Multilevel secure object-oriented data model — issues on noncomposite objects, composite objects, and versioning

by Bhavani Thuraisingham

The MITRE Corporation, Burlington Road, Bedford, MA 01730

I. INTRODUCTION
Object-oriented systems are gaining increasing popularity due to their inherent ability to represent conceptual entities as objects, which is similar to the way humans view the world. This power of representation has led to the development of new generation applications such as computer-aided design/computer-aided modeling (CAD/CAM), multimedia information processing, artificial intelligence, and process control systems. However, the increasing popularity of object-oriented database management systems should not obscure the need to maintain security of operation. That is, it is important that such systems operate securely to overcome any malicious corruption of data as well as to prohibit unauthorized access to and use of classified data. For many applications, it is also important to provide multilevel security. Consequently, multilevel database management systems are needed to ensure that users cleared to different security levels access and share a database with data at different security levels in such a way that they obtain only the data classified at or below their level.

In a recent article in this journal [Thura90a], we discussed the multilevel security issues of an object-oriented database system and described a simple multilevel object-oriented data model. Like this model, most secure object-oriented data models developed since then (see, for example, [Kee90, Thura90, Mille90]) have considered only the simple attributes of an object. For example, the title, author, publisher, and date of publication are simple attributes of a book. Such attributes can also be easily represented by a relational model. In contrast, the book cover, preface, introduction, various chapters, and references form the components of a book and cannot be treated as simple attributes of an object. The book, consisting of these components, has to be collectively treated instead as a composite object. This was addressed by Kim et al. [Kim87, Kim88] in a nonmultilevel secure environment. Composite objects involve the IS-PART-OF relationship between objects. This relationship is based on the notion that an object is part of another object. Note that it is not possible to treat composite objects using a relational model without placing a tremendous burden on the application program to maintain the structure of the complex structures, thus conferring upon the object model another advantage over the relational model.

Hypermedia systems, CAD/CAM systems, and knowledge-based systems are inherently more complex by their very nature and, therefore, can be handled effectively only if their components are treated using composite objects. For example, in hypermedia systems each document is a collection of text, graphics, images, and voice and needs to be treated as a composite object. In a CAD/CAM system, the design of a vehicle consists of designs of its components such as chassis, body, trunk, engine, and doors. Knowledge-based systems are being applied to a wide variety of applications in medicine, law, engineering, manufacturing, process control, library information systems, and education. These applications need to process complex structures. Therefore, support for composite objects in knowledge-based applications is essential.

In many object-oriented applications, such as Hypermedia systems and CAD/CAM, it is necessary to maintain documents and designs that evolve over time. In addition, alternate designs of an entity should also be represented because of the need for choice. If security has to be provided for these applications, then some form of version management should be supported by secure database systems. Another advantage to providing version management in secure applications is the uniform treatment of polyninstation and versioning. Note that for many secure applications it may be necessary to support polyninstation where users at different security levels have different views of the same entity. Polyninstation can be regarded as a type of versioning that cuts across security levels. Therefore, design of the version management component of an object-oriented data model can also be extended to include polyninstation.

In this article, we will continue with our investigation on multilevel security in object-oriented database systems and explore the issues on noncomposite objects, composite objects, and versioning. The organization of this paper is as follows: In Section 2
we discuss the issues involved in supporting noncomposite objects in a multilevel environment. Issues on composite objects are described in Section 3. Version management is discussed in Section 4. The paper is concluded in Section 5.

We assume that the reader is familiar with concepts in object-oriented database systems. For a discussion on object-oriented data model concepts such as noncomposite objects, composite objects, complex objects, IS-A hierarchy, and IS-PART-OF hierarchy, we refer to the ORION data model described in [Bare87, Kim87]. We also assume that the reader is familiar with concepts in multilevel secure database management systems (MLS/DBMS). In an MLS/DBMS, users cleared at different security levels access and share a database consisting of data at different security levels. The security levels may be assigned to the data depending on content, context, aggregation and time. It is generally assumed that the set of security levels form a partially ordered lattice with Unclassified < Confidential < Secret < Top Secret. An effective security policy for an MLS/DBMS should ensure that users only acquire the information at or below their level. An overview of multilevel database management systems was given in [Thur90a]. A useful starting point for concepts in multilevel database management systems is the Air Force Summer Study Report [AirFo83].

