Object Migration in Object-Oriented Databases

7M-4

Mohamed El-Sharkawi Yahiko Kambayashi
Faculty of Engineering, Kyushu University

1- Introduction

Record oriented data models are not adequate to new applications of database systems. Object-oriented data model is promising to be used in such applications. In this paper, we study updates on objects in object-oriented databases. Specifically, we study the effect of updating an object on its position in the class lattice. An update may affect the object such that it no longer satisfies conditions of its current class. We consider three types of updates on objects: (1) Adding instance variables, (2) Dropping instance variables, and (3) Modifying instance variables. An update may cause object migration. That is, the updated object may change its current class. As the new class may be different from the class before the update, some actions have to be taken. Our objective is to provide a procedure to automatically perform the migration. The idea is explained through an example. Consider the schema shown in Fig. 1. It models people in a university. The schema is represented by a rooted directed graph. Nodes represent classes in the schema. An edge from class C_i to class C_j means that C_i IS-A C_j. The root of the graph is a system defined class called OBJECT. Suppose that instances of class GOOD-GRAD students are defined to be graduate students with total marks exceeding certain value. From this, an update that modifies this value may cause an object to migrate from class GRAD to class GOOD-GRAD. Such object migration has some side effects that should be handled by the system. If class GOOD-GRAD has some instance variables not defined in class GRAD, values of these instance variables have to be provided by a user, or considered null (which sometimes may not be possible).

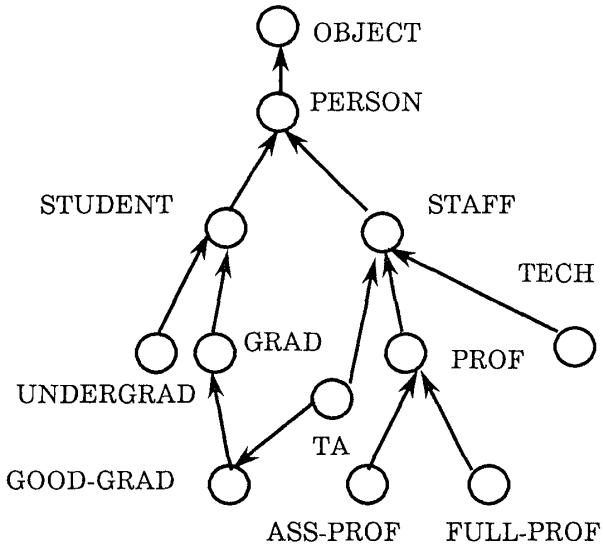

Fig.1 An example schema

Now, we can see that, as an update causes object migration the system has to take some actions. With respect to each update causing migration, we give a group of actions should be taken by the system (or in cooperation with a user).

2- Basic Concepts

In object-oriented data model, entities in the real-world are considered as objects. Properties of an object are divided into two parts, its status and its behavior. Status of the object is captured through its instance variables. Object behavior is encapsulated in a set of methods associated with the object. A method is a code that manipulates the object's status. To manipulate an object a message should be sent to the object. Response to a message is done by executing a method corresponding to the message. Objects communicate via sending messages. Objects having similar properties are grouped together to constitute a class. All objects belong to a class are its instances. Classes in the system are organized in a class hierarchy. An edge between two classes represents IS-A relationship between the two classes. A class inherits all properties of its immediate superclass. It may have also its own properties. For data modeling it is necessary to extend the class hierarchy into a class lattice. A class may have several immediate superclasses and it inherits all of their properties.The class lattice, also the class hierarchy, is rooted such that there is no dangling nodes. The root node is a system defined class called OBJECT. An instance variable gets its possible values from instances of a class in the system. The class domain is either one of system defined basic classes or any other user defined class. Basic classes include INTEGER, REAL, CHAR, and BOOLEAN. An instance variable that gets its value from one of the basic classes is called a basic instance variable, otherwise it is called a complex instance variable. Some instance variables may take any value in its domain, some others, however, have restricted range in which the value should exist. The range R_i of instance variable IV_i is a predicate which may be undefined, a simple predicate, or simple predicates connected by AND/OR. A simple predicate is :
1- $R_i \equiv IV_i$ (OP) K, 2- $R_i \equiv K_j \leqq IV_i \leqq K_u$, where K, K_j, and K_u are elements of D_i (the domain of IV_i), and OP is one of the set $\{=, \neq, <, \leqq, >, \geqq\}$.

3- Object Migration Procedures

In this section, we study updates in object-oriented databases. Objects are grouped into classes, updating instance variables of an object may affect the position of the object in the class lattice. We study effects of three types of updates on objects' positions: Adding some instance variables, Dropping some instance variables, and Modifying some instance variables. Updates may affect the object's position in the lattice as follows:
(1) Adding instance variables: the object will migrate to one of the subclasses of its current class.
(2) Dropping instance variables: the object will migrate to an appropriate one of the superclasses of its current class.
(3) Modifying instance variables: the object may migrate to another class.
We state actions that should be taken by the system as an update causes object migration. These actions depend on the update type. For each type of update, actions should be taken are given.

