Derived Data Update in Semantic Databases*

I-Min Amy Chen and Dennis McLeod
Computer Science Department
University of Southern California

Abstract

The derived data update problem involves the transformation of modifications of derived facts into corresponding changes to base facts and other derived facts. Using a database schema defined with a generic semantic database model which includes derived data specifications, techniques and algorithms are provided for appropriately modifying base data when derived data are changed. An experimental prototype system based upon this approach has been developed.

Keywords: derived data update, semantic databases, update propagation

1 Introduction

Derived data in a database is useful to accommodate multiple viewpoints on information, to maintain frequently referenced/computed data, and to support database protection/security [8, 9, 10, 11]. Consequently, a database is viewed here as consisting of base data, or explicitly stored facts, and derived data, which are computed from base data or derived data by derivation rules.

It is important for database users to be able to modify derived data directly, rather than issuing a less natural and possibly more complicated modification on the underlying base data. The *derived data update problem* [4, 5, 6, 7, 12, 13, 16] involves the transformation of modifications of derived data into corresponding changes to base data and other derived data. The transformation of modifications is usually termed *update propagation.*

The derived data update problem is considered difficult because update propagation is not always unique; there may be no way or more than one way to transform modifications. For example, the object class A may be defined as derived by the union of classes B and C; inserting an instance to A can cause the instance to be inserted to B, to C, or to both. In this case, the update transaction is *ambiguous.*

The derived data update problem is similar to the *view update problem* [4, 5, 6, 7, 12, 13, 16] in the terminology of relational databases. Because it is difficult to support update propagation, most database systems permit view update in extremely limited cases, or allow view update only if update propagation procedures are explicitly specified by designers or users. The problem of supporting derived data update in semantic (structurally object-oriented) databases is largely open; most existing approaches restrict derived data to be not changeable (e.g., IFO [1], Galileo [3], SDM [9], and Taxis [14]). DAPLEX [15] supports derived data update, but users must specify the corresponding update on the base data; the user is responsible for assuring the correctness of such derived data update transactions.

Since conceptual schemas specified with semantic database models are richer than record-oriented (e.g., relational) schemas, it is possible to automatically decide update propagation in a more substantive manner. Our approach to the derived data update in a semantic database is termed *schema-based,* since update propagation is based on the schema definition, which includes class (type) and attribute definitions, constraints, and derivation rules. For example, changes to an attribute will have direct effects on its inverse, the attributes derived from this attribute, and the other attributes from which this attribute is derived. Changes to instances of a class will have direct effects on the immediate superclass and subclasses of the class. In our approach, schema information is used to yield update rules. As a simple example, there is an update-rule which states that if class A is the union of classes B and C, then insertion to either class B or class C will cause insertion to class A. Update propagation is based upon these update rules.

The organization of the remainder of this paper is as follows. A generic semantic database model upon which the schema-based approach is based is briefly

---

*This research was supported, in part, by the USC AT&T Affiliates Program.

Proceedings of the Fifteenth International
Conference on Very Large Data Bases

Amsterdam, 1989

introduced in section 2; this data model is a simplification of SDM [9]. In section 3, we describe derived data update rules based on schema information. An experimental prototype system embodying a derived data update algorithm based upon the rules is presented in section 4. Conclusions and directions for future research are given in section 5.

2 A Generic Semantic Database Model

A generic semantic database model (GSDM), which embodies the main features of prominent semantic database models [11], is used as a basis for our approach to schema-based derived data update. The GSDM is a simplification of SDM [9]. The **Appendix** contains an example database schema describing a university application environment specified using the GSDM.

A *database schema* in GSDM consists of a collection of *classes*. Each class has a set of instances (members), which are the objects belonging to the class. Every class has a class name, and an associated collection of attributes. There are two kinds of attributes: member and class attributes. Member attributes describe properties of instances of a class; for example, every instance of class *COURSE* has a course number, an instructor, a teaching assistant, and students enrolled in the course. Class attributes specify properties of the class as a whole, e.g., the total number of instances in the class *COURSE*. As shown in the **Appendix**, class *COURSE* has a class attribute *Total-courses*, and the four member attributes *Course-number*, *Taught-by*, *Has-ta* and *Students-currently-enrolled*. A class may have a cardinality limitation, which restricts the number of instances of this class. For example, the class *TEACHING-ASSISTANT* has the cardinality limitation: *with size between 0 and 25*.

Each member attribute has a name and a specification of the range (*value class*) of this attribute; for example, attribute *Course-number* of *COURSE* has value class *INTEGERS*. If the attribute is derived, then there is also a derivation rule; e.g., a person's monthly income can be derived by: *Annual-income / 12*. An attribute can also be specified as the *inverse* of an attribute of some other class.

An attribute can have associated constraints, e.g., single-valued or multi-valued. A multi-valued attribute may have an associated upper and lower bound. For example, attribute *Takes* of *STUDENT* has an associated constraint: *multi-valued with size between 2 and 5*. An attribute which is not allowed to have null values is specified by *may not be null*; e.g., the *Name* of a person is constrained in this manner. If the value of an attribute cannot be changed, then *not changeable* is specified. *Exhausts value class* requires that all members of the value class must be the value of the attribute for some instance(s). An attribute for which overlapping values are not allowed is specified with *no overlap in values*.

The specification of a class attribute is similar to that of a member attribute, except that class attribute does not have an inverse, and does not have constraints specific to member attributes such as *exhausts value class* or *no overlap in values*. A class attribute can have a derivation which involves properties of the class as a whole, such as the total number of instances in the class; e.g., class attribute *Total-courses* of class *COURSE* has the derivation: *number of members in this class*.