2. NONCOMPOSITE OBJECTS IN MULTILEVEL DATABASES

Various approaches can be taken to handle noncomposite objects, which are objects with no composite instance variables. In this section, we discuss the various issues involved in handling the noncomposite instance variables of the model at the conceptual level. In Section 2.1, we discuss the basic assumptions of the model and in Section 2.2 we describe how noncomposite variables may be handled.

2.1 BASIC ASSUMPTIONS OF THE MODEL

The entities of classification in an object-oriented data model are the objects. That is, the instances, instance variables, methods, and classes are assigned security levels. The properties C1 to C4 discussed below are the basic security properties that are enforced:

C1. If o is an object (either an object-instance, class, instance variable, or method) then there is a security level L such that Level(o) = L.

C2. All basic objects (example, integer, string, boolean, real, etc.) are classified at system low.

C3. The security levels of the instances of a class dominate the security level of the class.

This property is meaningful because it makes no sense to classify a document at the Secret level while the document class that describes the structure of a document is at the Top Secret level. On the other hand, a Secret document class could have Secret and Top Secret document instances:

C4. The security level of a subclass must dominate the security level of its superclass.

This property is meaningful as it does not make sense to classify all documents as Secret and an English document to be Unclassified.

We assume that the following security policy is enforced—subjects (e.g., processes) and objects (e.g., classes, instances, instance variables, methods, composite links, etc.) are assigned security levels:

1. A subject has read access to an object if the subject's security level dominates that of the object.

2. A subject has write access to an object if the subject's security level is equal to that of the object.

3. A subject can execute a method if the subject's security level dominates the security level of the method and that of the object with which the method is associated.

4. A method executes at the level of the subject who initiated the execution.

5. During the execution of a method ml, if another method m2 has to be executed then m2 can execute only if the execution level of ml dominates the level of m2 and the object with which m2 is associated.

6. Reading and writing objects during method execution are governed by the properties 1. and 2.

2.2 NONCOMPOSITE INSTANCE VARIABLES

In this section, we describe some of the alternate security properties that may be enforced on the noncomposite instance variables (composite instance variables are discussed in Section 5). A similar argument can also be applied to handling methods. However, in this article we focus on structural aspects of an object-oriented data model, only, and not on the operational aspects. Therefore, we do not discuss methods in this article. Also, note that any reference to instance variables in this section implies noncomposite instance variables.

Two ways to assign security levels to instance variables are as follows:

C5. The security level of an instance variable of a class is equal to the security level of the class.

C5*. The security level of an instance variable of a class dominates the security level of the class.

If C5 is enforced, then it is assumed that the objects are single level. This is the assumption made in [Thur89a, Mille90] among others. If C5* is enforced, then it is assumed that an object is multilevel. This is the assumption made in [Kee89], among others. Note that we consider an object to be multilevel if its properties are classified at different security levels. We discuss each approach in the following two subsections. It should be noted that our main focus is on the representation of the real world entities at the con-
ceptual level. Therefore, we do not address the issues involved in the physical representation of the real world entities.

2.2.1 Single-level objects

If security property C5 is enforced, then the objects are assigned a single level. That is, instance variables have the same security level as that of the class with which they are associated. Therefore, if a document class is Unclassified, then its instance variables, say, title, author, publisher, and publication date are also Unclassified. Suppose a document also has a sponsor who funded its production and the fact that there is such a sponsor must be kept Secret. This means that the document has an additional instance variable that should be Secret. However, the security property C5 will not permit such an instance variable to be associated with a document. There are two solutions for this scenario. One is to create a different document class at the Secret level that has title, author, publisher, publication date, and is sponsored as its instance variables (Fig. 1(a); note that the Secret structures are darkened). Note that for every document instance of the Unclassified class there will be a document instance of the Secret document class. Both instances will have the same values for the attributes title, author, publisher, and publication date. The instances of the Secret document class will have the additional attribute of sponsor.

