MANDATORY SECURITY IN OBJECT-ORIENTED DATABASE SYSTEMS

M.B.Thuraisingham
The MITRE Corporation, Bedford, MA, 01730

Abstract

A multilevel secure object-oriented data model (using the ORION data model) is proposed for which mandatory security issues in the context of a database system is discussed. In particular the following issues are dealt with: (1) the security policy for the system, (2) handling polyinstantiation, and (3) handling the inference problem.

A set of security properties that has been established in this paper is more complete than those that have been proposed previously. Finally we describe how certain security constraints are handled by our model.

1. Introduction

In an object-oriented system, any entity such as an integer, automobile, person or city is modelled as an object. This power of representation enables new generation applications such as CAD/CAM, Image Processing, Artificial Intelligence and Process Control to be developed (see for example [KONA89]). For many of these applications it is becoming very important that they operate securely, while for some others it is also necessary to incorporate multilevel security. This is to overcome any malicious corruption of data as well as prohibit unauthorized access to and use of classified data.

At present no standard data model for object-oriented systems has been proposed. Until one becomes available, the choice of an object-oriented data model is an arbitrary one that is determined by the application. Thus the adoption of a standard secure object-oriented model would have to be delayed until a consensus is formed. That is, for the present security features have to be incorporated into an object-oriented system only on a model by model basis.

In this paper we consider only the ORION object-oriented data model which was developed at MCC [BANE87] and describe how security properties might be incorporated into such a model. We will call this model SORION. We then formulate a security policy for a database system based on this model and discuss other mandatory security issues such as polyinstantiation and handling the inference problem. Finally we describe how mandatory security constraints, which are rules assigning security levels to data, may be handled.

The organization of this paper is as follows: In section 2 we will describe concepts in multilevel secure database management systems (MLS/DBMS). In section 3 we will give an informal overview of the ORION data model. In section 4 we will describe SORION (Secure **ORION**), a multilevel secure object-oriented data model. This model extends ORION by incorporating security properties. In section 5 we will discuss the mandatory security issues in an object-oriented database system based on SORION. Handling security constraints are addressed in section 6. Finally the paper is concluded in section 7.

2. A Brief Account of Multilevel Secure Database Systems

In a multilevel secure database management system (MLS/DBMS) users cleared to different security levels access and share a database consisting of data at different sensitivity levels. The sensitivity levels (which we will also refer to as security levels) may be assigned to the data depending on content, context, aggregation and time. An effective security policy for MLS/DBMS should ensure that users only acquire the information to which they are authorized. The earliest of security policies, the Bell and LaPadula security model [BELL75], is stated below.

(i) Subjects are the active entities (such as processes) and objects are the passive entities (such as files)
(ii) Subjects and objects are assigned security levels. The set of security levels form a partially ordered lattice.(e.g. Unclassified < Confidential < Secret < TopSecret).

(iii) simple security property: A subject has read access to an object if the subject's security level dominates the security level of the object
(iv) *-property: A subject has write access to an object if the subject's security level is dominated by the security level of the object.
This security model is not sufficient to ensure multilevel security in a DBMS as users can pose multiple queries and infer unauthorized information from the legitimate responses that they receive. Despite its shortcomings, extensions to the Bell and LaPadula security model have since been proposed for MLS/DBMSs [DWYE88].

The relational data model has dominated much of the work on MLS/DBMSs (See for example DWYE87, STAC89a, STAC89b). As a result of such work, multilevel secure relational database systems have been developed not only as prototypes but also as products [ROUG87]. In recent times security issues have also been investigated in other systems such as object-oriented systems [KEEF88, THUR89a], functional database systems [THUR88a], entity relationship systems [GAJN88] and knowledge based systems [THUR89b] among others. A detailed description of the recent development in database security is given in [THUR89c].

An attempt was previously made to incorporate security features into the ORION object model from which a preliminary set of security properties was developed [LUNT88]. However, this work did not deal with many of the features essential to an object-oriented model such as aggregate objects and set constructs; neither did it address inheritance mechanisms adequately. In the present paper these issues as well as others (such as mandatory security policy, polyinstantiation and the inference problem) are addressed. Consequently the set of security properties developed here are more complete and could lead to a secure model that is easier to implement.