Classes in a GSDM database schema are organized as a collection of directed acyclic graphs (DAGs). A subclass can be derived from other classes or subclasses by one of the seven subclass constructors. A subclass can be (1) specified by attribute predicate; (2) explicitly specified (user-controlled); (3) the intersection of two classes; (4) the set difference of two classes; (5) the union of two classes; (6) the current set of values of a given attribute or (7) specified in certain format[1]. For example, subclass *COMPUTER-MAJOR-STUDENT* has the derivation rule: where Major = 'CS' or Major = 'CE' (a category 1 subclass). Subclass *STUDENT* is derived from *PERSON* with the derivation: specified (a category 2 subclass). Subclasses *BOTH-TA-RA* and *STUDENT-EMPLOYEE* are the intersection and the union of classes *TEACHING-ASSISTANT* and *RESEARCH-ASSISTANT* (category 3 and category 5 subclasses, respectively). Subclass *UNDERGRADUATE* is the set difference of classes *STUDENT* and *GRADUATE* (a category 4 subclass). *COURSES-TAKEN* has the derivation rule: where is a value of *Took* of *STUDENT* (a category 6 subclass).

A *database transaction* (or *transaction* for short) is a set of *database commands* (or *commands*) which after execution will not violate database integrity constraints. There are five kinds of modification commands: add instances to a class, remove instances from a class, change the value of a class attribute, add values to a member attribute, and delete values from a member attribute. Each of these five commands may cause update propagation if data changed is derived. In some cases, derived data update propagation rules can be constructed for individual commands, and in other cases the combination of commands within a transaction must be considered.

3 A Schema-Based Approach

We propose in this section an approach to derived data update for the GSDM. A list of update rules guiding derived data update is given. A derived data update transaction is allowed only if it follows update rules.

---

[1] Subclasses derived using data formatting specifications (e.g., for strings) are not explicitly considered here.

For example, "insertion to a class A which is the intersection of two classes B and C will cause the same instance to be inserted to B and C" is a sample update rule. An update rule may include default actions to resolve ambiguities. In the practical use of our approach, the database designer or user can override the defaults.

Update rules are gathered from the constructions of classes or attributes, and from the restrictions of classes or attributes. We call the former *structural information*, and the latter *constraints*. Structural information includes class definitions, subclass derivations, and derived attribute definitions. Constraints are specified in the class or the attribute definitions; examples of constraints include the cardinality limitation of attributes and classes, the existence of inverse of an attribute, and the restrictions that an attribute value cannot be null or cannot be changed. Update rules utilizing both structural information and constraints are presented here.

3.1 Update Rules From Structural Information

3.1.1 Derived Attributes

Derived attributes are those attributes which have associated derivation rules. For example, class *STUDENT-EMPLOYEE* has the derived attribute *Benefit* with the derivation rule: *Tuition-coverage* + *Stipend*. Changing the value of *Tuition-coverage* or the value of *Stipend* will cause the value of *Benefit* to change. However, direct update on the value of *Benefit* is not allowed since the update propagation is not unique.

Class *PERSON* has the derived attribute: *Monthly-income* with the derivation: *Annual-income* / 12. If a person's income changes, then the change can be made on *Annual-income*, and *Monthly-income* will change to the new value of *Annual-income* divided by 12. Changes can also be made to the value of *Monthly-income*, in which case the value of *Annual-income* is changed to be the new value of *Monthly-income* multiplied by 12.

Whether an update on a derived attribute is allowed depends upon how that attribute is derived. We have the following update rule for derived attribute update:

Update-Rule 1 (One-to-one Onto) *Update of a derived attribute is allowed if the derivation function f is a one-to-one onto total function. Update of underlying data will cause update of the derived attribute by using the function f. Update of the derived attribute will cause update of the underlying data by using the inverse of function f.*

3.1.2 Derived Subclasses

As stated in section 2, there are six kinds of subclasses in the GSDM considered here: (1) specified by attribute predicate, (2) explicitly specified, (3) the intersection of two classes, (4) the difference of two classes, (5) the union of two class, and (6) the current set of values of a given attribute. The following discussion is based on these subclass categories.

COMPUTER-MAJOR-STUDENT is a subclass of *STUDENT* with the derivation rule: *Major* of *STUDENT* is either 'CS' or 'CE'. Inserting a *COMPUTER-MAJOR-STUDENT* P1 with major 'CS' will propagate an insertion to class *STUDENT*. Inserting a *COMPUTER-MAJOR-STUDENT* P2 with major 'MATH' is an error. Inserting a *STUDENT* P3 with major 'CS' or 'CE' will cause the same student P3 to be inserted to class *COMPUTER-MAJOR-STUDENT*. Deleting an instance P4 from *STUDENT* will cause the deletion to be propagated to *COMPUTER-MAJOR-STUDENT*, providing P4's major is 'CS' or 'CE'. However, deleting a student from *COMPUTER-MAJOR-STUDENT* is ambiguous. This can be interpreted as a change of the student's major, or as a removal of that student from the class *STUDENT*.