The second solution is to create a Secret subclass of the Unclassified document class (Fig. 1(b)). The Secret subclass inherits all the instance variables of document. It has sponsor as an additional instance variable. Note that for every document instance of the Unclassified superclass there will be a document instance of the Secret subclass. Both instances will have the same values for the attributes title, author, publisher, and publication date. The instance of the subclass will have the additional attribute of sponsor.

The instance variables of an object can be regarded as links emanating from the object. The values pointed to by the links are also objects. Although the instance variables of a class have the same security level as that of the class, it does not necessarily mean that an instance variable of an instance of a class must have the same security level as that of the class. This is because property C3 assumes that the security level of an instance dominates the security level of the class. Therefore, if the class is Unclassified and its instance is Secret, then the instance variables associated with this instance must also be Secret. Note also that it does not make sense to classify an instance variable of this instance at a Top Secret level because a Secret user knows that there is such an instance variable. Note also that the level of the object pointed to by

the instance variable link (i.e., the value of the instance variable) must be dominated by the level of the link. Therefore, we have the following security properties on instance variables of objects:

C61. The level of the instance variable of an object must be the same as that of the object.

C62. The level of the value of an instance variable must be dominated by the level of the instance variable.

C63. If the instance variable c of an object is a complex instance variable, the security level of c is L, and if o1, o2, ...on are the objects that form the value of the instance variable c, then the security levels of o1, o2, ...on, are dominated by L.

Figure 2(a) illustrates two instances of an Unclassified document class. Note that the Secret document's title and author instance variable values are Secret. The remaining values are Unclassified. Figure 2(b) shows how complex instance variables may be modeled.

Next let us examine how polyninstantiation could be handled (note that by polyninstantiation we mean users at different levels having different views of the same entity—for a discussion on polyninstantiation in relational systems we refer to [Stach90]). Consider the Unclassified document shown in Figure 3(a). This document is Unclassified. It has instance variables title, author, publisher, and publication date. The publisher instance variable link points to NIL because it assumes that an Unclassified user does not know the publisher's name. Let us assume that a Secret user knows of the publisher. Also, the Secret subjects know that the real author of the document is James and not John. There are two ways to handle polyninstantiation. In the first approach, a new Secret document instance is created with attributes as shown in Figure 3(b). Note that in addition to the attributes specified, an attribute such as document-ID will also be necessary to relate the two objects. In the second approach, the polyninstantiated values are attached to the Unclassified document instance as shown in Figure 3(c).

One of the advantages of enforcing the security property C5 is that single-level objects can be mapped into single-level seg-
ments or files in a straightforward manner. As a result, traditional security policies (such as the Bell and LaPadula security policy [Bell75]) can be used to control access to the single-level objects. This way, systems with higher levels of assurance can be developed (for a discussion on assurance we refer to [Trust85]). A disadvantage with this approach is that the conceptual representation may not model the real world accurately. This is because in the real world multilevel objects do exist. That is, there could be individuals whose properties are classified at different security levels. A user’s view of the database should usually model the real world closely.

2.2.2 Multilevel objects
If we enforce the security property C5* instead of C5, then the objects could be multilevel. That is, the instance variables of the object could have different security levels. Note that in this approach the security level of the instance variables of a class could dominate the security level of the class. Therefore, the document shown in Figure 3 could be represented by the structure in Figure 4.

The instances of UDOC could be multilevel objects. For example, for each Unclassified document instance the instance variables title, author, publisher, and publication date are Unclassified. The instance variable sponsor is Secret. Also, the security level of the value of an instance variable must dominate the security level of the instance variable. That is, the following security properties are enforced:

C6*1. The level of the instance variable of an object dominates the level of the object.

C6*2. The level of the value of an instance variable must be dominated by the level of the instance variable.

C63. If the instance variable c of an object is a complex instance variable, the security level of c is L, and if a1, a2, ..., an are the objects that form the value of the instance variable c, then the security levels of a1, a2, ..., an, are dominated by L.

Figure 5 illustrates Unclassified and Secret documents that belong to the Unclassified document class of Figure 4. Note that by an Unclassified document we mean that the structure that represents the document is Unclassified. It could, however, have Secret components. Polyinstantiation could be handled either by creating a new object at a different security level or by polyinstantiating the value of an instance variable (see the discussion associated with Fig. 3).

