NATIONAL INSTITUTE OF TECHNOLOGY RAIPUR

Security in Objecte Oriented Database Management System

Pratik Pandey
6th Semester Biomedical Department, Database Management System

Guided by :
Prof. Saurabh Gupta

Abstract

TABLE OF CONTENT

1 Introduction

The data model for a general-purpose object-oriented database system attempts to model the "real-world" as a collection of objects, where each object in the "real-world" corresponds to one database object. These database objects can have a complex internal structure composed of other database objects to model the details of the "real-world" objects to which they correspond. Database objects such as these with a complex internal structure are sometimes called composite objects. The internal structure of an object is implemented using instance variables. An instance variable is nothing more than a named slot inside an object that can contain a value. This value may be a primitive data value (such as an integer number or a string of characters) or a pointer (object identifier) to another object. (Some systems do not distinguish between these two types of values all data is an object, including primitive numbers and character strings.) If the value of an instance variable is a pointer to another object, this object may in turn have instance variables pointing to other objects, and so on, to describe the internal structure of the "real-world" object being modeled.

A useful concept related to inheritance is abstract classes. An abstract class is an object class which will never contain objects. Instead, it is used in the inheritance hierarchy to define instance variables and methods that are in common among its subclasses. In this way, the instance variables and methods are defined once in the abstract class, and inherited by all the sub-classes rather than being defined individually in each.

Another aspect of an object-oriented data model (especially one that is behaviorally object oriented) is encapsulation. This means that the internal state of an object the values of its instance variables is hidden from view outside the object, and is accessible only through the methods that have been defined for the object class. This feature of an object-oriented data model seems particularly useful for security enforcement and control of data integrity.

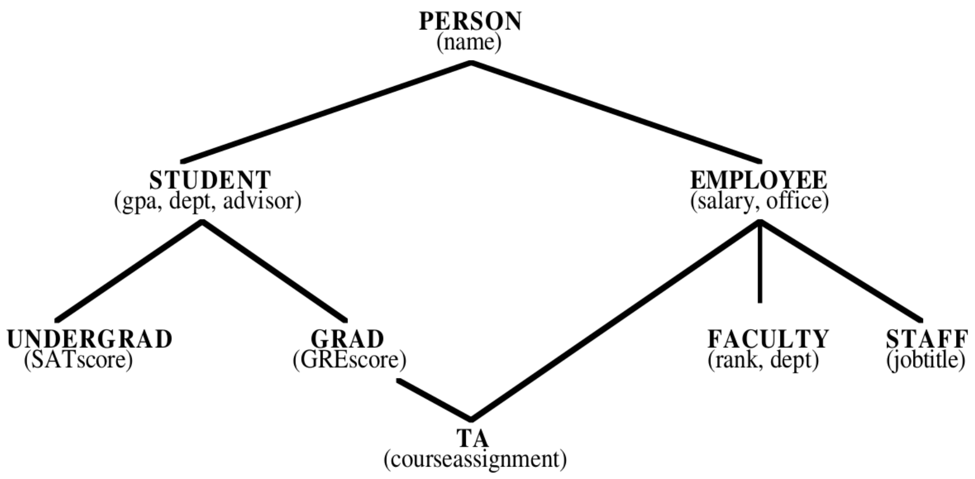


Figure 1: Inheritance Hierarchy Modeling University Staff

2 Security Consideration

Several features of the object-oriented paradigm for database management make it attractive from a security point of view First, an initial layer of protection is provided by the fact that all data is stored as values for instance variables that are encapsulated inside objects and available only through the methods defined for the object's class. These methods can be used to enforce security requirements for the data in the objects using techniques similar to those used in Hydra and other capability-based and abstract data type-based systems. In addition, the enriched semantic modeling capabilities of the object paradigm should allow the "real-world" and its security requirements to be modeled more naturally in the database. Finally, inheritance, at least on the surface, looks like a useful tool for simplifying the specification of security requirements. The security requirements for an application can be defined for object classes near the top of the inheritance hierarchy, and inherited by all classes lower in the hierarchy.

Two orthogonal views of the objects in a database can be used for defining security access controls. The first is to view the objects as composite objects organized into object hierarchies where the root object corresponds to some "real-world" object, and the other objects in the hierarchy define its internal structure. This view is completely independent of the inheritance hierarchy for objects, and is the view that must be used in structurally object-oriented systems. In this view, access controls must be defined to deal with composite objects – that is, authorizing access to an object and all the other objects that define its internal structure.

The orthogonal view of objects for defining access controls is the view presented by the inheritance hierarchy. This view is not available in structurally object-oriented systems, but it becomes important in behaviorally and fully object-oriented systems. It is the view that must be used if inheritance is to be exploited for defining access controls and authorizations. Using this view, access controls and authorizations are defined for the classes and objects in the inheritance hierarchy. The security model must define the semantics for inheritance of these access controls through the inheritance hierarchy. As is discussed below, this is not always easy to do. In fully object-oriented systems, both views can be used simultaneously for defining security requirements. It remains to be seen from future research and experience whether this is a good idea, and how the two views can be coordinated.

