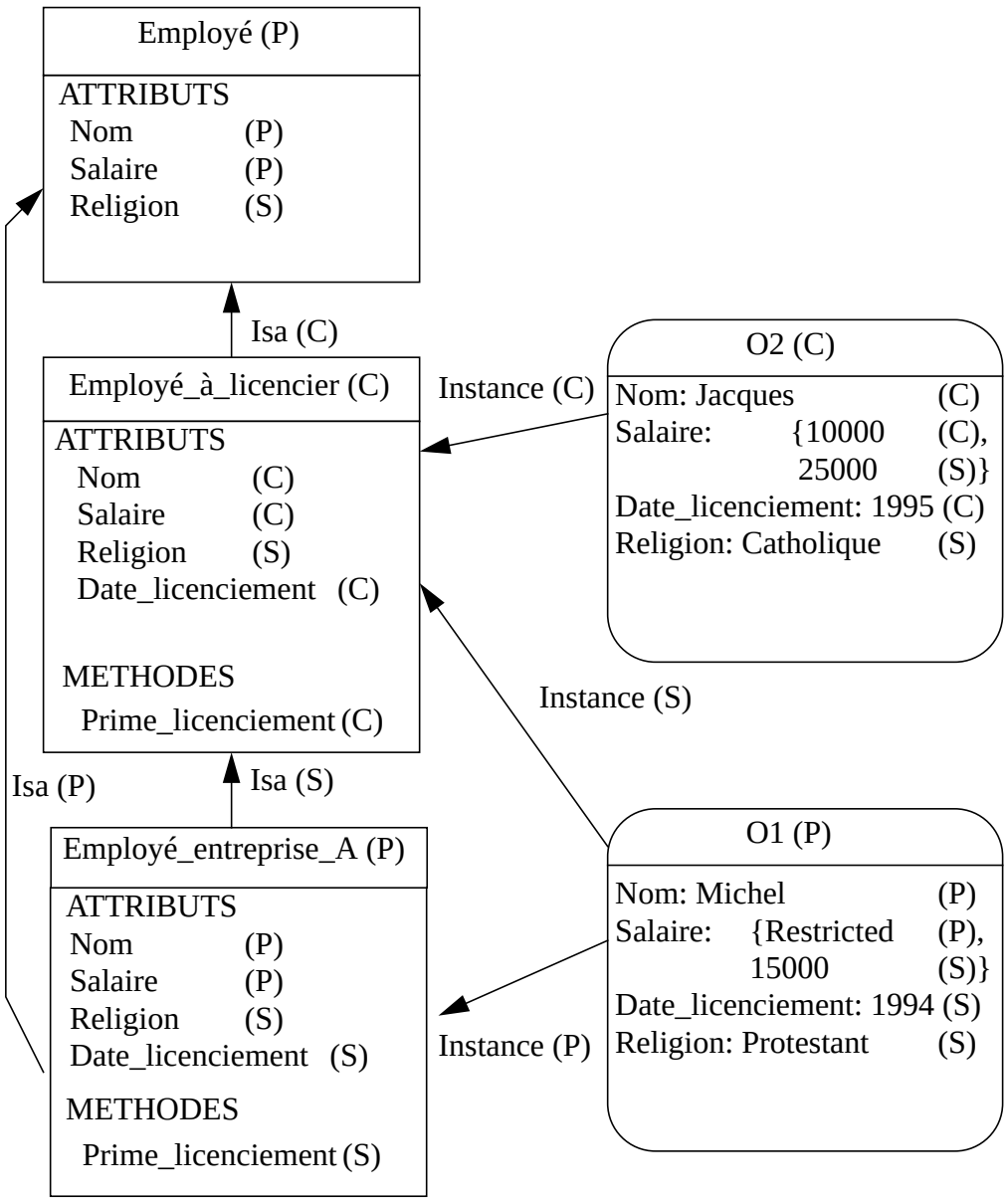

FIGURE 2 Exemple de base de données à objets multi-niveaux

- Cet axiome stipule que si l'information *Val(a,o,v)* est classifiée au niveau *l*, alors il existe une vue de cette information au niveau *l*. L'axiome (46) indique comment créer une vue à un niveau donné *l*, lorsqu'il n'existe pas de valeur *v* tel que *Val'(a,o,v,l)*.

