Rules for Designing Multilevel Object-Oriented Databases

Frédéric Cuppens,
ONERA/CERT
2 Avenue E. Belin
31055, Toulouse Cedex, France
Fax : 05 62 25 25 93
cuppens@cert.fr

Alban Gabillon
Eastern Mediterranean University
Famagousta
Mersin 10, Turkey
Fax : (00) 90 392 366 12 17
alban@compenet.compe.emu.edu.tr

Abstract

When implementing a multilevel security policy for Object-Oriented Databases (OODBs), several aspects have to be investigated. One of these aspect is the design of multilevel OODBs. In an OODB, data are organized in a complex structure built using different constructs (classes, objects, attributes, links ...). Therefore, a first problem is to determine what constructs of the object-oriented model should be associated with a security level. A second problem is then to define semantics for each assignment of a security level to an object-oriented construct. While assigning the security levels, we have also to be careful with the inference problems which may occur due to the integrity constraints inherent in the object-oriented paradigm. Therefore, a last purpose of this paper is to define a set of general rules to cope with this problem.

1. Introduction

In a multilevel security policy, every subject is associated with a *clearance level* and every piece of information is associated with a *classification level*. A subject is a user or a process. Every user is assigned with his own clearance level whereas every active process running on behalf of a user is associated with a security level dominated by (i.e. is lower than or equal to) the clearance level of this user.

Classifications and clearances are both taken out a set of security levels. For instance, the sensitivity levels Secret (S), Confidential (C) and Unclassified (U) may be used as security levels. Most of the existing multilevel security policies are based on the two following requirements of the Bell & LaPadula model [BL75]:

No Read Up: Subjects are only permitted to read data whose classification is dominated by their clearance.

No Write Down: Subjects are only permitted to write data whose classification dominates (i.e. is greater than or equal to) their clearance.

The “No Write Down” restriction is necessary to prevent a malicious agent (generally called a Trojan Horse) introduced in a high level process, from writing high classified data into a low classified file accessible by a subject cleared at a low level.

When implementing a multilevel security policy for OODBs, several aspects have to be investigated. One of these aspect is the design of multilevel OODBs. Designing multilevel OODBs involves the three following issues :

- Defining the information granularity. In the context of OODBs, defining the information granularity means determining what constructs of the object paradigm have to be associated with security levels. Should the objects be protected ? What about the classes ? the attributes ? the inheritance hierarchy ... ?
- Providing semantics for each association between a security level and a construct. As an example, what is the meaning of assigning a security level to an object ? Does this association protects the object content (its properties) or the object itself (its existence) ?
- Analyzing the effects of the classification of a particular construct on the classifications of other constructs. The knowledge of a low classified data may disclose the knowledge of another higher classified data. Thus, in order to control these unauthorized inferences, assignment of security levels must be carefully performed according to some inference control mechanisms. For example, the knowledge that *a* is an attribute of class *c* discloses the knowledge that *c* is a class. Therefore, the security level assigned to the fact "*c* is a class" must be dominated by the security level assigned the fact "*a* is an attribute of class *c*".

The objective of this paper is to define a set of general principles addressing these three problems. These general principles are stated as *rules* that must be applied when designing multilevel OODBs. These rules are stated using an easy-to-read presentation, that is without using any formal language. However, we argue that these rules are sound and complete and the interested reader is referred to [GAB95] for a proof of this result. Due to space limitation, this proof cannot be included in this paper. Note that [OS94] is another attempt to derive such a set of rules. However, our methodology leads to a much more complete set.

Note that the objective of this paper is not to show how a conceptual multilevel OODB may be implemented. For this issue see the MultiView model [BCGY94][CG96].

The remainder of this paper is organized as follows. In section 2, we define the different kind of data that deserve to be protected in an OODB schema. In section 3 we do the same investigations but for objects. In section 4, we consider the particular case of methods. Finally, section 5 concludes this paper.

2. Schema Protection

The purpose of this section is to determine the semantics of every association between each of the following schema construct and a security level :

- Class
- Class attribute
- Inheritance link

We also state the inference control rules that must be enforced when assigning security levels to these constructs.

Note that methods are also elements of the Database schema but since methods represent the dynamic aspect of the object-oriented model, we prefer to consider them separately in section 4.