3. Overview of ORION

As stated in [BANE87], all conceptual entities in ORION are modeled as objects. A group of objects with similar properties form a class that is also an object. A class could be a system-defined class, such as a class of integers or strings, or it could be a user-defined class such as a class of documents or employees. The objects of a class are called its instances. Associated with each class is a set of instance variables that describe the state of the instances of the a class. Object-ID, which uniquely identifies an object of the class is also an instance variable. For example, a class of employees could have Object-Id, social security number, name, salary and department as its instance variables. An instance variable is also an object.

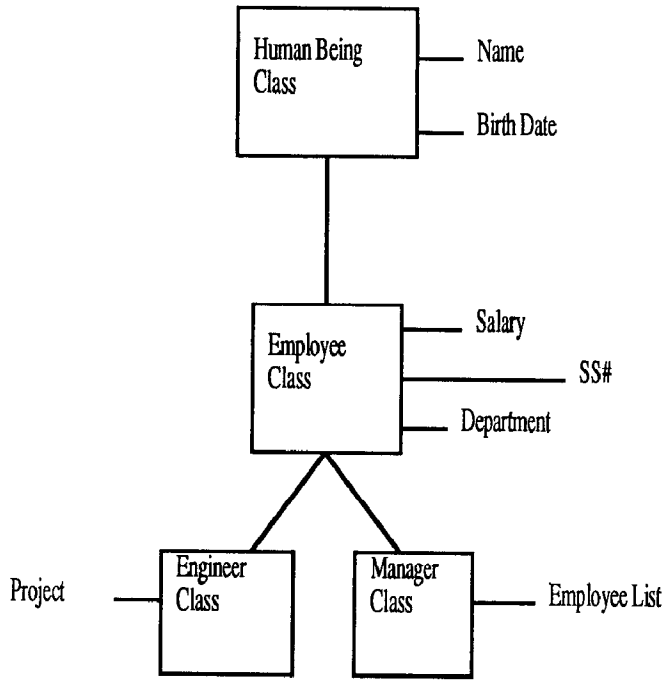

Figure 1a IS-A Hierarchy

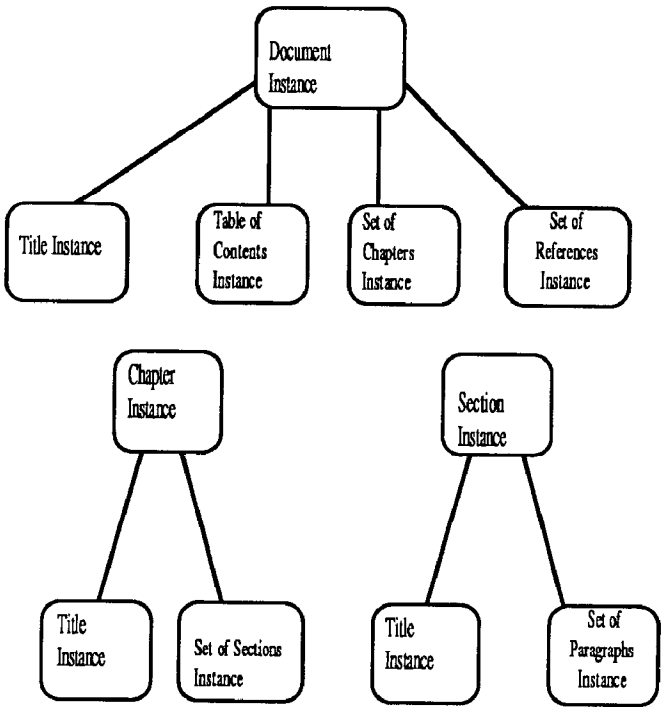

Figure 1b IS-PART-OF Hierarchy

A class also has methods associated with it that encapsulate the behavior of the objects associated with the class. For example, a message may be sent to the class of employees to retrieve the salary of a particular employee. The ORION data model also supports the set construct. For

example the class EMPS could be defined so that it consists of sets of employees as its instances.