An advantage of enforcing the security property C5* is that it models the real world closely. A disadvantage is that multilevel objects may have to be decomposed into single-level objects that could then be stored in single-level segments or files to provide higher levels of assurance. With such a decomposition, the performance advantages of storing related objects in clusters could be lost. The issues involved in providing performance as well as assurance need to be investigated further.

3. COMPOSITE OBJECTS IN MULTILEVEL DATABASES
In this section, we discuss the various issues involved in supporting composite objects in a multilevel environment. In Section 3.1, the security properties of composite objects are discussed. Representations of composite objects are discussed in Section 3.2. In Section 3.3, some theoretical properties of composite objects are discussed. Composite links connecting a composite object to its components are described in Section 3.3. In particular, the grouping of composite links and its formal semantics are described.

3.1 SECURITY PROPERTIES OF COMPOSITE INSTANCE VARIABLES
A composite object has a composite instance variable. Like non-composite instance variables, composite instance variables are also assigned security levels. Also, there are two ways to assign security levels to composite instance variables. They are:

C7. The security level of the composite instance variable is the security level of the class with which it is associated.
Multilevel secure O-O data model

C7*. The security level of a composite instance variable dominates the security level of the class with which it is associated.1

Figure 6 illustrates an example of security property C7 being enforced. Here, the composite instance variable (which describes the components of an object) of a class is assumed to be Secret. The noncomposite instance variables are Unclassified. The solution is to create an Unclassified class with the noncomposite instance variables and either create a new Secret class with the noncomposite as well as the composite instance variables (Fig. 6(a)) or create a new Secret subclass of theUnclassified class with the composite instance variable (Fig. 6(b)). Note that for every instance of the Unclassified class there is an instance of the Secret class. The Secret instance has the same values for the noncomposite instance variables of the Unclassified instance. In addition, the Secret instance will have a value for the composite instance variable.

Figure 7 illustrates the same example in which the security property C7* is enforced. That is, only one Unclassified class is created. Its composite instance variable is classified at the Secret level. The noncomposite instance variables are Unclassified. Note that for each Unclassified instance of this class the noncomposite instance variables are Unclassified. The composite instance variable is Secret. For a Secret instance of this class, all instance variables (noncomposite and composite) are Secret.

3.2 REPRESENTATION OF COMPOSITE OBJECTS

3.2.1 Alternatives

In this section, we discuss the alternative representations of composite objects. These representations are not affected by the security property enforced on the composite instance variables (i.e., either C7 or C7*). However, the following security property, which describes the relationship between the composite instance variable and the composite links, is enforced:

C8. The security level of a composite instance variable of an object is dominated by the security level of the composite links

1Note: compare C7 and C7* with the respective properties C5 and C5*.
Figure 11. Sharing among polyinstantiated composite objects.

Figure 12. Two documents at different security levels.

Figure 13. Granularity of polyinstantiated object.

Section 10, the Secret version of the document is represented (note that the darkened structures represent the entities classified at the Secret level).

An alternate approach to representing the composite document of Figure 8 is shown in Figure 11. In this alternate approach, the security level of an object dominates the security level of all of its components. That is, the Secret version of the Unclassified document shares the Unclassified sections with the Unclassified version of the document. The Secret version of the document consists of some additional Secret sections. Note that the polyinstantiated version link between the Unclassified document instance and the Secret document instance can be implemented in various ways. The important point here is that there is some way for a Secret user to know that the Secret document instance is actually a polyinstantiated version of the Unclassified document instance.

Although the complete duplication of the document at different security levels is avoided in the representation of Figure 11, a new document instance at the Secret level still has to be created. A third alternative is to represent the document exactly as it is represented in the real world (see Fig. 8). That is, an Unclassified document could have Secret as well as Unclassified sections. The Secret sections are erased from the view of the Unclassified users. The Secret users can go elsewhere and obtain the Secret sections only. This way, it is not necessary to create a new document instance at a different security level.