2.1 Classes

A class is an element of the Database schema. A class groups methods and attributes. Assigning a security level *l* to a class *c* could have two meanings :

- The security level *l* protects the content of class *c*. In this case an insufficiently cleared user (i.e. a user cleared at level *l'* with *l'*<*l*) is not permitted to know the list of properties included in class *c*.
- The security level *l* protects the existence of the class. In this case an insufficiently cleared user is not authorized to know that class *c* belongs to the Database.

We consider that an object represents a complex real world entity containing different pieces of information, each of them having its own sensitivity. Since a class groups objects sharing the same properties, we argue that each class property (attribute or method) may be separately classified. Thus, it would be restrictive to assign a security level to a class in order to globally classify its content because it would prevent us from separately classifying each class property. On the other hand, in some applications, assigning a security level to a class in order to protect its existence represents a very important functionality. So, we state the following rule :

Rule 1 Each class is associated with a security level. Assigning a security level to class *c* protects the existence of class *c* i.e. the fact that *c* is a class of the Database.

In Figure 1, two classes are represented: the classes AIRCRAFT and HYPERSONIC_AIRCRAFT. We assume we have three security levels only : the Secret security level denoted by the letter S, the Confidential security level denoted by the letter C and the Unclassified security level denoted by the letter U. Secret information is in **bold**, confidential information is in *italic* and unclassified information is in regular font. The existence of class AIRCRAFT is unclassified. Every user has the right to know that this class belongs to the Database. By contrast, the class HYPERSONIC_AIRCRAFT is a confidential class. Only secret and confidential users can see it.

2.2 Class attributes

Assigning a security level *l* to an attribute *a* of type *t* in a class *c* could have two meanings[1]:

- The security level *l* protects the fact that attribute *a* of class *c* is of type *t*.
- The security level *l* protects the existence of attribute *a* i.e. the fact that attribute *a* belongs to class *c*.

In our opinion, having the possibility to protect the fact that a given attribute belongs to a given class is a very important functionality. In Figure 1, the fact that the Weapons attribute belongs to class AIRCRAFT is not protected but the fact that the Nuclear_Bomb attribute belongs to class AIRCRAFT is protected at the secret level.

[1] One might consider a third meaning: The security level *l* is viewed as a constraint limiting the security level that may be assigned to the value of the attribute *a* in every instance of the class *c*. This meaning is used by [KTT89] in the SODA model but with the following generalization: Each class attribute is actually associated with an interval of security levels. The security level assigned to the value of an attribute of a given instance is constrained to be chosen from the interval associated with the corresponding class attribute.

Protecting the fact that a given class attribute is of a given type is perfectly acceptable although we think this does not represent a crucial functionality. For instance, in Figure 1, we could associate the fact « the type of Weapons attribute of class AIRCRAFT is set of string » with a confidential (or secret) security level. This would allow us to hide the fact that an aircraft may have several weapons. However adopting this functionality would lead us to manage two distinct security levels for each class attribute : one security level to protect the existence of the class attribute and one security level to protect its type. For the sake of simplicity, we prefer to associate each class attribute with a single security level protecting the existence of the attribute and, in the same time, implicitly protecting its type. So, we state the following rule :

Rule 2 Each class attribute is associated with a security level. Assigning a security level to an attribute *a* of a class *c* protects its existence, i.e. the fact that *a* is an attribute of *c*.

The knowledge « *a* is an attribute of class *c* » discloses the fact that *c* is a class. In Figure 1, if the fact « Speed is an attribute of class HYPERSONIC_AIRCRAFT » were unclassified it would disclose to unclassified users the confidential fact « HYPERSONIC_AIRCRAFT is a class of the Database ». In order to avoid such a disclosure of confidential information every attribute of class HYPERSONIC_AIRCRAFT must be at least confidential. Generalization of this principle leads to the following inference control rule :

Rule 3 The security level assigned to an attribute *a* of a class *c* must dominate the security level assigned to *c*.