$$\forall a \forall o \forall v \forall l, Val'(a, o, v, l) \rightarrow Val''(a, o, v, l) \quad (44)$$

- Cet axiome stipule que s'il n'existe pas de valeur classifiée au niveau *l* pour l'attribut *a* de l'objet *o* dans le modèle Vue Unique, alors la vue au niveau *l* de la valeur de l'attribut *a* de l'objet *o* est égale à la vue de cette valeur au niveau *l'* qui est juste dominé par le niveau *l*.

$$\forall o \forall a \forall v \forall v' \forall l \forall l', Val''(a, o, v', l') \wedge Justbelow(l', l) \wedge \neg Val'(a, o, v, l) \rightarrow Val''(a, o, v', l) \quad (45)$$

Justbelow(l',l) est une abréviation définie de la façon suivante :

$$\forall l \forall l', Justbelow(l', l) \leftrightarrow l > l' \wedge \neg(\exists l'', l'' > l' \wedge l > l'')$$

- Cet axiome est valable pour tous les prédicats *P"*. Il stipule que si $P(t_1,...,t_n)$ appartient à la vue de niveau *l*, alors il existe un niveau *l'* dominé par le niveau *l* tel que $P(t_1,...,t_n)$ est classifié au niveau *l'*.

$$\forall t_1 \ldots \forall t_n \forall l, P''(t_1, \ldots, t_n, l) \rightarrow \exists l', P'(t_1, \ldots, t_n, l') \wedge l' \leq l \quad (46)$$

Notons que nous pouvons utiliser les axiomes (43) à (46) pour dériver les théoremes[1] contreparties Multi-Vue de chaque axiome non protégé (1) à (15). Par exemple la contrepartie Multi-Vues de l'axiome (1) est la suivante :

$$\forall a \forall c \forall l, CA''(c, a, l) \rightarrow Class''(c, l) \quad (47)$$

Ces théorèmes nous permettent de conclure que chaque vue mono-niveau se «comporte» comme une base de données classique non protégée.

4.3 Exemple de base de données Multi-Vues

Cette section présente la base de données à objets Multi-Vues dérivée de la base de données à objets multi-niveaux présentée à la Section 3.7. Cette base de données à objets est cohérente avec les axiomes et théorèmes (1) à (46). Les hypothèses faites dans cet exemple sont les suivantes :

- Les informations publiques, confidentielles et secrètes sont respectivement écrites avec une fonte normale, *italique* et **grasse**.
- Les attributs hérités n'ont pas été représentés.

En guise de commentaires nous pouvons souligner les points suivants :

- Toute information qui n'est pas une valeur d'attribut est répliquée dans tous les niveaux dominant son niveau de protection (axiome (43)).
- La valeur *Restricted* du *Salaire* et la valeur *Michel* du *Nom* de la vue confidentielle de l'objet *O1* et la valeur *Michel* du *Nom* de la vue secrète de l'objet *O1*, la valeur *Jacques* du *Nom* et la valeur *1995* de la *Date_de_licenciement* de la vue secrète de l'objet *O2* sont dérivées grâce à l'axiome (45).
- Toute information classifiée à un niveau *l* n'apparait pas dans les vues de niveau *l'* strictement dominé par *l*.

D'un point de vue théorique, cette base de données est représentée par la théorie que nous avons définie aux sections 4.1 à 4.2 à laquelle s'ajoutent un certain nombre d'axiomes permettant de dériver toutes les informations représentées graphiquement sur la Figure 3.

5 Implantation

Au chapitre précédent, nous avons défini le modèle Multi-Vues qui est un modèle de sécurité formel pour les bases de données à objets. Ce chapitre propose une implantation naturelle du modèle Multi-Vues. Le principe de base est d'associer à chaque vue de niveau *l* une base de données mono-niveau de niveau *l*. Tout fait de la forme $P''(c_1,...,c_n,l)$ est ainsi enregistré dans une base de données mono-niveau de niveau *l*. Les différentes bases de données mono-niveau ainsi obtenues se comportent chacune comme une base de données classique. La seule exception réside dans l'existence de certains mécanismes de réplication entre les différentes bases de données mono-niveau. Ces mécanismes visent essentiellement à assurer la propagation automatique des mises à jour. Nous montrons ainsi comment implanter l'axiome (45) de réplication des valeurs d'attributs et comment assurer automatiquement la cohérence entre les schémas des différentes bases de données. Il est intéressant de remarquer qu'il existe des liens assez forts entre ces mécanismes et ceux utilisés dans les approches multi-versions. Il semble donc que des techniques d'implantation des versions pourraient probablement être reprises dans le contexte du modèle Multi-Vues.