### 3.2.2 Object sharing

Object sharing is an important requirement for hypermedia and CAD/CAM applications. For example, it may be necessary for various sections and paragraphs to be shared between different documents. In a multilevel environment, it is possible for different documents to be at different security levels but share sections and paragraphs. This scenario is illustrated in Figure 12. Object sharing is addressed in more detail in the discussion on composite links given later.

### 3.2.3 Polyinstantiation

As described earlier, it is possible for two users at different security levels to have different views of the same entity. For example, it is possible for an Unclassified section of a document to be just a cover story to a more sensitive version. Polyinstantiation can occur at different stages. At one extreme, one can have the whole document polyinstantiated. At the other extreme, one has a word or a figure polyinstantiated. Figure 13 shows two ways of polyinstantiating sections of a document. In the first approach, the Unclassified section is polyinstantiated at the Secret level (Fig. 13(a)). In the second approach, the cover story is compared with the actual version. If possible, the actual version is decomposed into paragraphs. The sensitive paragraphs are classified at the Secret level. The remaining paragraphs are Unclassified. If an Unclassified paragraph contains false information, then it can be polyinstantiated at the Secret level (Fig. 13(b)).

To reduce the amount of polyinstantiated objects, the objects could be decomposed into smaller units, as much as possible, and the smaller units could be polyinstantiated if necessary. It should be noted that polyinstantiation is still a research issue in multilevel database systems. The issues involved in handling polyinstantiation in object-oriented systems are discussed in Section 4 where we regard polyinstantiation as a special form of versioning.

### 3.3 COMPOSITE LINKS

A composite link is a link that connects a composite object with one of its components. A composite link is also assigned a security level. Figure 14 illustrates possible composite links from a composite object O to one of its components M. We assume that the links are bidirectional. That is, for each link P, there is link P' in the reverse direction. The following security property is enforced:

C9. Let P be a composite link whose reverse link is P'. Then Level(P) = Level(P').

Some of the cases shown in Figure 14 are not meaningful. For example, it does not make sense to form an Unclassified link between a Secret composite object and its Secret component. Further, supporting all the cases of Figure 14 will make certain types of links (to be discussed below) difficult to implement. Therefore, we impose the following security property on the composite objects:

C10. Composite link property
If \( P \) is a composite link between a composite object \( O \) and its component \( M \), then Level\((P) \geq \text{1.u.b.}\) \([\text{Level}\((O)\), \text{Level}\((M)\)]\).

We also assume that:

\[
\begin{align*}
\text{Level}(P) &= \text{Level}(\text{Exist}(P)) \\
\text{Level}(O) &= \text{Level}(\text{Exist}(O)) \\
\text{Level}(M) &= \text{Level}(\text{Exist}(M))
\end{align*}
\]

where \( \text{Level}(\text{Exist}(e)) \) is the security level of the existence of an entity \( e \).

Enforcing the composite link property will permit only the cases illustrated in Figure 14(a) – (e).

In some cases, it may be necessary for composite objects not to share their components. In other cases, it may be necessary for the existence of a component object to be dependent on the existence of the composite object. These considerations have led object-oriented database researchers to define various types of composite links [Kim87]. We review these definitions and discuss how they may be affected due to multilevel security.

Various types of composite links have been studied in the literature [Kim87, Kim88]. A composite link from object \( O \) to component \( M \) may be either exclusive or shared. If it is an exclusive link, then it is not possible for another composite object \( O' \) to have any link to \( M \). If it is shared, then it is possible for other composite objects to have shared links to \( M \).

The links shown in Figures 14(d) and 14(e) cannot be exclusive or shared. Suppose these links are exclusive. An Unclassified user can see the object \( M \), but he will not know that \( M \) is a component of a composite object. Therefore, he could add an exclusive or shared composite link \( P' \) from another Unclassified object \( O \) to \( M \). This second link violates the exclusive link property. This scenario is shown in Figure 15(a). If the link \( P \) is shared, and if the link \( P' \) is exclusive, the exclusive property link is violated. This scenario is illustrated in Figure 15(b).