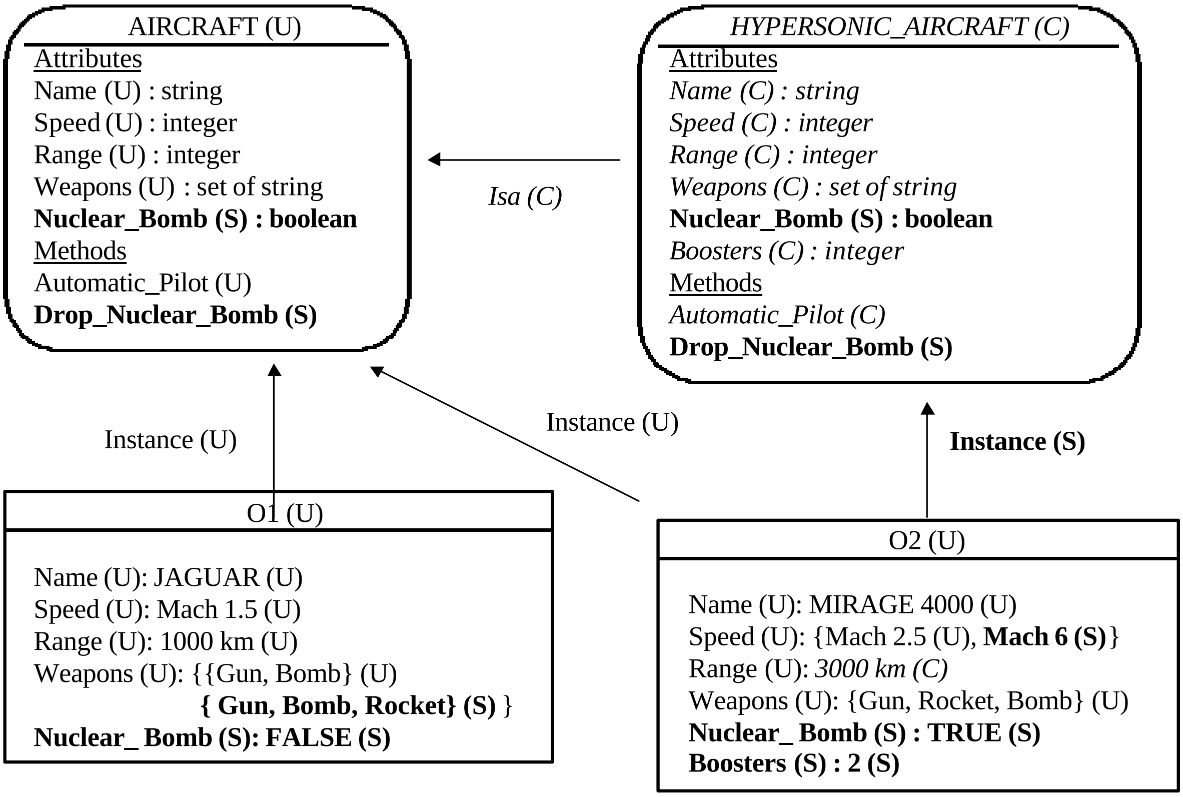

Figure 1: Example of a multilevel object-oriented Database

2.3 Inheritance Links

Classes inherits properties through inheritance links. In Figure 1, all properties of class AIRCRAFT are inherited by class HYPERSONIC_AIRCRAFT. Assigning a security level to an inheritance link between a class *c'* and a class *c* protects the existence of this link i.e. it protects the fact that *c'* inherits *c*. So we state the following rule :

Rule 4 Each inheritance link is associated with a security level. Assigning a security level to an inheritance link between a class *c* and a class *c'* protects the fact that *c'* inherits from *c*.

The knowledge « class *c'* inherits from class *c* » discloses the fact that *c* and *c'* are classes. In Figure 1, if the link between AIRCRAFT and HYPERSONIC_AIRCRAFT were unclassified, it would disclose to unclassified users the fact that HYPERSONIC_AIRCRAFT is a class of the Database. In our example, the purpose of the confidential security level assigned to the inheritance link between class HYPERSONIC_AIRCRAFT and class AIRCRAFT is to avoid the disclosure of the existence of confidential class HYPERSONIC_AIRCRAFT. Generalization of this principle leads to the following inference control rule :

Rule 5 The security level assigned to an inheritance link between two classes *c* and *c'* must dominate the least upper bound of the security level assigned to *c* and the security level assigned to *c'*.