1. Voir [GAB95] pour une démonstration des théorèmes.

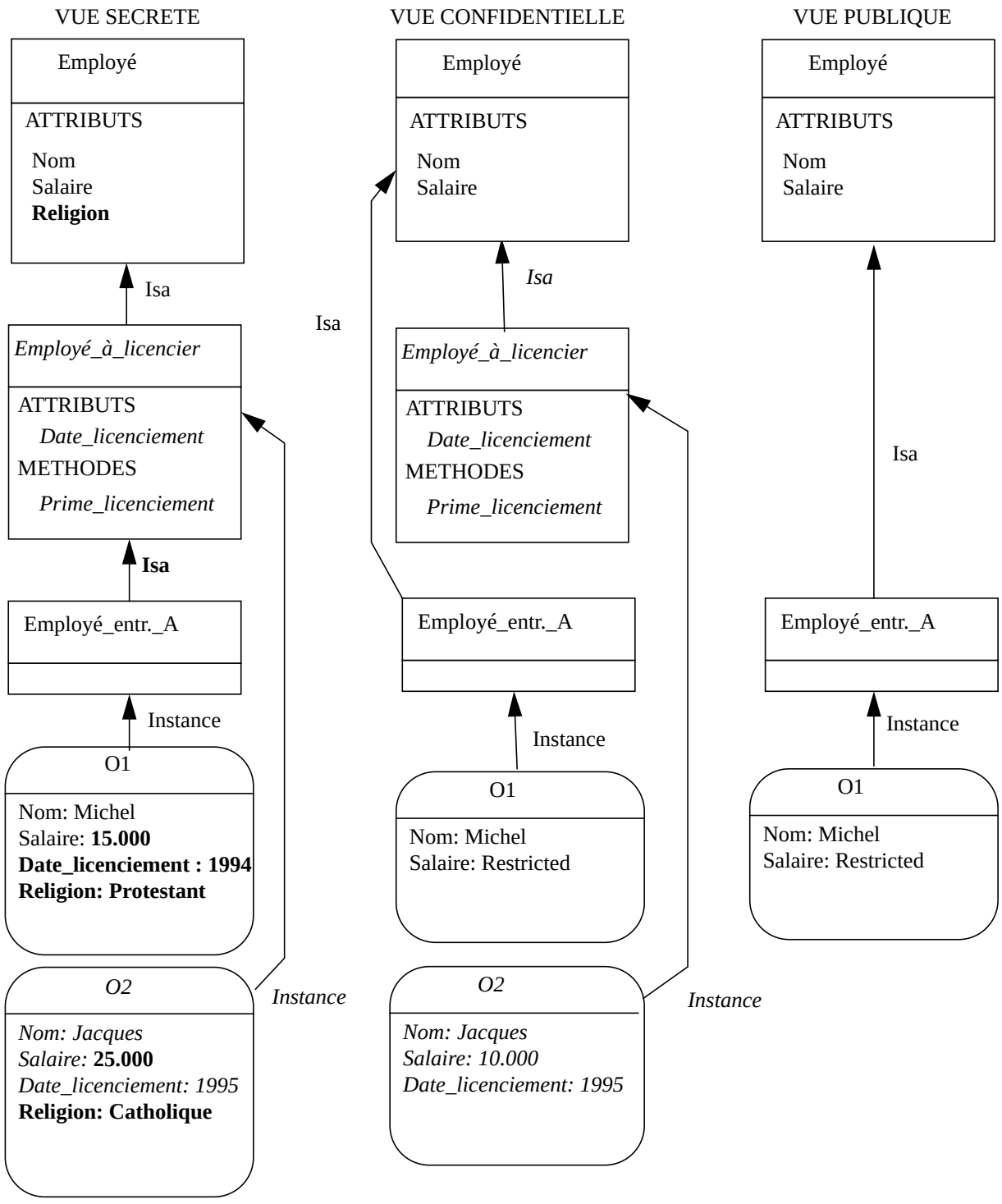

FIGURE 3 Exemple de base de données à objets Multi-Vues

5.1 Principes d'implantation

Les principes d'implantation que nous proposons sont les suivants :

