History Update in Temporal Object-Oriented Databases

2L−10 Mohamed El-Sharkawi+ Yahiko Kambayashi*

+Faculty of Engineering, Kyushu University

*Faculty of Engineering, Kyoto University

1- Introduction

Advanced database applications, e.g. CAD and OIS, have two requirements. First, they need semantically rich data model, since record oriented data models are not able to support these applications. Object-oriented data model is promising to be used in such applications. The second requirement is extending and adding some new features to database management systems. Among these features are support of long transactions, controlling versions, and adding the time dimension to the data. In this paper, we discuss problems in updating the history in temporal object-oriented databases (TOODB). (Detailed discussions of some problems in answering queries in TOODB's can be found in [1].) In object-oriented databases an update on an object may enforce it to change its class [2]. That is the object will *migrate* to another class. Due to object migration updating the history of an object may affect the position of the current version of the object (it may also affect other versions). Accordingly, when the history of an object is updated, the system has to check whether this update will affect one of the object's versions.

2- Basic Concepts

2-1 Object-Oriented Data Model

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 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.

2-2 Object Migration in OODB's

An update may cause object migration. That is, the updated object may change its current class. The idea is explained through an example. Consider the schema shown in Fig. 1. It models people in a

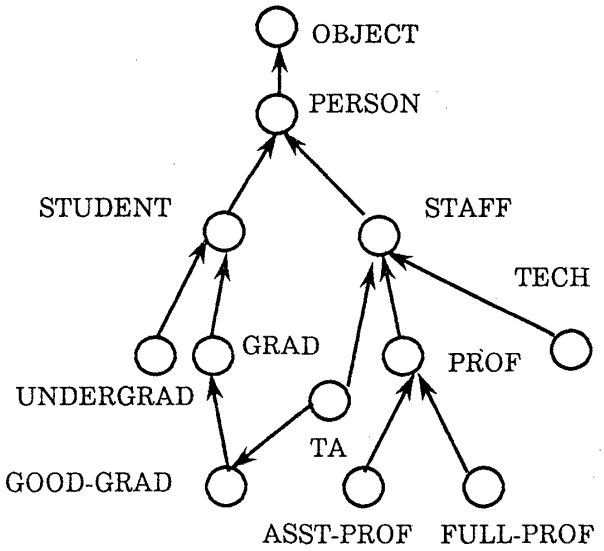

Fig.1 An example schema

university. 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.

2-3 Temporal Databases

For new applications, it is necessary to store the current status as well as the history of the world. In temporal databases, users may access history data by using temporal queries. The query contains, in addition to conditions should be satisfied by the output, the time at which the output is valid. For example, find the Salary of Tom from March 1989 to Feb. 1990. Several efforts have been done to extend the relational data model with the time dimension. Temporal databases are classified into three types rollback, historical, and temporal databases. This classification is based on two criteria, the type of the time supported by the database and whether it is permitted to update the history. There are two types of time: transaction time and valid time. Transaction time is the time at which information was stored in the database. Valid time is the time at which the real-world was changed. Rollback databases support transaction time, historical databases support valid time, and temporal databases support both transaction and valid times.In rollback databases, however, it is not permitted to update the history as known of now. In our discussion it is immaterial which time is supported.

3- History Update in TOODB's

Databases that support the time dimension are classified into three types [SA] rollback, historical, and temporal. One of the features of historical and temporal databases is the possibility of updating the history as known of now. In this section, we will study history update in TOODB's. Modifying the history in TOODB's may have some side effects. To show that consider the schema shown in Fig. 2. At