The knowledge « *a* is an attribute of class *c* and class *c'* inherits from *c* » discloses the fact « *a* is an attribute of class *c'* » (since *a* is inherited by *c'*). In Figure 1, the existence of class attribute Nuclear_Bomb in class HYPERSONIC_AIRCRAFT is secret whereas the existence of the inheritance link between AIRCRAFT and HYPERSONIC_AIRCRAFT is confidential. If attribute Nuclear_Bomb in class AIRCRAFT were confidential, it would disclose to confidential users that Nuclear_Bomb is an attribute of HYPERSONIC_AIRCRAFT. So in our example, either the security level assigned to attribute Nuclear_Bomb in class AIRCRAFT must be at least secret or the security level assigned to the inheritance link between AIRCRAFT and HYPERSONIC_AIRCRAFT must be at least secret. Generalization of this principle leads to the following inference control rule :

Rule 6 The least upper bound of the security level assigned to an attribute *a* of a class *c* and the security level assigned to an inheritance link between a class *c'* and *c* must dominate the security level assigned to *a* in class *c'*.

3. Instances Protection

Defining the information granularity for objects is subject to a large debate. Basically, there are two possible approaches:

1. The Single Level Object approach [ML92][JK90]. In this approach, every object is assigned with a security level. This security level protects all the information encapsulated in this object. Security models based on this approach are easy to implement. Indeed, the logical multilevel OODB can be defined as a set of physical single level Databases. Each single level Database of level *l* contains all the objects of level *l*. Mandatory access controls based on the Bell & LaPadula model can be easily interpreted. However, in this approach, the expressive power of the multilevel policy is poor.

2. The Multilevel Object approach [KTT89][VB91]. In this approach, every attribute of every object is assigned with a security level. This security level protects the value of this attribute. Security models based on this approach provide the multilevel policy with higher expressive power.

In our opinion, the single level object approach is too restrictive. An object may represent a complex real world entity and may contain data of different sensitivities. So, in this article, we adopt the Multilevel Object approach.

Note that directly implementing multilevel objects is impractical since it would lead to manage one physical multilevel Database. Such an implementation is not realistic since it is very difficult to define secure operations on a single multilevel file. This explains why some models using multilevel objects propose a technique to decompose a multilevel object into several single level objects [BJ93][JK90][CG96]. However this implementation issue is not the purpose of this paper.

The objective of this section is to determine the semantics of every association between each of the following constructs and a security level :

- Object
- Inheritance link
- Object attribute
- Attribute value

We also state the inference control rules that must be enforced when assigning security levels to these constructs.

3.1 Objects

We consider that every property of an object must be separately classified. Thus, assigning a security level to an object in order to protect its content is useless. However as for classes, it is useful to assign a security level to each object in order to protect its existence in the Database. So, we state the following Rule :

Rule 7 Each object is assigned with a security level. Assigning a security level to an object *o* protects the fact that *o* is an object of the Database.

In Figure 1, there are two objects O1 and O2 whose existences are unclassified.

3.2 Instance Links

Objects inherit attributes through instance links. Assigning a security level to an instance link between an object *o* and a class *c* protects the existence of this link i.e. the fact that object *o* is an instance of class *c*. So we state the following rule :

Rule 8 Each instance link is assigned with a security level. Assigning a security level to an instance link between an object *o* and a class *c* protects the fact that *o* is an instance of *c*.

In Figure 1, the fact that O1 is an aircraft is unclassified. On the other hand, the fact that O2 is an aircraft is unclassified whereas the fact that it is an hypersonic aircraft is secret. This means that unclassified and confidential users should believe that O2 is a classic aircraft and should ignore that it is actually an hypersonic aircraft.

The knowledge « object *o* is an instance of class *c* » discloses the fact that *o* is an object and *c* is a class. In Figure 1, if the link between O2 and HYPERSONIC_AIRCRAFT were unclassified, it would disclose to unclassified users the fact « HYPERSONIC_AIRCRAFT is a class of the Database ». In order to avoid such a disclosure, the security level assigned to the instance link between O2 and HYPERSONIC_AIRCRAFT must be at least confidential. Generalization of this principle leads us to the following inference control rule :

Rule 9 The security level assigned to the instance link between an object *o* and a class *c* must dominate the least upper bound of the security level assigned to *o* and the security level assigned to *c*.

One of the integrity constraints inherent in the object paradigm is the following : an object must be instance of at least one class. Therefore, the knowledge « *o* is an object » discloses the fact « there exists at least one class *c* such that *o* is instance of *c* ». In Figure 1, if the instance link between O2 and AIRCRAFT were confidential, unclassified users would not be able to see any parent class for object O2. Hence, these unclassified users would be able to determine the existence of a protected instance link involving O2. In order to avoid such a disclosure, there must be at least one unclassified instance link involving object O2. Generalization of this principle leads to the following inference control rule[2] :

Rule 10 If an object *o* is associated with a security level *l*, then *o* must be instance of at least one class *c* such that the instance link between *o* and *c* is classified at level *l*.