Update-Rule 2 (Attr-Pred-Based-1) *Let T_c be a subclass of T_p which is derived by using an attribute predicate P. Inserting an instance A to T_c which satisfies P will cause the same instance to be inserted to T_p. However, if A does not satisfy P, then this is an error. Inserting an instance B to T_p, provided B satisfies P, will cause B to be inserted to T_c. Deleting an instance C from T_p will cause C to be deleted from T_c if C is also an instance of T_c.*

Update-Rule 3 (Attr-Pred-Based-2) *Let T_c be a subclass of T_p which is derived by $A = V$, where A is an attribute of T_p and V is a specified value. Inserting a new instance x to T_c will cause x to be inserted to T_p, and value V to be added to the attribute A of x.*

STUDENT is an explicitly specified subclass of class *PERSON*. Any insertion to *STUDENT* will be propagated to *PERSON*. Any deletion from *PERSON* will be propagated to *STUDENT*. New data inserted to *PERSON* can either be propagated to class *STUDENT* or not. To simplify the problem, the default action is not to propagate. Also by default, the deletion of an instance from *STUDENT* will not be propagated to *PERSON*.

Update-Rule 4 (Specified) *Let T_c be a subclass of T_p with derivation "specified." Inserting an instance A to T_c will cause the same instance to be inserted to T_p if it is not there already. Deleting an instance B from T_p will cause deletion from T_c if B is also an instance of T_c. Insertion to T_p or deletion from T_c will not be propagated (by default).*

Subclass *BOTH-TA-RA* is an intersection of class *TEACHING-ASSISTANT* (*TA* for short) and class

RESEARCH-ASSISTANT (*RA* for short), i.e., derived by "is in TA and is in RA." Inserting an instance P1 to class *TA*, providing P1 is also a research assistant, will cause P1 to be inserted to class *BOTH-TA-RA*. Inserting P2 to *BOTH-TA-RA* will cause P2 to be inserted to both classes *TA* and *RA*. Deleting P3 from *RA* will cause P3 to be deleted from *BOTH-TA-RA* if P3 is an instance of *BOTH-TA-RA*. Deleting P4 from *BOTH-TA-RA* is ambiguous; it can mean deleting P4 from *TA*, from *RA*, or from both.

Update-Rule 5 (Intersection) *Let T_a, T_b and T_c be subclasses of T_p, where T_c is derived by "is in T_a and is in T_b." Inserting x to T_a (or T_b), providing x is also in T_b (or T_a), will cause x to be inserted to T_c. Inserting x to T_c will cause x to be inserted to T_a and T_b (and T_p). Deleting an instance y from either T_a or T_b will cause y to be deleted from T_c if y is also an instance of T_c.*

Class *UNDERGRADUATE* and class *GRADUATE* are both subclasses of *STUDENT*. *UNDERGRADUATE* is a set difference of *STUDENT* and *GRADUATE* (i.e., derived by "is not in GRADUATE"). Inserting a student P1 to class *STUDENT* can cause P1 to be inserted to *GRADUATE* or *UNDERGRADUATE*. If the derivation of *GRADUATE* is satisfied (i.e., *Level* of P1 is equal to 'Graduate'), then the instance P1 will be inserted to *GRADUATE*. If the derivation is not satisfied, then P1 will be inserted to *UNDERGRADUATE*. Of course, there are cases where update propagation is ambiguous, e.g., P1's *Level* is unknown, or the derivation of *GRADUATE* is "specified." A reasonable default here is to insert the instance to class *UNDERGRADUATE*. Inserting an instance P2 to *GRADUATE* will cause P2 to be inserted to *STUDENT* if P2 is not already an instance of *STUDENT* (therefore, it is not an instance of *UNDERGRADUATE* either.) If P2 is an instance of class *STUDENT* and class *UNDERGRADUATE*, then P2 will be removed from class *UNDERGRADUATE*. Inserting a student P3 to class *UNDERGRADUATE* will cause the same instance P3 to be inserted to class *STUDENT* if it is not in *STUDENT* and is not in *GRADUATE*. However, it will be an error if P3 is an instance of *GRADUATE*. Changing a student from *GRADUATE* to *UNDERGRADUATE* must be done by first deleting that student from *GRADUATE*, and then inserting the same student to *UNDERGRADUATE*.

A transaction which deletes an instance P4 from *STUDENT* will delete P4 from *GRADUATE* or *UNDERGRADUATE* depending upon the nature of P4. Whether deleting an instance from *GRADUATE* is legal or not and how it will affect *STUDENT* depends upon how *GRADUATE* is derived. If deleting an instance from class *GRADUATE* is legal and it does not affect the same instance in *STUDENT*, then that instance will be added to *UNDERGRADUATE*. Deleting a student from *UNDERGRADUATE* is ambiguous; the update propagation can be either to delete that instance from *STUDENT*, or to add that instance to *GRADUATE*. Since it is not always possible to make the instance a member of class *GRADUATE*, the default action is to delete that instance from *STUDENT*.

Update-Rule 6 (Difference) *Let T_a and T_b be subclasses of T_p, and T_b is derived by "is not in T_a." Inserting an instance x to T_p will cause x to be inserted to T_a if the derivation of T_a is satisfied; it will cause x to be inserted to T_b if the derivation of T_a is not satisfied. If the derivation of T_a cannot be evaluated*[2]*, then x will be inserted to T_b by default.*

Deleting an instance y from T_p will cause y to be deleted from T_a or T_b, depending upon to which class y belongs. Deleting an instance y from T_a will cause y to be added to T_b if the deletion is legal and y is still an instance of T_p after it is deleted from T_a. Deleting an instance y from T_b will cause y to be deleted from T_p by default.

