Hypertext view update problem

Gilles Falquet, Luka Nerima, Seongbin Park
Centre Universitaire d'Informatique
University of Geneva
24, Rue General-Dufour
CH-1211 Geneve 4, Switzerland
{Gilles.Falquet, Luka.Nerima, Seongbin.Park}@cui.unige.ch

March 8, 2000

Abstract

A hypertext view is a derived hypertext for a given database. We are interested in providing a mechanism of translating hypertext view update operations to appropriate data manipulation operations in database. However, given a hypertext view update operation, we cannot always uniquely determine what needs to be done in terms of database operations to perform the intended hypertext view update operation. Therefore, the goal of this study is to identify *ambiguities* that arise in the *translation* mechanism and investigate how *link information* (i.e., static link information such as links in hypertext and dynamic information such as traversals) can be used in resolving the ambiguities. We provide update semantics for each of the hypertext view update operations and show how the ambiguities can be resolved with link information.

1 Introduction

Given a database and a derived hypertext from the database (i.e., hypertext view), we would like to provide a translation mechanism of hypertext view update operations into database operations. We call this problem the *hypertext view update problem.* Providing hypertext views with users is helpful in various ways; navigating among the set of hypertext nodes is easier than writing SQL queries and searching by browsing can be done for searching database contents. Often times, users that browse through database contents want to have capabilities of editing the contents that appear in the browser. Having a system that supports hypertext interfaces could enhance applications for the Web as well. However, it is not always unique what to do in database since there exist ambiguities. On the other hand, in the hypertext environment, users navigate through the hypertext nodes and we can exploit the link history information in order to resolve possible ambiguities.

A hypertext view is a (virtual) hypertext derived from the database relations or classes. From a hypertext point of view, a hypertext view is nothing else than a hypertext (i.e., nodes and links) which is derived from a database instance (set of object collections). From a database point of view, a hypertext view is a set of database views together with a linking structure that interconnects these views or elements of these views. To be precise, they are not exactly views, but views which have been transformed in order to become hypertext nodes (or, pages). For instance, a hypertext view on a relational database is not a set of relational views (i.e., derived relations) but a set of HTML (or, XML, etc) pages derived from relational views. Similarly, hypertext views on an object-oriented database are not a set of imaginary/virtual classes/collections but a set of HTML or XML pages obtained from real or, virtual classes. So, some transformation is involved in order to make a true hypertext content from a database view. (See figure 1.)

The database view update problem has been studied extensively [B92, M84, SLT91], and the concept of a hypertext view on a database was defined differently [S98, FGN98]. However, to our knowledge, this is the first attempt to solve the hypertext view update problem.

2 Models

2.1 Database Model

We consider a basic object-oriented model which can be seen as a fragment of the O_2 model [LRV]. Each object has an identity (oid) and a value. The values of objects are tuples $[name_1 : value_1, \cdots, name_n : value_n]$, where

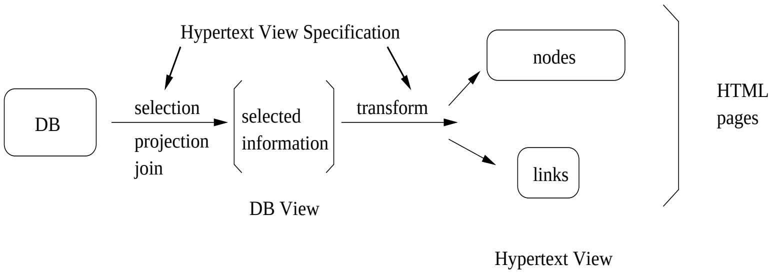


Figure 1: Database, view, and hypertext view. Notice that what appears in a browser (i.e., HTML page) is a *portion* of a hypertext view. Database schema and hypertext view specification determine each hypertext view construction. Hypertext view specification consists of the set of node schemes that specify the collection from which the node's content is to be drawn; the selection and ordering criteria; the elements that form the content; and the links to other nodes [FGN98].

each $name_i$ is an attribute name and each $value_i$ is an atomic value (of type integer, boolean, float, string), or a reference to another object, or a collection of references (set, list, bag). The type of a class constrains the values of its objects. A database instance is a set of objects and a set of named collections. Each object is an instance of a class and its value belongs to its class type. A named collection is a set (or list) of objects which are instances of a class or of one of its subclasses. There is a special collection called *clipboard* which is of type OBJECT. A database schema is a set of class and collection definitions and we consider the following operations: Insert(object, collection), Delete(object, collection), Update(object, attribute, value).