Many authors [KTT89][Lun90][ML92] add the following more restrictive rule:

> If an object *o* is an instance of a class *c*, then the security level of *o* must dominate the security level of *c*.

This rule is unnecessarily restrictive if one considers that (a) an instance link may be explicitly classified and (b) an object may be an instance of several different classes. In particular, this rule would not enable us to consider that O2 is an instance of class AIRCRAFT (at the unclassified level) but also an instance of class HYPERSONIC_AIRCRAFT (at the secret level).

3.3 Object Attributes

Assigning a security level *l* to an attribute *a* of an object *o* could have two meanings :

- The security level *l* protects the value of attribute *a* in object *o*.
- The security level *l* protects the existence of attribute *a* in object *o*.

Having the possibility to protect the value of an attribute is obviously an important requirement. Therefore, protection of attribute values will be separately considered in section 3.4.

[2] From this reasoning, we can derive that " If the object *o* is associated with the security level *l*, then the object *o* must be instance of at least one class *c* such that the instance link between the object *o* and the class *c* is classified at a level dominated by *l*". However, due to Rule 8, the security level assigned to the instance link between the object *o* and the class *c* must dominate the level *l*. Combination of these results leads to Rule 10.

However, having the possibility to protect the fact that a given attribute belongs to a given object is also an important functionality. For example, in Figure 1, the existence of Nuclear_Bomb attribute in object O1 is secret. We state the following Rule :

Rule 11 Each object attribute is associated with a security level. Assigning a security level to an attribute *a* of an object *o* protects the fact that *a* is an attribute of *o*.

The knowledge « *a* is an attribute of object *o* » discloses the fact that *o* is an object. So we state the following inference control rule :

Rule 12 The security level assigned to an attribute *a* of an object *o* must dominate the security level assigned to *o*.

The knowledge « *a* is an attribute of class *c* and *o* is an instance of class *c* » discloses the fact « *a* is an attribute of object *o* » (since *a* is inherited by *o*). In Figure 1, the existence of attribute Boosters in class HYPERSONIC_AIRCRAFT is confidential whereas the existence of this attribute in object O2 is secret. If the instance link between O2 and HYPERSONIC_AIRCRAFT were confidential, it would disclose to confidential users that Boosters is an attribute of O2. So in our example, either the security level assigned to attribute Boosters in class HYPERSONIC_AIRCRAFT must be at least secret or the security level assigned to the instance link between O2 and HYPERSONIC_AIRCRAFT must be at least secret. Generalization of this principle leads to the following inference control rule :

Rule 13 The least upper bound of the security level assigned to an attribute *a* in a class *c* and the security level assigned to an instance link between an object *o* and *c* must dominate the security level assigned to *a* in object *o*.

Objects inherits attributes from classes. Thus, the knowledge « *a* is an attribute of object *o*» discloses the fact that there is at least one class *c* such that *o* is an instance of *c* and *a* is an attribute of *c*. In Figure 1, if attribute Nuclear_Bomb of O1 were confidential, it would disclose to confidential users that O1 is instance of at least one class having a secret Nuclear_Bomb attribute. In order to avoid such a disclosure, Nuclear_Bomb attribute of O1 must be secret. Generalization of this principle leads to the following inference control rule :

Rule 14 If an attribute *a* of an object *o* is classified at level *l*, then *o* must be instance of at least one class *c* such that *a* is an attribute of *c* and *l* is equal[3] to the least upper bound of the security level assigned to *a* in class *c* and the security level assigned to the instance link between *o* and *c*.

3.4 Attributes values

Assigning a security level to an attribute value *v* of an attribute *a* of an object *o* protects the fact that the value of the attribute *a* of object *o* is equal to *v*. For instance, in Figure 1, the fact that the name of O1 is JAGUAR is an unclassified information. The fact that the weapons of O2 are {Gun, Bomb, Rocket} is also an unclassified information. The fact that the range of object O2 is 3000 km is a confidential information and so on ... So we state the following Rule.

[3] Due to Rule 13, it cannot be greater.

Rule 15 Each attribute value is associated with a security level. Assigning a security level to a value *v* of an attribute *a* of an object *o* protects the fact that the value of *a* in object *o* is equal to *v*.