Classes *TEACHING-ASSISTANT* (*TA* for short), *RESEARCH-ASSISTANT* (*RA* for short), and *STUDENT-EMPLOYEE* are all subclasses of class *GRADUATE*. *STUDENT-EMPLOYEE* is the union of *TA* and *RA* (i.e., derived by "is in TA or is in RA"). Inserting an instance P1 to *TA* or to *RA* will cause P1 to be inserted to *STUDENT-EMPLOYEE*. Inserting a student to *STUDENT-EMPLOYEE* is ambiguous, because that student can be a teaching assistant or a research assistant. Deleting an instance from TA (or RA) will cause the instance to be deleted from *STUDENT-EMPLOYEE* if that instance is not in RA (or TA). Deleting an instance from *STUDENT-EMPLOYEE* will cause that instance to be deleted from both *TA* and *RA*.

Update-Rule 7 (Union) *Let T_a, T_b and T_c be subclasses of T_p; T_c is derived by "is in T_a or is in T_b." Inserting an instance x to T_a or T_b will cause x to be inserted to T_c if x is not already in T_c. Deleting an instance y from T_a (or T_b) providing y is not in T_b (or T_a) will cause y to be deleted from T_c. Deleting y from T_c will cause the same instance y to be deleted from both T_a and T_b (if it is there).*

Class *COURSES-TAKEN* is a subclass of *COURSE* with the derivation: is a value of *Took* of *STUDENT*; this means that the instances of *COURSES-TAKEN* are those instances of *COURSE* for which there exists student(s) who have taken that course. If the fact "P1 took CS102" is added to the database, then the corresponding fact associated with *COURSES-TAKEN*, viz., "CS102 is a course taken by P1" will be inserted to the database. If the fact "CS102 is a course taken by P1" is inserted to the database, then the corresponding fact "P1 took CS102" will be generated. Removing

---

[2] That is, the derivation rule is "specified," or values of attributes in the derivation are unknown.

the fact "P1 took CS102" will cause the fact "CS102 is a course taken by P1" to be deleted from the database, and vice versa. This example can be generalized as follows:

Update-Rule 8 (Attribute-Value-Based-Class) *Let T_p be a class with member attributes $A_1, \cdots, A_n$, and T_c is a class with derivation "is a value of A_i of T_p" (i is between 1 and n). Let t be an instance of T_p. Inserting a new value x to attribute A_i of instance t will generate corresponding data in T_c. Inserting new data to T_c will generate corresponding data in A_i of T_p. Deleting x from attribute A_i of instance t will remove corresponding data in T_c. Deleting data from T_c will cause the corresponding data in A_i of T_p to be removed.*

3.2 Update Rules From Constraints

There are cases when structural information is inadequate to decide a unique update propagation, but in which constraints embedded in the schema definition may be used to resolve the ambiguity. Examples of such constraints (on attributes) are *multi-valued* and *may not be null*. Value classes of attributes also provide information which is sometimes useful in resolving ambiguities.

3.2.1 Attributes and Their Value Classes

Class *STUDENT-EMPLOYEE* is the union of *TA* and *RA*. According to the discussion in the previous subsection, insertion to a union subclass such as *STUDENT-EMPLOYEE* is ambiguous. Information from member attributes and their value classes can however be used to decide a unique update propagation. For example, inserting an instance P1 to *STUDENT-EMPLOYEE*, providing that P1 is in charge of a course CS101, will cause this instance P1 to be added to *TA*, since *Is-in-charge-of*is an attribute of *TA*. However, P1 may also be an instance of *RA*, if classes *TA* and *RA* are not disjoint. A default action is to insert P1 to *TA* only.

Update-Rule 9 (Value-Class) *Suppose T_c is a subclass with derivation "is in T_a or is in T_b," T_a has an attribute Attr with value class T_d, and T_b does not have such an attribute. Inserting an instance x to T_c given Attr of x has a value from T_d will cause x to be inserted to T_a only (by default).*

3.2.2 Multi-valued with Size Restrictions

A multi-valued attribute may have a specified upper and lower bound, limiting the size of the attribute. A class can also have the size limitations. This information can be used to decide update propagation.

Class *STUDENT-EMPLOYEE* is the union of *TA* and *RA*. Suppose TA has a size limitation to be between 15 and 25. Insertion to *STUDENT-EMPLOYEE* may be ambiguous, but if there are already 25 TAs, then the new student employee can only be a RA, otherwise there will be a constraint violation. In this case, the system will suggest that the new data should be an instance of RA.

There are similar situations for intersection. *BOTH-TA-RA* is an intersection of *TA* and *RA*. Deleting an instance from *BOTH-TA-RA* providing there are 15 TAs will suggest the deletion should only be propagated to class *RA*.

Update-Rule 10 (Max-Size) *Let T_c be a subclass with the derivation "is in T_a or is in T_b"; T_a has a size limitation to be between Min and Max. Inserting a new instance x to T_c given T_a already has Max size will suggest that x should be inserted to T_b only.*

Update-Rule 11 (Min-Size) *Let T_c be a subclass with the derivation "is in T_a and is in T_b"; T_a has a size limitation to be between Min and Max. Deleting an instance x from T_c given T_a has Min size will suggest that x should be deleted from T_b only.*

3.2.3 Nullable

A class with an attribute specified as "may not be null" will prevent any insertion to that class unless the value of that attribute is given. Consider the class STUDENT-EMPLOYEE, and suppose attribute *Is-in-charge-of* of *TA* cannot have null values. Inserting an instance P1 to *STUDENT-EMPLOYEE* without giving a value to *Is-in-charge-of* will suggest that P1 is a research assistant.