It does not make much sense to make exclusive the links shown in Figures 14(d) or 14(e). This is because the links shown in Figures 14(d) and 14(e) can only be specified by a Secret user. If this user really wants the link to be exclusive, then he could create a Secret object replicating \( M \) and impose an exclusive link from \( O \) to this new object. However, if the links shown in Figures 14(d) and 14(c) are not allowed to be shared then it will make the model overly restrictive. A possible solution to overcome this problem is as follows. Suppose an Unclassified user wants to define an exclusive link from \( O \) to \( M \). He can do so only if \( M \) does not already exist. In this case, he can create \( M \) and impose an exclusive link from \( O \) to \( M \). Now, no other users can have any composite links from any object to \( M \). If \( M \) already exists, there is always a possibility of a higher-level object to have a composite shared link to \( M \). Therefore, there cannot be an exclusive link from \( O \) to \( M \). If the Unclassified user wants to impose an exclusive link from \( O \) to \( M \), then he will have to replicate \( M \) and specify the link.

A composite link from an object \( O \) to its component \( M \) may be either dependent or independent. If it is dependent, then \( M \) cannot exist without \( O \) provided there is no other object \( O' \) that...
Multilevel secure O-O data model

has a link to $M$. If the link from $O$ to its component $M$ is independent, then $M$ can exist without $O$.

Note again that the links shown in Figures 14(d) and 14(e) cannot be dependent links. For example, consider the link in Figure 14(e). Suppose this link is dependent. Also assume that no other object has a link to $M$. Since the object $O$ is Unclassified, an Unclassified user can delete this object. Since he does not know of the existence of $P$, the object $M$ is not deleted. A Secret user cannot delete $M$ because it is at a lower level. A different problem occurs if the link $P$ in Figure 14(d) is made dependent. If, for some reason, a Secret user wants to delete $O$, he cannot do so because of the dependent link from $O$ to $M$. This is because he cannot delete the object $M$ either. He will have to wait until $M$ gets deleted first. Although this situation is not a violation of the dependent link property, it could cause objects that are not in use to consume space. Note that in the link shown in Figure 14(e), the dependent link property can still be enforced. For example, if an Unclassified user deleted $O$, since he does not know of the existence of the link and also since $M$ is Secret, he will not delete $M$. However, a consistency checker which runs at the Secret level can detect this problem and delete $M$ to preserve the dependent link property.

4. VERSIONING IN A MULTILEVEL ENVIRONMENT

We first review the model of versions of objects in object-oriented data models such as ORION [Baner78], and then extend the concepts to a multilevel environment. The discussion will be limited to noncomposite objects only. For a discussion on versioning for composite objects in a multilevel environment, we refer to [Thur90b].

A class is defined to be versionable if versions of the instances of the classes can be created. The versions of an instance provide a hierarchy of versions called the version derivation hierarchy. Information about the version derivation hierarchy of an object $o$ is maintained in an object called the generic instance of $o$.

If the noncomposite instance variable link of an object $o'$ points to a version instance of another object $o$, then $o'$ is statically bound to $o$. If the noncomposite instance variable link of an object $o'$ points to the generic instance of another object $o$, then $o'$ is dynamically bound to $o$. The system could assign a default version instance of $o$ to be assigned to this link (see Fig. 16).

Let an object $o'$ have an instance variable link to another object $o$. Suppose a version $v$ of $o'$ is obtained. Then the model should specify as to whether the instance variable link of $v$ should also point to $o$, or the link is assigned to some other value (e.g., NIL, a generic instance of $o$, or another version of $o$).

In a multilevel environment, we identify three types of versions: historical versions, alternate versions, and polyninstantiated versions. Historical versions are due to the evolution of objects over time. Alternate versions store alternate representations of the same entity. Both the historical versions and alternate versions can be handled within as well as across security levels. Polyninstantiated versions are produced when users at different security levels have different views of the same entity. They can only be handled across security levels.

Figure 16 illustrates a version derivation hierarchy of an Unclassified object. Here, versions are created within and across security levels. The generic instance has information on the version derivation hierarchy. Assuming that there are only two security levels, Unclassified and Secret, the generic instance stores Unclassified information of the hierarchy at the Unclassified level and Secret information of the hierarchy at the Secret level.