Two types of class hierarchies may be formed. One is the IS-A hierarchy where a class has subclasses associated with it. The subclass inherits all of the instance variables and methods associated with its superclass. For example, the class of all human beings has the class of all employees as its subclass. This subclass will inherit all the instance variables from its superclass plus have additional instance variables such as social security number, salary and department. If a class has more than one superclass and two or more superclasses have the same instance variable names, then the value of the instance variable inherited by the class depends on some apriori rule enforced. This is called multiple inheritance. The IS-A hierarchy described here is illustrated in Figure 1a.

The second class hierarchy is the IS-PART-OF hierarchy. Here an object of a class is considered to be the aggregation of a set of objects, each of which belongs to some class. Such an aggregate object is also called a composite object. For example, a document object, which is a member of the document class, consists of a title, table of contents, set of chapters and references. A chapter object that belongs to the chapter class has a title and set of sections as its components. A section object that belongs to the section class has a title and set of paragraphs as its components. The IS-PART-OF hierarchy described here is illustrated in Figure 1b.

4. SORION - A Multilevel Secure Object-oriented Data Model

SORION has evolved from ORION by incorporating security levels for all entities and enforcing security properties that must be satisfied. The issues are discussed in this section.. We discuss multilevel objects, multilevel classes, the rules enforced on classes and subclasses, multilevel methods, multiple inheritance and aggregate classes and objects. We also discuss security properties of relationships objects, a construct introduced to model multimedia systems [WOEL86].

Objects

The entities of classification are all kinds of objects. These include the objects, the classes, the methods, and the instance variables. An object could be a basic object or a complex object. A basic object could be either an integer, boolean, real or string. A complex object is any object which is not basic. This also includes a set object. The following security properties hold for the objects.

P1: If o is a object, then there is a level L such that Level(o) = L

P2: If o is a basic object, then Level(o) = system-low (this is usually the Unclassified level in the military environment)

P3: The security level of the name must dominate the security level of its value.
That is, salary object could be Secret while its value of 20K is Unclassified. Note that the value 20K is a basic object and therefore its level is system-low.

P4: If o is a set object {a1,a2,.....an}, then Level(o) >= l.u.b.(Level(a1), Level(a2), ...Level(an))
That is, if o is a set of employees {John, Mary , James} and the security levels of John, Mary and James are Unclassified, Secret and TopSecret respectively, then the security level of o must be at least TopSecret.

P5: The security levels of the instance variables of an object are the same as that of the object.
For example, let an employee object be classified at the Secret level. Let the instance variables of Employee object be Name, Salary and SS#. Then all three instance variables are also Secret. However, the values of these instance variables such as "John", 20K and "000-00-000" could be assigned security levels less than the Secret level.

Classes

A class has two components associated with it; a structure and a set of methods. Structure of a class is described by the instance variables defined on the class. Methods are the operations that are performed on the instances of the class. We first define the security properties associated with the structure portion of a class and then describe the security properties associated with the methods.

Following are the security properties for class:

P6: If C is a class, then there is a security level L such that Level(C) = L

P7: The security levels of the instances of a class must dominate the security level of the class.

P8: Anyone who can read the name of a class should also be able to read the names of instance variables of class C.
That is, the security levels of the instance variables are that of the class. However, if a user cannot read an instance of a class, then this user cannot read the instance variables of this instance also.
For example, let EMP be a class with instance variables

Name, Salary and SS#. Let EMP be classified at the Secret level. Then all three instance variables will also be classified at the Secret level. Suppose EMP has an instance say o at the TopSecret level. Then the name, salary and SS# for o will be classified at the TopSecret level. As mentioned earlier, the values themselves need not be TopSecret.We enforce rule P8 in order to avoid the multilevel update problem. We will address this point later in section 6.