- Nous proposons de décomposer la base de données Multi-Vues en plusieurs bases de données mono-niveau. A chaque vue de niveau *l* est associée une base de données à objets mono-niveau de niveau *l*.
- Dans le modèle Multi-Vues, un objet multi-niveaux et une classe multi-niveaux sont respectivement représentées par plusieurs vues d'objets et de classes. Afin de respecter le principe de l'unicité de l'identificateur d'un objet, chaque vue d'objet et chaque vue de classe doit être identifiée de façon unique. Nous proposons d'implanter ce principe de la façon suivante. Chaque vue à un niveau *l* d'un objet multi-niveaux *o* (respectivement d'une classe multi-niveaux *c*) est identifiée de façon unique par le couple *(o,l)* (respectivement *(c,l)*).
- Nous proposons d'implanter l'axiome (45) de la façon suivante. Toute valeur d'attribut d'une vue *(o,l)* n'ayant pas de valeur d'attribut correspondante au niveau *l* dans l'ensemble polyinstantié de valeurs d'attribut de l'objet multi-niveaux *o*, est égale à un *pointeur*. Ce pointeur est une expression syntaxique qui permet de récupérer dynamiquement la valeur du même attribut dans la vue de niveau immédiatement inférieur.
- Dans [GAB95], nous montrons que pour tous les prédicats *P"* (excepté *Val"*) le théorème suivant peut être dérivé des axiomes (46) et (43):

$$\forall t_1 \ldots \forall t_n \forall l \forall l', P"\ (t_1, \ldots, t_n, l) \wedge l' > l \rightarrow P"\ (t_1, \ldots, t_n, l') \qquad (48)$$

Ce théorème stipule que si une information $P(t_1,...,t_n)$ appartient à une vue de niveau *l*, alors cette information appartient à toutes les vues dont le niveau domine le niveau *l*. Appliqué aux prédicats *CA*" et *Method"* il stipule que les attributs appartenant à une vue de bas niveau d'une classe quelconque sont répliqués dans les vues de plus haut niveau de cette même classe. Afin d'assurer dynamiquement ce principe de réplication, nous créons un lien *Isa* entre une vue de n'importe quelle classe et une vue de cette même classe à un plus bas niveau. Ce lien d'héritage permet d'éviter la réplication d'attributs et de méthodes appartenant à une vue de bas niveau dans une vue correspondante de haut niveau.

L'implantation de la base de données Multi-Vues représentée sur la Figure 3 peut alors être représentée par la Figure 4. Nous pouvons commenter cette figure de la façon suivante :

- Les trois vues publique, confidentielle et secrète (cf FIGURE 3) sont devenues trois bases de données publique, confidentielle et secrète.
- Le concept d'identificateur a été étendu. Toute entité (objet ou classe) est maintenant identifiée par un couple.
- Toute valeur d'attribut d'un objet *(o,l)* n'ayant pas de valeur d'attribut correspondante au niveau *l* dans l'ensemble polyinstantié de valeurs d'attribut de l'objet multi-niveaux *o*, est égale à un *pointeur*. Par exemple la valeur de l'attribut *Nom* de l'objet *(O1,C)* (respectivement de l'objet *(O1,S)*) est égale au pointeur *(O1,P).Nom* (respectivement *(O1,C).nom*). Notons que grâce à ces pointeurs, une mise à jour au niveau public de l'attribut *Nom* de l'objet *(O1,P)* serait automatiquement repercutée aux niveaux confidentiel et secret.
- La réplication des attributs et des méthodes est automatiquement assurée grâce aux liens d'héritage Isa_2, Isa_3, Isa_6, Isa_7 et Isa_8. L'ajout d'un attribut au niveau public dans la classe *(Employé,P)* par exemple, serait automatiquement repercuté aux niveaux confidentiel et secret grâce à ces liens d'héritage.

Finalement, précisons que dans cet article, nous ne décrivons pas les processus de création, mise à jour, suppression. Ces processus ont déjà été largement présentés dans [BCGY93b][BCGY94]. Signalons simplement que la création complète de la base de données de la FIGURE 4 nécessiterait l'exécution d'une succession de trois transactions respectivement au niveau public, au niveau confidentiel et au niveau secret.