Update-Rule 12 (Nullable) *Let T_c be a union of T_a and T_b; T_a has an attribute Attr which is specified as "may not be null." Inserting an instance x to T_c without giving a value to Attr will suggest that x should be inserted to T_b only.*

3.2.4 Changeable

In addition to the one-to-one case described previously, there are other cases in which a derived attribute can be updated. For example, attribute *Benefit* of class *STUDENT-EMPLOYEE* is defined to be: *Tuition-coverage* + *Stipend*. If *Stipend* is defined to be "not changeable," then a change to the attribute *Benefit* can result in a unique change to the attribute *Tuition-coverage*.

Update-Rule 13 (Changeable-1) *Let class T have attributes $A_1, \cdots, A_n$. Attribute A_n is derived by a function f with attribute names $A_1, \cdots, A_m$ $(m < n)$. If $A_1, \cdots, A_{m-1}$ are not changeable, then update on A_n will result in a unique update on A_m, which is* *an inverse of function f with values of* $A_1, \cdots, A_{m-1}$ *fixed.*

Suppose *MEN* is a subclass of *PEOPLE* with attribute *Sex* equal to "male," and attribute *Sex* is specified as "not changeable." In this case, deleting an instance from *MEN* can only mean deleting the corresponding instance from *PEOPLE*, but not changing the value of attribute *Sex*.

Update-Rule 14 (Changeable-2) *Let* T_c *be a subclass of* T_p *based on the attribute predicate P. If attributes used in P are all not changeable, then deleting an instance x from* T_c *will result in deleting x from* T_p*.*

3.2.5 Exhausts Value Class

Exhausts value class provides a global guide to deciding update propagation. Let *STUDENT*, *SCHOOL* and *COMPANY* be three classes; *ORGANIZATION* is the union of *SCHOOL* and *COMPANY*. Suppose *STUDENT* has an attribute *Attends* which has the constraint *exhausts value class* with value class *SCHOOL*. [3] Inserting "ABC" as an instance of *ORGANIZATION* will make "ABC" an instance of *COMPANY*.

Update-Rule 15 (Exhausts-Value-Class) *Let* T_a, T_b, T_c *and* T_d *be classes. Let* T_d *be defined as: "is in* T_b *or is in* T_c*." Suppose* T_a *has an attribute Attr with value class* T_b *with constraint "exhausts value class." Inserting an instance x to* T_d *without corresponding instances in Attr of* T_a *will insert x to* T_c *only.*

4 Experimental Prototype

An experimental prototype system based upon the approach described above has been designed and implemented. Currently this prototype is implemented in the C language and runs on SUN/UNIX. This prototype handles update propagation based upon the rules described above. The data representation and algorithms this prototype uses, along with a brief description of the implementation, are presented here. Two working examples are also included.

4.1 System Implementation

Figure 1 illustrates the overall structure of the experimental prototype. Here, a user first defines a database schema in the GSDM; this GSDM external schema is then translated into an internal representation called the *kernel database representation*. The internal representation consists a set of triplets with the format (domain, mapping, range). For example, a schema description such as "STUDENT is a subclass of PERSON with derivation *specified*" is encoded into the internal triplets: (STUDENT, has-superclass, PERSON) and (STUDENT, has-derivation, "specified"). Database facts such as "P1 is a person with name John" are also encoded into internal triplets: (PERSON, has-instance, P1) and (P1, Name, 'John'). In this way, both data and meta-data are represented uniformly [2].

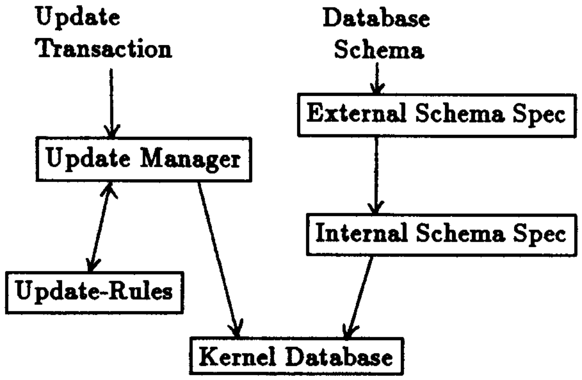

Figure 1: Block Diagram of Schema-Based Approach

To modify the contents of a database, new data is either added to or deleted from the kernel database. Since this prototype is designed to test derived data update functions, only simple update commands are provided. Insertion is done by a command: INSERT (domain, mapping, range). For instance, inserting a fact that P2 is a person with name Mary is done by: INSERT (PERSON, has-instance, P2) and INSERT (P2, Name, 'Mary'). Deletion has a similar form: DELETE (domain, mapping, range). Update (or modification) is considered here as a deletion followed by an insertion with a connection between the two steps. A complete query language for this system, which includes retrieval commands, insertion commands, deletion with or without conditions, and update commands, is currently under development.

The update manager receives update commands from users, and uses an update algorithm, which embodies the update rules, to determine update propagation. The update algorithm is presented immediately below.