P9: The security level of a subclass must dominate the security level of the superclass.
For example, if SENIOR-EMP is a subclass of EMP, and if EMP is Secret, then SENIOR-EMP must be at least Secret. Stated more formally, for every security level L, the model of a superclass must contain the model of a subclass,where a model of a class at a security level L is the set of all instances of that class whose security levels are dominated by L. A model of a class C at level L is denoted by M(C,L).

P10: The instance variables of a subclass (whether inherited or defined) have the same security level as that of the subclass. For example, SENIOR-EMP will inherit the name instance variable form EMP. If EMP is Secret and SENIOR-EMP is TopSecret, then name in SENIOR-EMP is still TopSecret although name in EMP is Secret.

Methods

The domain of methods could be multiple classes. The range of a method is a class. A method can be regarded as a function object. The following are the security properties of methods.

P11: If m is a method, then there is a security level L such that Level(m) = L

P12: If a method m is defined on C1 X C2 X .....Cn and its range is C, then
Level(m) >= l.u.b.(C1, C2, ...Cn, C)

P13: A model of a method m (C1 x C2 x .....Cn -> C) at a security level L (denoted M(m,L)) is the set of all partial functions from M(C1,L) x M(C2,L) x .....M(Cn,L) into M(C,L).
A method m1 is a submethod of a method m2, if M(m1,L) is a subset of M(m2,L) for all security levels L.

P14: If C1 is subclass of C2 and m2 is a method of C2, then there is a method m1 of C1 with the same name as m2 such that m1 is a submethod of m2.This is the method inheritance property. The following condition also holds:
Level(m1) = l.u.b.(Level(m2), Level(C1))

Multiple Inheritance

As described earlier, a class may inherit the methods and instance variables from one or more superclasses. In the case of conflict, some apriori rule should determine how to resolve it. In the case of SORION, the following properties will determine how conflicts should be resolved.

P15: Let C be a subclass of C1, C2, ...Cn. Let the instance variable V be associated with C1, C2.....Cn. C will inherit the instance variable associated with class Cj (1 <= j <= n) such that Level(Cj) dominates the levels of the remaining classes. If there are more than one such Cj, then some apriori rule should be enforced to resolve the conflict.

P16: Let C be a subclass of C1, C2, ...Cn. Let the method m be associated with C1, C2.....Cn. C will inherit the method associated with class Cj (1 <= j <= n) such that Level(Cj) dominates the levels of the remaining classes. If there are more than one such Cj, then some apriori rule should be enforced to resolve the conflict.

Aggregate Classes and Objects

This is the IS-PART-OF hierarchy which could be defined on classes as well as class instances. In the case of class aggregation, a class C may be an aggregate of classes C1, C2, ...Cn.. Then each instance of C is an aggregate of the instances of C1, C2, ...Cn respectively. For example consider the automobile class. This class could be an aggregate of the classes Engine, Chassis and Wheels. The instances of Engine class are the various types of engines, the instances of Chassis are the various types of car structures and the instances of Wheels are the various sets of four wheels. Any car is composed of an engine, a set of wheels and a chassis. The following security property holds:

P17: Level(C) >= l.u.b.(Level(C1), level(C2), ...Level(Cn))

In the case of aggregate objects, an object o may be composed of objects o1, o2, ...on. This does not mean that if o belongs to class C and o1, o2...on belong to classes C1, C2, ...Cn respectively, then C is an aggregate of the classes C1, C2, ...Cn. An example of aggregate object is a book object which consists of a title, introduction, chapter 1, chapter 2 and conclusion. Another book object may consist of a title, introduction, body and conclusion. Both books will belong to the book class; but they have different structures. The following security property holds.

P18: Level(o) >= l.u.b. (Level(o1), Level(o2), ...Level(on))

Properties P17 and P18 imply the following property:

P19: If (i) o is an instance of C
(ii) o1, o2, ...on are instances of C1, C2, ...Cn respectively
(iii)C is an aggregate of c1, C2, ...Cn and
(iv) o is an aggregate of o1, o2,.....on
then Level(o) >= l.u.b.(Level(o1), Level(o2, ...Level(on))

Relationship Objects

Relationship objects are necessary to represent multimedia information. For example, there could be a link from a line instance to the voice instance associated with the line. The two objects (line and voice) are connected by a relationship object which is basically the link from the line to the voice. The following security property holds for relationship objects.