The knowledge « *v* is the value of attribute *a* of object *o* » discloses the fact « *a* is an attribute of object *o* ». So we state the following inference control rule :

Rule 16 The security level assigned to a value *v* of an attribute *a* of an object *o* must dominate the security level assigned to *a* in object *o*.

One of the integrity constraints of the object paradigm is the following : an object attribute must have a value[4]. Therefore, the knowledge « *a* is an attribute of object *o* » discloses the fact « there exists a value *v* for attribute *a* of object *o* ». In Figure 1, the Range attribute of O2 is unclassified whereas the value 3000 of this attribute is confidential. Unclassified users can see the Range attribute of O2, but they cannot see any value for this attribute. Therefore, these users can guess the existence of a high classified value for the Range attribute of O2. Basically, there are two solutions to avoid these kind of disclosure :

- Stating a rule enforcing that « the security level associated to the value *v* of an attribute *a* of an object *o* must be equal to the security level associated to *a* in object *o* ». Clearly, this Rule is too restrictive. It would considerably downgrade the expressive power of the security policy since it leads to assign a single security level protecting both the existence of the object attribute and its value. We reject it without any hesitation.
- Allowing object attributes to be *polyinstantiated* [DLSSH88]. Polyinstantiation means that an object attribute may be associated with different values. These values are *distinguished* by their classification levels. This functionality is a means to implement cover stories. A cover story is a low classified data deliberately introduced to hide the existence of another higher classified data. In Figure 1, the value Mach 2.5 of the attribute Speed of O2 is typically a cover story. This value is a lie provided to any unclassified user who wants to know the speed of the Mirage 4000. Since unclassified users are provided with this lie, they cannot guess the existence of a secret speed value for the Mirage 4000. Cover stories are also used to partially hide the truth. The value {Gun, Bomb} of the Weapons attribute of O1 is not really a lie. It rather represents an incomplete information since the Jaguar is also equipped with rockets, as indicated in the secret value {Gun, Bomb, Rockets}[5].

We state the following rule :

Rule 17 An object attribute of level *l* may be associated with different values distinctly classified with a security level dominating the level *l*.

There is no particular rule to decide whether an object attribute must be polyinstantiated or not. The decision to disclose or not to low cleared users the existence of a high classified data is up to the Database Designer. In the particular case of the Speed attribute of object O2, the decision is not to disclose the existence of the secret value Mach 6, leading to the insertion of the cover story Mach 2.5. On the other hand, in the case of the Range attribute of object O2, the Database

[4] this value may be a NULL value.

[5] A similar comment may be done for the object O2 which is an instance of AIRCRAFT at the unclassified level. This represents an incomplete information since O2 is an instance of HYPERSONIC_AIRCRAFT but this is secret information.

Designer decided that it was not important to hide to unclassified users the fact that the range value of the MIRAGE 4000 was a protected information. Thus, the Database Designer did not insert a cover story for this attribute at the unclassified level.

The value of an object attribute may be another object. In this case, such a value represents an *association* between two objects. Thus, the knowledge "the value of attribute *a* of object *o* is object *o'* " discloses the fact "*o'* is an object". So we state the following inference control rule:

Rule 18 If a value *v* of an attribute *a* of an object *o* is equal to an object *o'* then the security level assigned to *v* must dominate the security level assigned to *o'*.

In Figure 1, for the sake of simplicity, we did not represent such an association between two objects. So, in order to illustrate Rule 18, let us assume that we modify figure 1 by creating a third class MISSION and let us consider that O3 is a *secret* instance of this class. Let us now add an attribute Used_in to the class AIRCRAFT to represent the fact that a given aircraft is used in a given mission and, let us assume that aircraft O1 is used in mission O3. This means that the value of attribute Used_in of object O1 is equal to O3. In this case Rule 18 states that the security level assigned to this value O3 must be at least secret since we assumed the existence of mission O3 to be secret.

4. Methods

A method is a program implementing an operation. A method is attached to a class. It is associated with a signature and a code. Assigning a security level *l* to a method *m* of a class *c* with signature *s* and code *p* could have four meanings :

- The security level *l* protects the existence of method *m* i.e. the fact that method *m* belongs to class c.
- The security level *l* protects the signature of the method i.e. the fact that the signature of method *m* of class *c* is *s*.
- The security level *l* protects the code of the method i.e. the fact that the code of method *m* of class *c* is *p*.
- The security level *l* protects the execution of method *m* of class *c*.