6 Conclusion

Notre premier objectif dans cet article a été de développer un modèle *formel* de sécurité multi-niveaux pour les bases de données à objets. La définition de ce modèle s'est faite en trois phases :

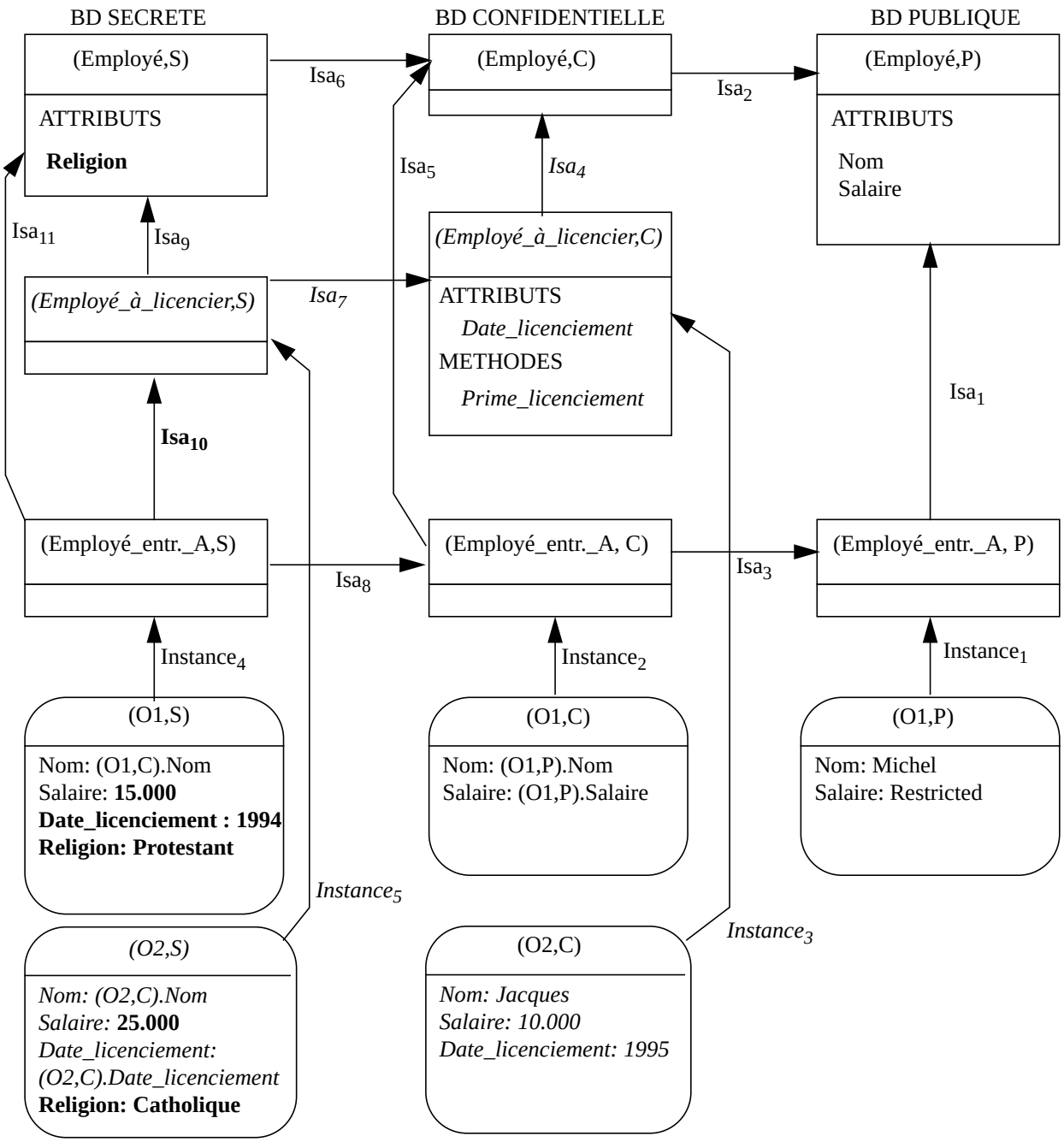

FIGURE 4 Exemple de base de données à objets sécurisée avec le modèle Multi-Vues

- Nous avons tout d'abord défini à la Section 2 un langage du premier ordre qui nous a permis de représenter une base de données à objets non protégée. Nous avons en particulier exprimé grâce à ce langage des contraintes d'intégrité devant être respectées dans n'importe quelle base de données à objets.
- Nous avons ensuite à la Section 3 étendu notre langage pour pouvoir représenter la protection de toute formule atomique représentant une information. Nous avons aussi dérivé des théorèmes généraux qui doivent être respectés lors de l'assignation des niveaux de sécurité aux différentes formules atomiques. Nous avons ainsi défini le modèle Vue Unique.
- Enfin, à la Section 4 nous avons spécifié le modèle Multi-Vues. Le modèle Multi-Vues est un raffinement du modèle Vue Unique. Il est obtenu en décomposant la base de données multi-niveaux Vues Unique en une collection de vues mono-niveau.

Notre deuxième objectif dans ce chapitre a été de développer un modèle *complet* de sécurité multi-niveaux pour les bases de données à objets :

- Nous avons inclus dans notre modèle la possibilité de représenter toute type d'information pouvant être enregistrée dans une base de données à objets. Une classe, un objet, une méthode, un attribut, une valeur d'attribut, un lien d'instantiation, un lien d'héritage sont des informations qui peuvent être représentées et donc de ce fait qui peuvent être protégées.
- Nous avons inclus dans notre modèle la possibilité de gérer des leurres. La gestion des leurres représente en effet, à notre avis, une fonctionnalité importante devant être offerte par un SGBD multi-niveaux.

Notons qu'en raison de la limitation du nombre de pages, cet article n'a pu présenter le problème des mises à jour dans une base de données à objets sécurisée avec la modèle Multi-Vues. Ce problème est traité de façon formelle dans [GAB95]. Finalement, signalons que la plupart des concepts que nous décrivons dans [BCGY93b][BCGY94] et que nous formalisons dans cet article ont été repris par [SMKL95] dans leur implantation d'une politique de sécurité dans le système de gestion de base de données à objets ONTOS.