In this figure, the generic instance of object $O$ has an Unclassified version instance $V_1$. $V_2$ is a polyninstantiated version of $V_1$ at the Secret level. $V_3$, $V_5$, and $V_7$ are historical versions of $V_1$, $V_2$, and $V_3$, respectively. $V_4$ and $V_6$ are alternate versions of $V_3$ and $V_4$, respectively. $V_8$ could be either a historical or a polyninstantiated version of $V_4$ at the Secret level.

The following are possible security properties for versions of noncomposite objects:

C11. Let $v$ be a version instance of the object $o$. Then Level($v$) $\geq$ Level($o$).

C12. Let $g$ be the generic instance of an object $o$. Then Level($g$) = Level($o$).

C13. Let $o'$ have an instance variable link to version $v$ of object $o$. Then Level($o'$) $\geq$ Level($v$).

C14. Let $o'$ have an instance variable link to generic instance $g$ of object $o$. Then Level($o'$) $\geq$ Level($g$).

C15. Let $o'$ have an instance variable link to an object $o$. Let $v'$ be a version instance of $o'$. Then the instance variable link of $v'$ points to one of the following:

1. NIL,
2. a, provided Level($v'$) $\geq$ Level($o$),
3. generic instance $g$ of $o$, provided Level($v'$) $\geq$ Level($g$), and
4. a version instance $v$ of $o$, provided Level($v'$) $\geq$ Level($o$).

5. CONCLUSION

In this article, we reviewed the developments of security in object-oriented systems and discussed the alternate ways that noncomposite objects could be handled in a multilevel environment. We then focused on the issues that must be handled in order to pro-
vide support for composite objects in a multilevel environment. In particular, the security properties of composite objects, representation of composite objects, and composite links were described. We then discussed issues on version management for a multilevel secure object-oriented database system.

Future research in this area will include the development of a multilevel secure object-oriented data model to support non-composite objects, composite objects, object sharing, and versioning. The issues discussed in this paper will aid the development of such a model. Another important issue that has not been addressed in this paper is a model for concurrency control. Locking as a concurrency control mechanism for object-oriented database systems was proposed in [Kim88]. However, it is well known that the locking technique causes a covert channel. For example, two users at the Secret and Unclassified levels could request a read lock and a write lock, respectively, to an Unclassified data object. If the Secret user already has obtained the read lock, then the write lock will not be given to the Unclassified user. If the Secret user does not have a read lock then the write lock is given to the Unclassified user. If the Secret and Unclassified users collide, then they can synchronize a series of requests to the Unclassified data object in such a way that from the pattern observed by the granting/denial of the requests to the Unclassified user, information can be covertly passed by the Secret user to the Unclassified user. It has also been argued that the traditional approaches to concurrency control could cause a performance bottleneck. This is because the transactions in object-oriented applications are of very long durations [Kort88]. Therefore, novel concurrency control techniques need to be developed. A preliminary investigation on concurrency control in multilevel object-oriented systems is reported in [Thura90b].

Once a data model has been developed, the next step will be to focus on the security policy and implementation issues. The objects could be multilevel at the conceptual stage and could be decomposed and stored physically in single-level segments (or files) to obtain higher levels of assurance. However, such an approach loses the advantages of storing composite objects in clusters (which has been strongly recommended for operation in a nonmultilevel environment). Storing a composite object together with its components in clusters greatly enhances the performance of database systems [Kim87]. Therefore, it is important to conduct research on the issues involved in enhancing the performance of the system, but at the same time provide higher levels of assurance.

Finally, the design of a multilevel secure object-oriented database system should be based on the data model and security policy that was developed. Such a design should provide the support for query processing, schema management, dynamic schema evolution, update processing, and transaction management and should handle integrity as well as security constraints. Many of these functions are still research topics in object-oriented database systems. Therefore, much remains to be done before multilevel object-oriented database management systems can be developed.

ACKNOWLEDGMENT

The authors gratefully acknowledge the Rome Air Development Center (RADC) for sponsoring this work under contract F19628-89-C-0001. We thank Joe Giordano of RADC for his support and encouragement throughout the project. We thank John Faust of RADC for monitoring the project. We thank Maureen Chehely for her comments.

REFERENCES