Let us examine these four possibilities.

A method is a class property. Having the possibility to protect the fact that a given method belongs to a given class is an important functionality. In Figure 1, we consider that the security levels assigned to the methods correspond to this first meaning. Therefore, the fact that the Automatic_Pilot method belongs to class AIRCRAFT is not protected but the fact that the Drop_Nuclear_Bomb method belongs to class AIRCRAFT is protected at the secret level. So we state the following rule:

Rule 19 Each method is associated with a security level. Assigning a security level to a method *m* of a class *c* protects its existence, i.e. the fact that *m* is a method of *c*.

The knowledge « *m* is a method of class *c* » discloses the fact that *c* is a class. In Figure 1, if the fact « Automatic_Pilot is a method of class HYPERSONIC_AIRCRAFT » were unclassified, it would disclose to unclassified users the confidential fact « HYPERSONIC_AIRCRAFT is a class of the Database ». In order to avoid such a disclosure of confidential information every method of the class HYPERSONIC_AIRCRAFT must be at least confidential. Generalization of this principle leads to the following inference control rule :

Rule 20 The security level assigned to a method *m* of a class *c* must dominate the security level assigned to *c*.

The knowledge « *m* is a method of class *c* and class *c'* inherits from *c* » discloses the fact « *m* is a method of class *c'* » (since *m* is inherited by *c'*). In Figure 1, the existence of method Drop_Nuclear_Bomb in class HYPERSONIC_AIRCRAFT is secret whereas the existence of the inheritance link between AIRCRAFT and HYPERSONIC_AIRCRAFT is confidential. If the method Drop_Nuclear_Bomb in class AIRCRAFT were confidential, it would disclose to confidential users that Drop_Nuclear_Bomb is a method of HYPERSONIC_AIRCRAFT. So in our example, either the security level assigned to method Drop_Nuclear_Bomb in class AIRCRAFT must be at least secret or the security level assigned to the inheritance link between AIRCRAFT and HYPERSONIC_AIRCRAFT must be at least secret. Generalization of this principle leads to the following inference control rule :

Rule 21 The least upper bound of the security level assigned to a method *m* in a class *c* and the security level assigned to an inheritance link between a class *c'* and *c* must dominate the security level assigned to *m* in class *c'*. [6]

Let us now turn to the second possibility. Protecting the code of a method may be an important requirement. Since a method is attached to a class, the code of a method discloses much information about the behavior of the instances of the class. Thus, the code of a method may be associated with its own security level. Notice however that classifying the code of methods is not illustrated in Figure 1 (the code of methods is not presented in Figure 1, anyway). So we state the following rule:

Rule 22 The code of a method is associated with a security level. Assigning a security level to the code *p* of a method *m* of a class *c* protects the fact that the code of *m* in class *c* is *p*.

The knowledge "the code of method *m* of class *c* is *p*" discloses the fact "*m* is a method of class *c*". So we state the following inference control rule :

Rule 23 The security level assigned to the code *p* of a method *m* of a class *c* must dominate the security level assigned to *m* in class *c*.

Observing the code of a method may disclose the *existence* of some object-oriented constructs referenced in the statements of this code. So we state the following inference control rule:

Rule 24 The security level assigned to the code *p* of a method *m* of a class *c* must dominate the least upper bound of security levels protecting the existences of object-oriented constructs referenced in *p*.

For instance, it is likely that method Drop_Nuclear_Bomb references attribute Nuclear_Bomb. Therefore, Rule 24 requires that the security level assigned to the code of method

[6] Note that attributes can be seen as a special case of methods. Under this view, rules 2,3 and 6 are subsumed by the rules 19,20 and 21.

Drop_Nuclear_Bomb must dominate the security level assigned to the existence of attribute Nuclear_Bomb in class AIRCRAFT.

The third possibility is to protect the signature of a method. This is perfectly acceptable but, as for attributes type, we think that this does not represent a crucial requirement. So we do not consider this possibility.

Finally, the last possibility is to assign a security level to a method in order to protect its execution. This would mean that an insufficiently cleared user would not be able to execute the method. We claim that this kind of protection is not relevant to a confidentiality problem. More precisely, from the confidentiality point of view, the only restriction upon the execution of a method is stated by the following rule:

Rule 25 A methods *m* of a class *c* may be executed by any subject who can see this method i.e. any subject whose clearance dominates the security level assigned to this method.