Bibliographie

[BCGY93a] N. Boulahia-Cuppens, F. Cuppens, A. Gabillon, K. Yazdanian. Techniques to Handle Multilevel Objects in secure Object-Oriented Databases. *Proc of the OOPSLA 93 Conference Workshop on Security in Object-Oriented Systems. Washington D.C., USA. September 93*.

[BCGY93b] N. Boulahia-Cuppens, F. Cuppens, A. Gabillon, K. Yazdanian. Multiview Model for MultiLevel Object-Oriented Database. *Proc. of the Ninth Annual Computer Security Applications Conference. December 93. Orlando, Florida.*

[BCGY94] N. and F. Cuppens, A. Gabillon, K. Yazdanian. "Decomposition of Multilevel Objects in an Object-Oriented DataBase". *Proc. of the third European Symposium on Research In Computer Security (ESORICS). Brighton UK 1994.*

[BDK92] F. Bancilhon, C. Delobel, P. Kanellakis. Building an Object-Oriented Database System, The Story of O_2. *Morgan Kaufmann 1992.*

[EY92] G. Eizenberg and K. Yazdanian. Polyinstantiation in Relationnal Database-Some Semantic Questions. *Research Directions in Database Security, IV 1992.*

[GAB95] A. Gabillon. Sécurité Multi-Niveaux dans les Bases de Données à Objets. *Thèse de Doctorat. ENSAE. 1995.*

[KTT89] T.F. Keefe, W.T. Tsai and M.B. Thuraisingham. SODA : A Secure Object-Oriented Database System. *Computer & Security, Vol 8, N 6, 1989* .

[Lun91] T.F. Lunt. Polyinstantiation: an Inevitable Part of a Multilevel World. *Proc. of the IEEE Workshop on Computer Security Foundations. Franconia, New Hampshire. June 1991.*

[ODMG93] T. Attwood, J. Duhl, G. Ferran, M. Loomis and D. Wade. The Object Database Standard. Release 1.1. *Edited by R.G.G Cattel. Morgan Kaufmann 1993.*

[SJ92] R. Sandhu and S. Jajodia. Polyinstanciation for cover stories. *Computer security - Esorics 92, Toulouse, France. Springer Verlag.*

[SMKL95] M. Schaefer, P.Martel, T.Kanawati and V. Lyons. 'Multilevel Data Model for the Trusted ONTOS prototype. *IX Database Security. Status and Prospects. Elsevier Science Publisher B.V. (North-Holland). IFIP 1995.*

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