P20: Let R be a relationship object which describes a relationship between two objects O1 and O2. Then
Level(R) >= l.u.b.(Level(O1), Level(O2))

5. Mandatory Security Issues in a SORION-based Object-Oriented Database System

In this section we will describe the mandatory security issues in an object-oriented database system which is based on SORION. In section 5.1. we will describe our mandatory security policy. Polyinstantiation issues will be describe in section 5.2. Finally in section 5.3 we will describe how the inference problem in database security could be handled.

5.1 Security Policy

The security policy for an object-oriented database system based on SORION consists of the following properties:

(i) Subjects and entities (we use the term entity instead of an object as it is usually stated in security policies in order to not confuse between the object in security policies and object in an object-oriented system) are assigned security levels.
(ii) A subject has read access to any entity if the subject's security level dominates the security level of the entity.
(iii) A subject has write access to an entity, if the subject's security level equals the security level of the entity.
(iv) A subject can execute a method if the subject's security level dominates both the security level of the method and the type on which the method is defined.
(v) A methods executes at the security level of the subject who initiated the execution.
(vi)During the execution of a method m1, if another method m2, has to be executed, then m2 can execute only if the execution level of m1 dominates both he security level of m2 and the security level of the type on which m2 is defined.
(vii) If a new object has to be created as a result of executing a method, the object is created at the security level of the subject who initiated the execution of the method.

Property (ii) is the simple property specified in the Bell and LaPadula security policy. Property (iii) is different from the *-property because writeup is not permitted (this is because it does not seem natural for a subject to write some data and not be able to read it later). The remaining properties are enforced due to method execution.

5.2 Polyinstantiation

Polyinstantiation generally occurs when two subjects at different security levels have different views of a single entity in the real world. In an object-oriented world, the different views could relate to different object values, different class structures, different class methods and different method definitions. We will describe each type of polyinstantiation below.

Object Value polyinstantiation: an Unclassified user views the object o as (John, 20K, 333) while a Secret subject views o as (John, 30K, 333).
Class Structure polyinstantiation: an Unclassified subject views the EMP class as consisting of the instance variables (name, SS#) while a Secret subject views EMP as consisting of (name, SS#, salary).
Class method polyinstantiation: an Unclassified user views EMP as having the methods get-name, change-name while the Secret subject views EMP as having methods get-name, change-name, get-salary, change-salary.
Method polyinstantiation: an Unclassified user views a method update-salary to have one parameter which is the amount by which the salary should be increased. A Secret user views this method as having two parameters; one is the amount and the other is the new salary value which is returned to the user.

Polyinstantiation is still a major research issue even in multilevel relational database systems. We will briefly describe possible scenarios for object polyinstantiation and give a possible solution.

Polyinstantiation occurs when
(i)) an Unclassified subject has created an object, say, o1 and a Secret subject creates a second object, say, o2 to represent the same entity and the Secret subject gives a different value or structure to the object created.
(ii) a Secret subject has created an object o1. The Unclassified subject is unaware of the existence of o1 and it creates another object to represent the same entity in the real world.The structure or value of the obiect

created by the Unclassified subject may be different from those of o1.
(iii) an Unclassified subject has created an object say o1 and a Secret subject uses the name of o1 to represent a different entity in the real world.
(iv) a Secret subject has created an object say o1 and an Unclassified subject uses the name o1(which we assume is an Unclassified name) to represent a different entity in the real world.

A possible solution to handle the various types of polyinstantiations could be the following:

(1) A Secret subject requests to use the same name that is already used for an Unclassified object only when it wants to polyinstantiate the Unclassified object. Otherwise a different name is used.

(2) When a Secret subject creates an object (which is not a polyinstantiated object) then the Secret subject should use a Secret name for that object (that is, we assume that the identifiers used for objects are also assigned security levels).