Of course, additional security requirements may be specified so that some subjects are authorized to execute a given method and other subjects are not authorized to do so. For instance, depending on its role in the Database system, a given subject may be authorized to execute the method Automatic_Pilot whereas another subject may be not authorized to do so. Our claim is that it is not the purpose of a multilevel security policy to define these security requirements but instead it is the purpose of a Discretionary Access Control policy (DAC, see for instance [PHD88]) or a Role Based Access Control policy (RBAC, see for instance [DHDT95]). Thus we shall not consider that a security level may be used to protect the execution of a method.

After starting execution, a method becomes a *subject* associated with a *clearance* level *l* inherited from the user on behalf of whom it is running. If this executing method tries to access data associated with a security level strictly dominating level *l*, then this method will normally fail due to insufficient clearance.

5. Conclusion

Our objective in this paper was to define some general principles for designing multilevel OODBs. We first determine the constructs of the object paradigm that should be associated with security levels. Then, we define semantics for each association between a security level and a construct. Finally, we state some inference control rules that must be enforced when assigning security levels. In [CG96], we provide a formal method allowing us to derive from the integrity constraints of the object paradigm, all of the inference control rules presented in this paper. In [GAB95], we show that the set of inference control rules derived by this formal method is sound and complete according to the integrity constraints of the object paradigm.

These rules do not provide us with a complete methodology for designing secure Database applications. In MOMT (Multilevel Object Modelling Technique) [MST96], designing such an application involves the three following steps:

1. Analyzing the requirements to detect potential security vulnerabilities.

2. Designing the multilevel Database.

3. Designing the modules of the automated system.

Clearly, Rules 1 to 25 stated in this paper should be used in second step of this methodology. To our knowledge, we are not aware of any proposal considering a complete list of rules such as the one we define in this paper. For example, only a small subset of rules 1 to 25 are effectively taken into consideration in MOMT. Moreover, other proposals do not always provide clear semantics for the association between an object-oriented construct and a security level.

An issue not addressed in this paper is implementation of a conceptual multilevel Database. As an example, implementing as a single multilevel file the Database represented in Figure 1 is not realistic. For security reasons, classified data of a *n*-level Database should be distributed over a set of *n* single level Databases. See MultiView model [BCGY94][CG96] for a discussion and for a presentation of the implementation principles.

References

[BCGY94] 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. UK 1994. Springer Verlag.

[BJ93] E. Bertino, S. Jajodia. *Modelling Multilevel Entities Using Single Level Objects.* Proc. of the 3th international conference on Deductive and Object-Oriented Databases (DOOD93).

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

[CG96] F. Cuppens and A. Gabillon. *A Logical Approach to Model a Multilevel Object-Oriented Database.* Proc. of the 10th Annual IFIP WG 11.3 Working Conference on Database Security. Como Italy.1996.

[DHDT95] S. Demurjian, M. Hu, T. Dagget and T. Ting. *User-Role Based Security Enforcement Mechanisms for Object-Oriented Systems and Applications*. Proc. IFIP 11.3 WG 11.3 Workshop on Database Security. Rensselaer USA 1995.

[DLSSH88] D. Denning, T. Lunt, R Schell, W Shockley and M. Heckman. *The Sea View Security Model*. Proc. of the 1988 IEEE Symposium on Research in Security and Privacy.

[GAB95] A. Gabillon. *Sécurite multi-niveaux dans les bases de données à objets*. Ph.D. dissertation. ENSAE 1995.

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

[KTT89] T. Keefe, W. Tsai and B 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. Result of the IFIP WG 11.3 Workshop on Database Security.

[ML92] J.K. Millen and T.F. Lunt. *Security for Object-Oriented Database Systems.* Proc. of the 1992 IEEE Symposium on Research in Security and Privacy.

[MST96] D. Marks, P. Sell and B. Thuraisingham. MOMT*: A Multilevel Object Modelling Technique for designing secure database applications*. Journal of Object-Oriented Programming. July-August 1996.

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

[PHD88] H. Pfefferle, M. Hartig, K. Dittrich. *Discretionary Access Control in Structurally Object-Oriented Database Systems*. Proc. IFIP 11.3 WG 11.3 Workshop on Database Security. 1988.

[VB91] V. Varadharajan and S. Black. *Multilevel Security in a Distributed Object-Oriented System.* Computer and Security 10. 1991.