3-1 Dropping instance variables

(1-a) The system has to check that the dropped instance variables of object O in class C_i are:
all (not a subset of) unfixed non-inherited instance variables. (A fixed instance variable does not change its value due to object migration.)
We have the following three cases:
case 1-1: The dropped instance variables are non-inherited and there is a unique immediate superclass of C_i.

(1-b) the object migrates into its immediate superclass and if there is an instance variable IV_i defined for class C_i of which O is an instance and IV_i overrides IV_j of the superclass of C_i; after the migration the value of IV_i should be replaced by value of IV_j (the original instance variable). Same action is also necessary in case of overridden methods.

(1-c) Non-inherited methods should be dropped.

case 1-2: The dropped instance variables are non-inherited and the class has several immediate superclasses

(1-d) In this case, system has to consult the user in order to determine the new class. For example, after dropping non-inherited instance variables of an object in class TA, the system has to consult the user in order to determine if the object, after the update, belongs to class STAFF or GOOD-GRAD. The new class can be determined automatically if there is some integrity constraint that prevents the updated object to migrate into a specific superclass. For example, the TA object after the update is no longer a STAFF.

case 1-3: The dropped instance variables are the non-inherited and those inherited from C_n, C_n is an immediate superclass of C_i. The object migrates to class C_k, which is the immediate superclass of C_n.

(1-e) Drop instance variables inherited from any class C_h, C_h is a subclass of C_n and a superclass of C_i.

(1-f) If C_i has other immediate superclass which is neither a super nor a subclass of C_n, instance variables inherited from this class has to be dropped when migrating to C_k. Values of instance variables inherited in C_k should be provided or considered null. The problem here, is that if C_i has some fixed instance variables inherited from a superclass which is not in the path of C_j. The user has to be informed with such case.

3-2 Adding instance variables

(2-a) The system has to ensure that the added instance variables are one of those defined in one of the subclasses of the current class. Otherwise, the update is not correct.

(2-b) Value of any property which will be overridden in the new class has to be modified.

(2-c) When instance variables are added to object in class C_i and the new class is C_j such that C_j has several superclasses, values of all instance variables defined in this set (except those of C_i) have to be provided or assumed to be null valued.

3-3 Modification of instance variables

There are the following four cases:
case 3-1: The object's class before modification, C_i, and the new one, C_j, have same immediate superclasses. Following actions have to be taken:

(3-1-a) Unfixed non-inherited properties in class C_i have to be dropped from the object.

(3-1-b) Non-inherited instance variables defined in class C_j have to be added to the object after the update and their values should be provided or considered to be null valued.

(3-1-c) All inherited properties that are overridden in class C_i (C_j) and not so in class C_j (C_i) have to be modified in the object after the update.

(3-1-d) If C_j has superclasses different from those of C_i, values of instance variables that are inherited from these classes should be provided or considered to be null valued.

case 3-2: The object's class before modification (C_i) and the new one (C_j) have different immediate superclass.

(3-2-a) All unfixed properties defined in class C_i will be dropped from the new object.

(3-2-b) All properties defined in class C_j become properties of the new object.

case 3-3: New class C_j is a superclass of the original class C_i. In this case, there are two possibilities, C_i has only one immediate superclass C_j and C_i has several immediate superclasses. In the first case, there are two situations; 1-when C_j is the unique immediate superclass of C_i, the following actions have to be done:

(3-3-a) Unfixed non-inherited properties have to be dropped from the object after update.

(3-3-b) Overridden inherited properties defined in class C_i have to be modified in the new object.
The second situation when: 2- C_j is not the immediate superclass of C_i, actions (3-3-a) and (3-3-b) have to be done in addition to the actions:

(3-3-c) Properties inherited from any class C_k such that C_k is a subclass (and superclass) of C_j (C_i) has to be dropped.

(3-3-d) If C_i has immediate superclass C_k which is not a subclass of C_j, properties inherited from this class has to be dropped. Same problem of fixed instance variables inherited from a superclass which is not in the path of C_j will arise.
In the second case, when C_i has several immediate superclasses,:

(3-3-e) All unfixed non-inherited properties defined in C_i and overridden properties originally defined by a superclass C_k ($C_k \neq C_j$) have to be dropped from the object after update.

(3-3-f) Values of instance variables defined in class C_j and overridden in class C_i have to be modified in the new object.

case 3-4: New class C_j is a subclass of the original class C_i. There are two situations:
case 3-4-1: there is no class C_k such that C_k is a superclass of C_j, the following action will be taken:

(3-4-a) Properties defined in class C_j are added to the object and values of overridden inherited properties should be modified.

case 3-4-2: C_j has superclasses other than C_i.

(3-4-b) is similar to (3-4-a).

(3-4-c) Properties inherited will be defined and values of inherited instance variables from these classes must be provided or set to be null valued.

(3-4-d) All overridden inherited properties from superclasses other than C_i have to be defined in the new object.
Using these steps system automatically performs actions should be taken when objects are updated.

References

[1] ACM TOIS, Vol.5, No.1, Jan. 1987.
[2] Banerjee, J., et al. Proc. ACMSIGMOD, 1987.