4.2 Derived Data Update Algorithm

The algorithm **Update-Propagation(X)** uses update tables to determine update propagation when command X is issued; the command X can be adding instances to or removing instances from a class, or

---

[3] Informally, this means that a school must be attended by some students.

setting or modifying the value of a member or class attribute. Generally speaking, X has format (op, d, m, r), where op is an operation, d is a domain, m is a mapping, and r is a range.

```
Update-Propagation(op, d, m, r):

• Case 1: m is an attribute name.

    1. For each triplet Y = (a, b, c)
       and (b is the inverse of m
          or b is an attribute derived from m
          or m is an attribute derived from b)
            { check the update table for
                derived attributes;
              if (a, b, c) is inserted to or
                deleted from the kernel database
                  do Update-Propagation
                    (op, a, b, c); }
    2. For each category 6 subclass z that is related to m
          { check the update table for
              category 6 subclasses;
            if (r, z, d) is inserted to or
              deleted from the kernel database
                do Update-Propagation
                  (op, r, z, d); }

• Case 2: d is a base class, and m = "has-instance."

  For each y which is a category i subclass of d ($1 \leq i \leq 6$)
    { check the update table for category i
        subclasses;
      if (y, has-instance, r) is inserted to or
        deleted from the kernel database
          do Update-Propagation
            (op1, y, has-instance, r); }
  Whether op1 is insertion or deletion depends
  upon the entries in the update table.

• Case 3: d is a category k subclass ($1 \leq k \leq 6$),
  and m = "has-instance."

    1. For each y which is a superclass of d
          { check the update table for category k
              subclasses;
            if (y, has-instance, r) is inserted to or
              deleted from the kernel database
                do Update-Propagation
                  (op1, y, has-instance, r); }
       Whether op1 is insertion or deletion depends
       upon the entries in the update table.
    2. For each z which is a category i subclass of
       d ($1 \leq i \leq 6$)
          { check the update table for
              category i subclasses;
            if (z, has-instance, r) is inserted to or
              deleted from the kernel database
                do Update-Propagation
                  (op1, z, has-instance, r); }
       Whether op1 is insertion or deletion depends
       upon the entries in the update table.
    3. If d is a category 6 subclass
       and A is a related attribute of d
          { check the update table for
              derived attributes;
            if (r, A, d) is inserted to or
              deleted from the kernel database
                do Update-Propagation
                  (op, r, A, d); }
```

4.2.1 Derived Attributes

Let A be an attribute, derived or not.

	Insert x to	Delete x from
A	Apply **One-to-One Onto** and **Changeable-1**; ambiguous, if not applicable.	Apply **One-to-One Onto** and **Changeable-1**; ambiguous, if not applicable.

4.2.2 Subclass Based on Attribute Predicates (Category 1)

Let T_c be a subclass of T_p based on an attribute predicate P.

	Insert x to	Delete x from
T_p	Insert x to T_c if P is satisfied.	Delete x from T_c.
T_c	Insert x to T_p if P is satisfied; Error if P is not satisfied; If P is with format A = V, then apply **Attr-Pred-Based-2** otherwise, ambiguous.	Apply **Changeable-2**. Ambiguous, if not applicable.

4.2.3 Specified Subclasses (Category 2)

Let T_c be a subclass of T_p with derivation "specified."

	Insert x to	Delete x from
T_p	No action (default).	Delete x from T_c.
T_c	Insert x to T_p.	No action (default).

4.2.4 Intersection Classes (Category 3)

Let T_c be a subclass with derivation "is in T_a and is in T_b."

	Insert x to	Delete x from
T_a	Insert x to T_c, if x is in T_b; no action, otherwise.	Delete x from T_c.
T_b	Insert x to T_c, if x is in T_a; no action, otherwise.	Delete x from T_c.
T_c	Insert x to T_a and T_b.	Apply **Min-Size**. Ambiguous, if not applicable.

4.2.5 Difference Classes (Category 4)

Let T_c be a subclass of T_p but not in T_a.

	Insert x to	Delete x from
T_p	Insert x to T_a, if derivation of T_a is satisfied; insert x to T_c, if derivation of T_a is not satisfied; insert x to T_c (default), if derivation of T_a cannot be evaluated.	Delete x from T_a and T_c.
T_a	Insert x to T_p, if x is not in T_p, T_c; delete x from T_c, otherwise.	Insert x to T_c, if the deletion is legal and x is still an instance of T_p.
T_c	Insert x to T_p, if x is not in T_a; error otherwise.	Delete x from T_p (default).

4.2.6 Union Class (Category 5)

Let T_c be a class with derivation "is in T_a or is in T_b."

	Insert x to	Delete x from
T_a	Insert x to T_c.	No action if x is in T_b; delete x from T_c, otherwise.
T_b	Insert x to T_c.	No action if x is in T_a; delete x from T_c, otherwise.
T_c	Check rules **Value-Class**, **Max-Size**, **Nullable**, **Exhausts-Value-Class**; ambiguous, if not applicable.	Delete x from T_a, T_b.

4.2.7 Attribute-Value-Based Subclasses (Category 6)

Let T_c be a subclass which is derived by: "is a value of Attr of class T_p."

	Insert x to	Delete x from
T_p	Insert corresponding instance to T_c.	Delete corresponding instances from T_c.
T_c	Insert corresponding instances to Attr of T_p.	Delete all instances in Attr of T_p corresponding to x.

4.3 Examples

We present in this subsection two illustrative working examples. For each example there is an initial database state, an update transaction, and a description of update propagation.

4.3.1 Example One

Consider the database transaction that inserts a graduate student P6 with name 'Tim' and student-id 789 to the database. This graduate student is taking the courses CS1 and EE1. He has an annual income of $18,000.