3 Data Integrity Control

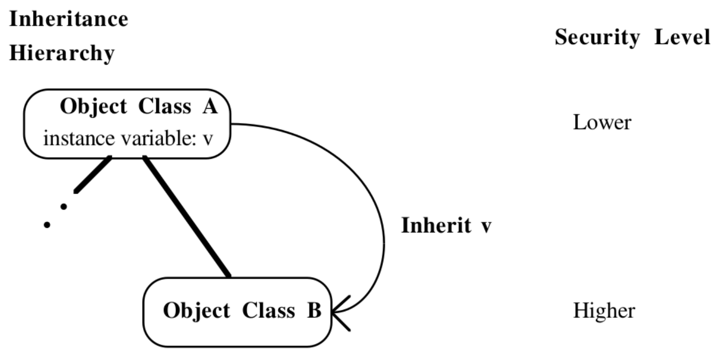
Another aspect of security is integrity of data in a database. The object paradigm enhances enforcement of data integrity in several ways. It encapsulates data inside objects so that the data can be manipulated only by methods defined for the object class. The richer data semantics modeled with the object paradigm make it easier to identify and specify integrity policies to be enforced. And, inheritance can be used to distribute an integrity policy defined for one object class to all of its subclasses. Law and Spooner describe one approach taking advantage of the object paradigm for integrity enforcement within engineering database systems. However, as in the case of discretionary and mandatory access controls, inheritance creates other problems. If object class B is a subclass of A, and inherits instance variable v from A, is there a violation of the integrity of class A if B's methods change the value of v? If integrity constraints defined in A for v are also inherited by B and enforced when B's methods change v, then perhaps not. However, the semantics of this situation must be clearly defined. In general, the fact that the set of all instance variables that B can modify is not defined in one place, but scattered throughout all B's ancestor classes in the inheritance hierarchy, makes enforcement of data integrity more complicated. This is compounded by dynamic binding of method invocation requests to methods as done in many object-oriented systems. This makes it impossible to know until run-time which specific method will be invoked, and what objects it will change.

4 Discretionary Control

Discretionary access controls are concerned with controlling the way individual users manipulate and modify individual objects. Several research efforts have addressed discretionary access controls for composite objects in structurally object-oriented systems. In ORION object-oriented database system models of authorization for relational database systems in two ways. First, they define the notion of implicit authorization allowing the system to deduce new authorizations from prior authorizations explicitly stored in the system. Second, in conjunction with implicit authorizations, they extend the authorization model to cover composite objects. DAMOKLES object-oriented database system model is similar to the first in that it attempts to deal with propagation of access privileges from one object to another to handle authorization for composite objects. However, the approach is very different and is based on dividing an object into descriptive, structural, version, and roles parts, and allowing each of these parts to be dealt with separately for access control purposes.

5 Mandatory Access Controls

Whereas discretionary access controls are concerned with controlling access to individual objects, mandatory access controls are concerned with classifying objects into security levels, and defining controls over the flow of objects between levels. Mandatory access control requirements for trusted computer systems are defined in the "Orange Book". Almost no work to date has addressed mandatory access controls in a general-purpose object-oriented database system. Meadows and Landwehr give a brief description of how the object paradigm might be used to implement a trusted application, and some preliminary work is being done at SRI. As with discretionary controls, the object paradigm offers potential advantages to specification of mandatory access controls due to the richer and more natural data model for modeling the "real-world". However, as pointed out in , for trusted applications, it will be necessary to trust the inheritance mechanism. Verification of the inheritance mechanism may not be easy. Also, for purposes of mandatory access controls, it appears that an object-oriented data model will have all the problems of the relational model(e.g., polyinstantiation), and possibly more, due to the richer semantic content of the data model. Other problems exist also. Consider an example similar to the one in the preceding section where object class B is a child of object class A in an inheritance hierarchy. In this case, assume that A has a higher security level than B. Should B be allowed to inherit instance variables from A? This situation could be interpreted as a flow of information from a higher security level (A) to a lower one (B). Interpreted this way, it violates the Orange Book specifications for trusted secure systems. Suppose the situation is reversed so that object class B has a higher security level than A. Suppose also that B inherits instance variables from A. On the surface, this would not seem to be a violation. But, suppose that a user at the same classification level as B attempts to change the value of one of the instance variables inherited from A for objects in class B. This can be interpreted as a write-down to a lower classification level, again violating the specifications of the Orange Book for trusted secure systems. Since mandatory access controls require any user who accesses objects in class B to have a classification level at least as high as B's, such cases are possible. Both situations discussed above can occur in a general-purpose object-oriented database system. In fact, the second situation may be quite common, occurring anytime a subclass which is a specialization of its parent class contains additional information at a higher security classification than its parent class. As in the discretionary case, these problems can often be avoided for a particular application; but a general-purpose database system must be able to deal with them. Also, as in the case of discretionary controls, the problem is largely one of chosing the semantics to be associated with the inheritance mechanism. Unfortunately, it is not clear that a definition of the semantics for inheritance exists that is consistent with mandatory access control requirements, or that the same semantics will be required in all cases.



User U with the same security level as object class B changes the value of v using one of B's methods.