(3) If an Unclassified subject wants to create an object say o1 to represent the same entity which is already represented by a Secret object say o2, then the Unclassified subject will use an Unclassified name for o1. By 2), this will be different from the Secret name used by o2. However, with this approach there is no way to determine that o2 is a polyinstantiated version of o1 (unless we introduce the notion of primary key of an object which is not part of an object model).

We can justify (3) by taking Reiter's Closed World Assumption (CWA) [REIT78] into consideration. CWA states that information is represented in the database if and only if it is true in the real world. Therefore for an entity to be represented by some Secret object and not by an Unclassified object means that the entity which exists in the Secret world does not exist in the Unclassified world. For the entity to be brought into the Unclassified world it has to be downgraded (by some trusted subject). Then the Secret object which represents the entity must be deleted as the entity is now in the Unclassified world. An Unclassified object is created to represent this entity. However, this same entity can have different values or structures in the Secret world. Then a Secret object can be created later to represent the same entity with the same name as that of the Unclassified object.

With the solution that we have proposed we do not have to handle the case where two subjects at different security levels request the same identifier for two different objects which represent two different entities.

5.3 Inference problem

Security violations via inference occurs when users pose multiple queries and acquire unauthorized information [THUR87, MORG87]. A solution to handling the inference problem in relational systems is to augment a relational DBMS with a logic-based inference engine and a knowledge base. The inference engine will detect security violations via inference when processing queries [THUR88c, KEEF89]. A similar inference controller can be built for object-oriented systems also [THUR89d]. Two approaches to implementing such an inference controller are as follows: In the first approach, the database as well as the security constraints are expressed in a logic programming language with support for representing and manipulating objects. An example of such a language is object-prolog [ZANI84]. In the second approach, an object-oriented database system is augmented with an inference engine and a rule base. The inference engine is based on an extension to first order logic. The queries are modified first by the inference engine before the object-oriented DBMS processes them. The techniques proposed in this second approach can be used to augment a SORION-based object-oriented database system with a logic-based inference engine which will detect security violations.

6. Handling Security Constraints

In our discussion on ORION we enforced the condition that the instance variables of a class are assigned the same security levels as that of a class. An advantage of this approach is that we do not have to handle the multilevel update problem. For example, let the EMP class have instance variables Name, SS# and Salary with the Salary instance variable being classified at the Secret level (EMP as well as the other two instance variables are classified Unclassified). Only a Secret subject can read all the instance variables of EMP. Suppose a Secret subject wants to update the object (John, 20K, 333) to (James, 30K, 333). That is John's name and salary should be changed. The security policy will not permit this Secret subject changing the name John to James because Name instance variable is Unclassified. The Secret subject can however change 20K to 30K. In order to change the name, the Secret subject may have to log-in later at the Unclassified level and perform the update. Another solution is for the Secret subject to polyinstantiate the same object with different values. In general neither solution is desirable.

With our approach, since all of the instance variables of an object are assigned the same level as that of the object, only a subject at the same level can perform the update. However in the real world, it may be necessary to classify the salary instance variable at the Secret level. Our solution

to this problem is to design the schema (which consists of the classes) in such a way that various security constraints which classify instance variables could be handled. We briefly illustrate this solution for three constraints.