- **Initial Database (triplets):**
  (PERSON, has-instance, P1)
  (P1, Name, 'David')
  (COURSE, has-instance, CS1)
  (COURSE, has-instance, EE1)
  (CS1, Course-number, 123)
  (EE1, Course-number, 101)
  (COURSE, Total-courses, 2)

- **Transaction:**

  INSERT (STUDENT, has-instance, P6)
  INSERT (P6, Name, 'Tim')
  INSERT (P6, Student-id, 789)
  INSERT (P6, Takes, CS1)
  INSERT (P6, Takes, EE1)
  INSERT (P6, Level, 'Graduate')
  INSERT (P6, Annual-income, 18000)

- **Update Propagation:**

  INSERT (STUDENT, has-instance, P6) will insert the triplet (STUDENT, has-instance, P6) to the kernel database, and call **Update-Propagation** (insert, Student, has-instance, P6). Since STUDENT is a category 2 subclass of PERSON, the update table for category 2 subclasses will be referenced, and a new instance (PERSON, has-instance, P6) is generated. This new instance will result in the invocation of **Update-Propagation** (insert, PERSON, has-instance, P6); however, no changes are made as a result. STUDENT has a category 1 subclass GRADUATE; therefore, the update table for category 1 subclasses is also referenced, and a new instance (GRADUATE, has-instance, P6) is added to the database. **Update-Propagation**(insert, GRADUATE, has-instance, P6) is called, but no changes are made as a result.

  The command INSERT(P6, Takes, CS1) inserts an instance (P6, Takes, CS1) to the database, and calls **Update-Propagation** (insert, P6, Takes, CS1). The update table for derived attributes is checked, and (CS1, Students-currently-enrolled, P6) is generated because *Students-currently-enrolled* is the inverse of *Takes*. (EE1, Students-currently-enrolled, P6) is generated for similar reasons. (P6, Monthly-income, 1500) is generated from (P6, Annual-income, 18000).

4.3.2 Example Two

Now consider the database transaction which inserts a new teaching assistant into the database. This new teaching assistant has name 'John' and student-id 123. He took CS1 before, and is taking CS2 and EE1 now. He is in charge of a course CS1.

- **Initial Database (triplets):**

  (PERSON, has-instance, P1)
  (P1, Name, 'David')
  (COURSE, has-instance, CS1)
  (COURSE, has-instance, CS2)
  (COURSE, has-instance, EE1)
  (CS1, Course-number, 123)
  (CS2, Course-number, 222)
  (EE1, Course-number, 101)
  (COURSE, Total-courses, 3)

- **Transaction:**

  INSERT (TEACHING-ASSISTANT,
  has-instance, P2)
  INSERT (P2, Name, 'John')
  INSERT (P2, Student-id, 123)
  INSERT (P2, Took, CS1)
  INSERT (P2, Takes, CS2)
  INSERT (P2, Takes, EE1)
  INSERT (P2, Is-in-charge-of, CS1)

- **Update Propagation:**

  The command INSERT (TEACHING-ASSISTANT, has-instance, P2) adds the triplet (TEACHING-ASSISTANT, has-instance, P2) to the database, and calls **Update-Propagation**(insert, TEACHING-ASSISTANT, has-instance, P2). Since TEACHING-ASSISTANT is a category 2 subclass of GRADUATE, the update table for category 2 subclasses is referenced, and the triplets (GRADUATE, has-instance, P2) and (P2, Level, 'Graduate') are generated. **Update-Propagation**(GRADUATE, has-instance, P2) will be executed, and the triplet (STUDENT, has-instance, P2) is generated. This new data in turn generates (PERSON, has-instance, P2).

  STUDENT has a category 3 subclass BOTH-TA-RA and a category 5 subclass STUDENT-EMPLOYEE; therefore, update tables for category 3 and 5 subclasses are checked, and (STUDENT-EMPLOYEE, has-instance, P2) is generated.

  (CS2, Students-currently-enrolled, P6) and (EE1, Students-currently-enrolled, P6) are generated from (P6, Takes, CS2) and (P6, Takes, EE1) respectively, because *Students-currently-enrolled* is the inverse of *Takes*. (CS1, Has-ta, P2) is the inverse of (P2, Is-in-charge-of, CS1). (CS1, COURSES-TAKEN, P2) is generated from (P2, Took, CS1), because class COURSES-TAKEN is derived by "is a value of Took of STUDENT."

5 Conclusions

In this paper, a schema-based approach to derived data update for semantic databases has been proposed. By contrast with view update in a relational database environment, this approach utilizes information in a semantic database schema, such as superclass and subclass relationships, value classes of attributes, inverses, etc., to propagate the effects of a modification of derived data. Some derived data updates which are ambiguous in a relational view update mechanism can be decided in this approach, e.g., insertion to class which is the union of two other classes. While the approach presented here is based upon the GSDM, it can be applied to other semantic database models as well.

An experimental prototype system based upon the approach described in this paper has been designed, implemented, and tested. It is important to note however that there are some cases in which derived data update operations have unique propagations but which our approach cannot detect. These difficulties result from complex derivation rules, particularly for derived attributes and category 1 subclasses.

Further research planned in this area includes: (1) extending the update rules for category 1 subclasses and for derived attributes, (2) completion of the experimental implementation of a query language and end-user interface for the prototype system, and (3) studying new approaches other than the schema-based to derived data update.

In particular, a second approach to derived data update termed the *rule-based approach* has also been studied and implemented in our research. This rule-based approach encodes all the schema-information into production rules, and then applies a rule inference mechanism to determine derived data update propagation. At present, we are conducting an analytic comparison of the schema-based and the rule-based approaches; this comparison includes the different meta-data roles and organizations in these two approaches, and their different mechanisms to determine derived data update. The study should provide further insight into the semantics of meta-data modification.

Acknowledgements

The authors would like to thank Richard Hull for his many comments and observations on the approach to derived data update described in this paper. The comments of the referees are also gratefully acknowledged.