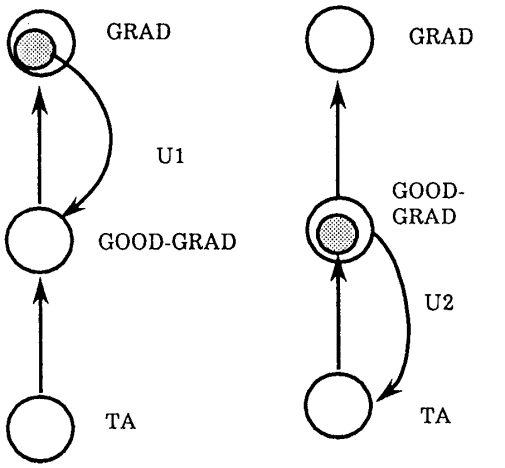

Fig.2 The effect of history update

t_1 update U_1 is done on object O to increase Total-Markes of O to 87. Accordingly, O will migrate to class GOOD-GRAD. At t_2 O is updated again to move into class TA. Later, at t_3, it is discovered that the Total-Markes of O should be corrected to be 78. If this history update is done, O will not satisfy conditions to be an instance of class TA. The system should consult the user before correcting the history.

There are two possibilities when the history is updated:

1- The correction of an update done at t_i does not affect any update done at t_j, $t_j > t_i$, in this case, the correction is applied. It happens when:

(a) The corrected instance variable is not one of those causing objects in the class to migrate, for example correcting an update that modified the address of object O when it was an instance of class GRAD will not affect its position in class TA.

(b) The corrected instance variable is one of those causing objects in the class to migrate, however the correction will not affect the object's current position. For example, in Fig. 2, correcting the Total-Markes from 87 to 97 will not invalidate the existence of O in class TA.

2- The correction of an update done at t_i does affect some update done at t_j, $t_j > t_i$, the corrected update may affect other updates in the following cases:

(a) The correction will make one of the migration conditions satisfiable. For example, an object in class C_i may migrate into class C_j iff: $A \geq 50$, $B < 80$, and $C = 20$, where A, B, and C are instance variables. Suppose at time t_1 A is updated to be 45, at time t_2 B is updated to 70 and C to 20. The update at time t_2 does not cause any migration. Later, at time t_3 it is discovered that the first update was not correct, A should be 54 instead of 45. If this mistake is corrected, the update at time t_2 should be completed, since, now, all conditions of migration are satisfied.

(b) The corrected instance variable does not cause any migration to objects in the class, however, it may cause migration of some versions in some other class. For example, consider an object O* that has two versions one in class C_1 and the other in class C_2. The version in class C_2 was generated from the version in class C_1. The condition for O* to migrate from C_2 to some other class C_3 is that $A > 45$. Assume that a history update is applied on the version in class C_1 to make its A equal to 60. Applying this history update will enforce the object to migrate from class C_2 to class C_3.

The system has to detect whether correcting an update has side effects. When the correction does not have any effect, it can be executed. Otherwise, the user has to be informed and the decision of performing the correction should be taken by the user. The system may support the user with information that help him to take the decision. These information may include the instance variables that causes the side effects, the new position of the object when the history is corrected, and etc.

Before we give an outline of the procedure that checks history corrections, we need the following definitions.

Definition 3. Associated with each class C_i a set of *migration conditions* denoted by $MC(C_i)$. A migration condition has the form:

Instance variable OP Value, where OP belongs to $\{=, \neq, >, \geq, <, \leq\}$ and Value is obtained from the domain of the instance variable.

Definition 4. The class of the most recent version of an object O is called the *current class* of O and denoted CC(O). The class of the version that will be updated is called the *history update class* and denoted HUC(O). The *history path* of object O, denoted HP(O), is the union of HUC(O), CC(O), and each class C_j, in the path between HUC(O) and CC(O), such that O has a version belongs to C_j.

Note that the history update class (HUC) and the current class (CC) of an object can be obtained by consulting class UPDATES.

Procedure History-Update-Validation
Input: Given a correction U(O, IV_i, NEW-VALUE)
Output: The correction is done, or, the user is informed of the side effects.
Method: **for** every class C_k in HP(O) **do**
if correcting the history makes one of $MC(C_k)$ true **then** inform the user; **exit;**
end do;
perform the correction;
exit;
END HISTORY UPDATE

References

[1] El-Sharkawi, M. "Answering Queries in Temporal Object-Oriented Databases," DASFAA, March 1991 (to appear).

[2] El-Sharkawi, M. Kambayashi, Y "Object Migration Mechanisms to Support Updates in Object-Oriented Databases," Proce. PARBASE-90.