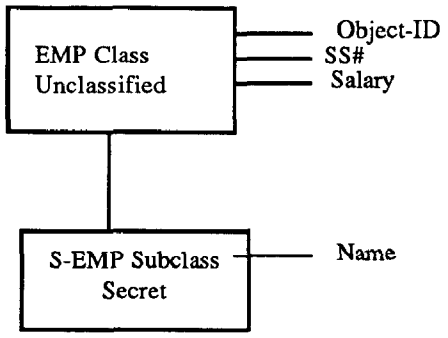

Figure 2a Simple Constraint

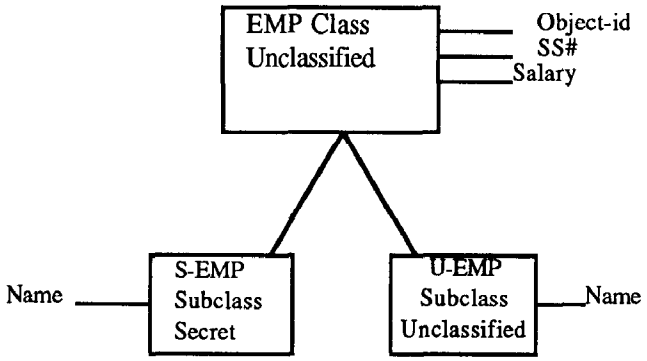

Figure 2b Content Constraint

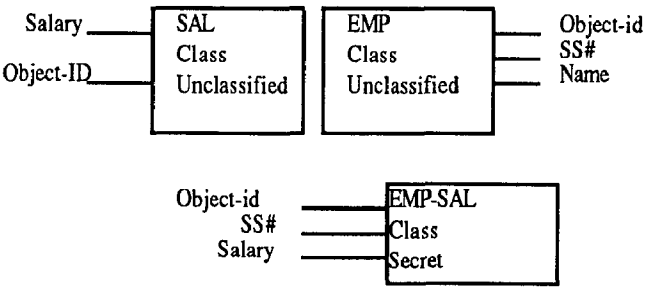

Figure 2c Context Constraint

Example 1: Name instance variable of EMP is Secret - this is an example of a simple constraint.
Solution: Create two classes EMP and S-EMP. The level of EMP is Unclassified and the level of S-EMP is Secret. Make S-EMP a subclass of EMP. The instance variables of EMP are Salary, SS# and Object-ID (note that the object-ID is an instance variable of all objects). The additional instance variable of S-EMP is Name (See Figure 2a).

With this solution, only the Secret subjects (or TopSecret subjects) can read the Name values of employees. The Salary and SS# values can be read by all subjects whose security levels dominate the Unclassified level. If a Secret subject wants to update say (John, 20K, 333) to (James, 30K, 333) then all he has to do is to update the instance in S-EMP.

Example 2: Name instance variable of EMP is Secret if the salary value is greater than 100K - this is an example of a content-based constraint.
Solution: Create three classes; EMP, EMP and U-EMP Make S-EMP and U-EMP to be subclasses of EMP. EMP has instance variables Salary, SS# and Object-ID. S-EMP and U-EMP have Name as an additional instance variable (see Figure 2b).

With this solution, all employees who earn more than 100K will be instances of S-EMP while the remaining employees will be instance of U-EMP. However, EMP will have as its instances all employees with instance variables SS#, Salary and Object-ID.

Example 3: Name and Salary instance variables taken together is Secret; individually they are Unclassified; - this is an example of a context-based constraint.
Solution: Create the classes EMP, SAL and EMP-SAL. EMP is Unclassified with instance variables Object-ID, SS# and Name. SAL is also Unclassified with instance variables Object-ID and Salary. EMP-REL is Secret. Its instance variables are Object-ID, SS# and Salary. That is, each instance in EMP-REL is the relationship between an object in EMP and the corresponding object in SAL (this gives the name-salary relationship and is therefore classified at the Secret level) (See Figure 2c).

7. Conclusion

We have developed a multilevel secure object-oriented data model, SORION, which has evolved from the ORION object model. We have also described the essential features of SORION with examples. Like ORION, SORION is based on the class, object, instance variable and method constructs. In addition, the set construct, IS-A hierarchy and IS-PART-OF hierarchy are also supported by SORION.

We have also discussed mandatory security in an object-oriented system based on SORION. We first described a multilevel security policy and then discussed issues such as handling polyinstantiation and handling the inference problem. Finally we discussed ways of handling security constraints. In this way, it is not necessary to classify the instance variables at levels different to that of the class. Such an approach will solve some of the update problems in secure database systems.

REFERENCES

[BANE87] Banerjee J. et al., "Data Model Issues for Object-Oriented Applications", ACM Transactions on Office Information Systems, Vol. 5, #1, April 1987, pp. 3-26.

[BELL75] Bell D.E and LaPadula L.J., "Secure Computer Systems: Unified Exposition and Multics Interpretation", Technical Report MTIS AD-A023588, The MITRE Corporation, July 1975.