References

[1] S. Abiteboul and R. Hull, *IFO : A Formal Semantic Database Model*, ACM Trans. on Database Systems, vol.12, no.4, Dec 1987

[2] H. Afsarmanesh and D. McLeod, *The 3DIS: An Extensible, Object-Oriented Information Management Environment*, ACM Trans. on Office Information Systems, 1989 (to appear)

[3] A. Albano, L. Cardelli and R. Orsini, *Galileo : A Strongly-Typed, Interactive Conceptual Language*, ACM Trans. on Database Systems, vol.10, no.2, June 1985, pp. 230-260

[4] F. Bancilhon and N. Spyratos, *Update Semantics of Relational Views*, ACM Trans. on Database Systems, vol.6, no.4, December 1981, pp. 557-575

[5] U. Dayal and P. A. Bernstein, *On the Correct Translation of Update Operations on Relational Views*, ACM Trans. on Database Systems, vol.8, no.3, September 1982, pp. 381-416

[6] A. L. Furtado, K. C. Sevcik and C. S. Dos Santos, *Permitting Updates Through Views of Data Bases*, Inform. Systems, vol.4, 1979, pp. 269-283

[7] G. Gottlob, P. Paolini, and R. Zicari, *Properties and Update Semantics of Consistent Views*, ACM Trans. on Database Systems, vol.13, no.4, December 1988, pp. 486-524

[8] P. Griffiths and B. Wade, *An Authorization Mechanism for a Relational Database System*, ACM Trans. on Database Systems, vol.1, no.3, September 1976, pp. 242-255

[9] M. Hammer and D. McLeod, *Database Description with SDM: A Semantic Database Model*, ACM Trans. on Database Systems, vol.6, no.3, September 1981, pp. 351-386

[10] S. Hudson and R. King, *An Adoptive Derived Data Manager for Distributed Databases*, Advances in Object-Oriented Database Systems, editor K.R. Dittrich, Springer-Verlag, 1988

[11] R. Hull and R. King, *Semantic Database Modeling: Survey, Applications, and Research Issues*, ACM Computing Surveys, vol.19, no.3, September 1987, pp. 201-260

[12] A. M. Keller, *The Role of Semantics in Translating View Updates*, IEEE Computer, January 1986, pp. 63-73

[13] A. M. Keller, *Choosing a View Update Translator by Dialog at View Definition Time*, Proc. Inter. Conf. on Very Large Data Bases, 1986, pp. 467-474

[14] J. Mylopoulos, P. A. Bernstein and H. K. T. Wong, *A Language Facility for Designing Database-Intensive Applications*, ACM Trans. on Database Systems, vol.5, no.2, June 1980, pp. 185-207

[15] D. Shipman, *The Functional Data Model and the Data Language DAPLEX*, ACM Trans. on Database Systems, vol.6, no.1, 1981, pp. 140-173

[16] A. Sheth, J. Larson, and E. Walkins, *Tailor: A Tool for Updating Views*, Proc. of Inter. Conf. on Extending Data Base Technology, Venice, Italy, March 1988

Appendix: (Example Database Schema)

```
PERSON
  identifiers :  Name
  member attributes :
    Name
      value class : STRINGS
      may not be null
      not changeable
    Age
      value class : INTEGERS
    Annual_income
      value class : REALS
    Monthly_income
      value class : REALS
      derivation : = Annual_income / 12

PROFESSOR
  interclass connection :
      subclass of PERSON where specified
  member attributes :
    Teaches
      value class : COURSE
      inverse : Taught_by

STUDENT
  interclass connection :
      subclass of PERSON where specified
  member attributes :
    Student_id
      value class : INTEGERS
    Major
      value class : STRINGS
    Level
      value class : STRINGS
    Takes
      value class : COURSE
      inverse : Students_currently_enrolled
      multi-valued with size between 2 and 5
    Took
      value class : COURSE
      multi-valued

COMPUTER_MAJOR_STUDENT
  interclass connection : subclass of STUDENT
        where Major = 'CS' or Major = 'CE'

GRADUATE
  interclass connection : subclass of STUDENT
        where Level = 'Graduate'

UNDERGRADUATE
  interclass connection : subclass of STUDENT
        where is not in GRADUATE

TEACHING_ASSISTANT
  interclass connection :
      subclass of GRADUATE where specified
  cardinality : with size between 0 and 25
  member attributes :
    Is_in_charge_of
      value class : COURSES_TAKEN
      inverse : Has_ta

RESEARCH_ASSISTANT
  interclass connection :
      subclass of GRADUATE where specified

STUDENT_EMPLOYEE
  interclass connection :
        subclass of GRADUATE
        where is in TEACHING_ASSISTANT
        or is in RESEARCH_ASSISTANT
  member attributes :
    Tuition_coverage
      value class : REALS
    Stipend
      value class : REALS
    Benefit
      value class : REALS
      derivation : = Tuition_coverage
                     + Stipend

BOTH_TA_RA
  interclass connection :
        subclass of GRADUATE
        where is in TEACHING_ASSISTANT
        and is in RESEARCH_ASSISTANT

COURSE
  identifiers :  Course_number
  member attributes :
    Course_number
      value class : INTEGERS
      single-valued
      may not be null
    Taught_by
      value class : PROFESSOR
      inverse : Teaches
    Has_ta
      value class : TEACHING_ASSISTANT
      inverse : Is_in_charge_of
    Students_currently_enrolled
      value class : STUDENT
      inverse : Takes
      multi-valued with size
                between 0 and 50
  class attributes :
    Total_courses
      value class : INTEGERS
      derivation : number of members
                   in this class

COURSES_TAKEN
  interclass connection :
        subclass of COURSE where is
        a value of Took of STUDENT
```