[DWYE87] Dwyer P., G.Jclatis and M.B.Thuraisingham, "Multilevel Security in Database Management Systems", Computers and Security, Vol. 6, #3, June 1987, pp. 252-260.

[DWYE88] Dwyer P., Onuegbe E., Stachour P. and Thuraisingham M.B., "Query Processing in LDV - A Multilevel Secure Relational database management System", Proceedings of the 4th Aerospace Computer Security Conference, Orlando, FL, December 1988.

[KEEF88] Keefe T.F., Tsai W.T. and Thuraisingham M.B., "A Multilevel Security Policy for Object-Oriented Systems", Proceedings of the 11th National Computer Security Conference, Baltimore, MD, October 1988.

[KEEF89] Keefe T.F., Thuraisingham M.B. and Tsai W.T., "Secure Query Processing Strategies", IEEE Computer, Vol. 22, #3, March 1989, pp.63-70

[KONA89] Konar A.F., Felix P. and Thuraisingham M.B., "XIMKON - An Expert Simulation and Control Program", Proceedings of the American Control Conference, Pittsburg, PA, June 1989.

[LUNT88] Lunt T.F. and Thuraisingham M.B., "Security for Hypermedia Systems", Unpublished Manuscript, November 21, 1988; also submitted to Computers and Security.

[MORG87] Morgenstern M., "Security and Inference in Multilevel Database and Knowledge-Base Systems", Proceedings of the ACM SIGMOD Conference, San Francisco, CA, May 1987.

[REIT78] Reiter R., "On Closed World Databases", in Logic and Databases, Ed: Gallaire H. and Minker J., Plenum Press, 1978.

[ROUG87] Rougeau P. and Stearns, "The Sybase Secure Database Server", A Solution to the Multilevel Secure DBMS Problem", Proceedings of the 10th National Computer Security Conference, Baltimore, MD, October 1987.

[STAC89a] Stachour P. and Thuraisingham M.B. "Design of LDV - A Multilevel Secure Relational Database Management System", Accepted for publication in IEEE Transactions on Knowledge and Data Engineering.

[STAC89b] Stachour P. and Thuraisingham M.B., SQL Extensions for Security Assertions", Accepted for publication in Computer Standards and Interfaces Journal.

[THUR87] Thuraisingham M.B., "Security Checking in Relational Database Management Systems Augmented with Inference Engines", Computers and Security, Vol. 6, #6, December 1987, pp.479-492.

[THUR88a] Thuraisingham M.B, "A Functional View of Multilevel Databases", Honeywell Corporate Systems Development Division Internal Notes, April 1988; also to appear in Computers and Security.

[THUR88b] Thuraisingham M.B, Tsai W.T. and Keefe T.F., "Secure Query Processing using AI Techniques", Proceedings of the 21st Hawaii International Conference on Systems Sciences, January 1988.

[THUR88c] Thuraisingham M.B., "Foundations of Multilevel Databases", Presented at the 1st RADC Database Security Invitational Workshop, Menlo Park, CA, May 1988.

[THUR89a] Thuraisingham M.B., "Security in Object-Oriented Database Systems", Accepted for publication in the Journal of Object-Oriented Programming

[THUR89b] Thuraisingham M.B., "Towards the Design of a Secure Data/Knowledge Base Management System", Accepted for publication in Data and Knowledge Engineering Journal.

[THUR89c] Thuraisingham M.B., "Recent Developments in Database Security", Tutorial Proceedings of the (IEEE) COMPSAC Conference, Orlando, FL, September 1989.

[THUR89d] Thuraisingham M.B., "Security Checking with Prolog Extensions", Presented at the 2nd RADC Database Security Invitational Workshop, Franconia, NH, May 1989.

[WOEL86] Woelk D. et al., "An Object-oriented Approach to Multimedia Databases", Proceedings of the ACM Sigmod Conference, 1986.

[ZANI84] Zaniolo C., "Object-Oriented Programming in Prolog",Proceedings of the IEEE Logic Programming Symposium, 1984.