2.2 Hypertext Model

In the hypertext model, each node has a unique identity and the content of a node is a sequence of elements (atomic or composite). A sequence of elements that represents a database object is called an item. An anchor is an element or a sequence of elements within the content of a node and it serves as a starting or ending point of a link. A link is defined by its starting and ending anchors and by its category which is either *reference* or *inclusion*. Hypertext view update operations that we consider are as follows:

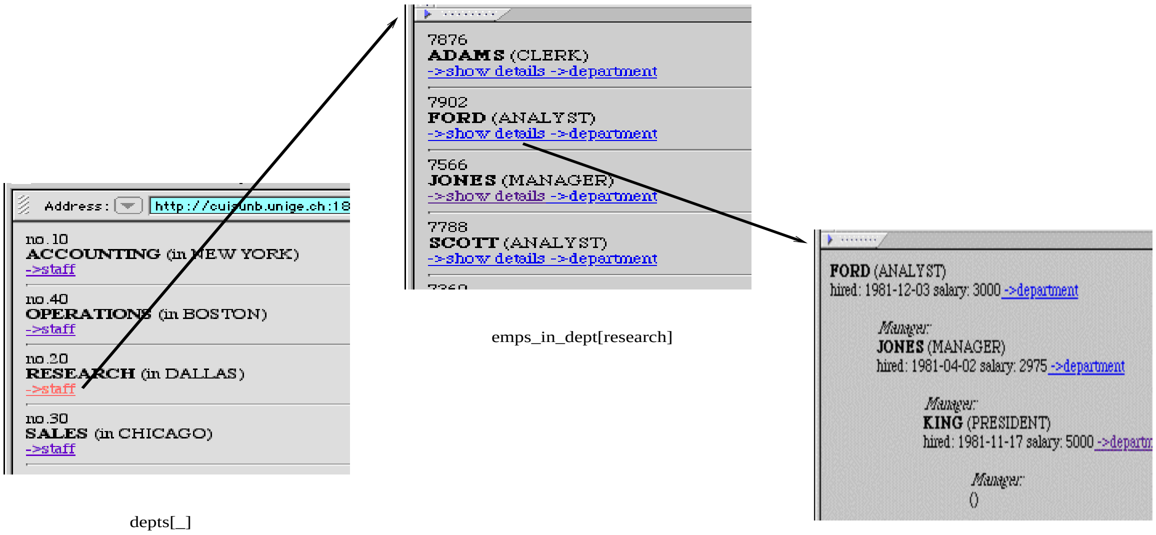
- Cut (item, node) – delete an item from the specified node and insert it in the clipboard[1]
- Copy (item, node1, node2, loc) – insert an item of a node1 into a node2 at the location
- Paste (item, node, loc) – insert an item of the clipboard into the specified node at the location
- Insert (item, node, loc) – insert an item into the specified node at the location
- Delete (item, node) – delete an item from the specified node
- Update (item, node) – change the value of the item of the specified node
- Change starting/ending anchor – i.e., change the destination of a link to another (or, the source of a link to another.)
- New/Create nodes

3 Hypertext view update problem

Given a database schema, hypertext view specification, and a hypertext view update operation,

- provide a *translation* mechanism that executes the intended operation in the database level and express the operation in terms of hypertext model,
- classify ambiguities that arise in the translation mechanism (i.e., identify truly ambiguous cases that designers need to provide specific information in order to resolve them.), and
- specify how link information can be used in resolving the ambiguities.

[1] putting an object in the clipboard does not necessarily mean to remove it from its collection.



depts[_]   emps_in_dept[research]   emp_with_mgr[ford]

Figure 2: Node instances for the example.

3.1 An illustrative example

Let's consider a simple database whose class definitions are given as follows:
class Emp (empno : int, ename : String, job : String, hiredate : Date, sal : int, mgr : Emp, department : Dept)
class Dept (deptno : int, dname : String, loc : String)
collections EMPS : set(Emp), DEPTS : set(Dept)
The node schema definitions are as follows[2]:

```
node depts[d :  set(Dept)]
     node_style ''list''
     items
     ''no.'', deptno, <break>
     <name> dname, '' (in '', loc, '') '',
     href emps_in_dept [self] " → staff"
     from DEPTS
     selected by self in d
     order by dname
```

```
node emps_in_dept [ d :  Dept ]
     node_style ''list''
     items
     empno,
     <name> ename , ''('', job, '')'',
     href emp_with_mgr[self] ''→ show details'',
     href depts [ department ] '' → department''
     from EMPS
     selected by department = d
     order by ename
```

```
node emp_with_mgr[e:  Emp]
     items
     <name> ename, ···,
     <indented-subnode> (
     ...
     include emp_with_mgr[mgr]
     )
     from EMPS
     selected by self = e
```

Suppose that we want to cut the item, "7566 JONES (MANAGER) ···", from the node instance emps_in_dept[research] in figure 2, we need to do the following:

- Check related node schema(s). (i.e., emps_in_dept or depts). This involves checking to which collections the corresponding object belongs, the selection criteria that can tell us which attribute is associated, and the parameter values which have the information about link.

[2]Each elements can be typed by prefixing it with a type identifier between < and >. A transformation function is associated to each type. For instance, if the target hypertext/language system is HTML, the function associated with <name> will surround its argument with and . The association between element types and transformation functions defines a style. Dissociating the style from the node schemas enables to target several hypertext systems for the same hypertext view definition.

- The intended semantics of this operation at the hypertext level is that next time the node will be computed (for instance, by activating the link, href emps_in_dept[research], or by reloading the page in the browser), this item should not appear.
- At the database level, we have some choices:
  1. Explicitly delete the object that corresponds to the item in the database – but, there could be situations where we need that object in another node; e.g., other attributes of another node schema of the object that were not used can be used in a different predicate (this is *more* than what we need.) Suppose that *obj* is the object that corresponds to the item in the database. Then, the deletion could mean the execution, Delete(*obj*, EMPS).
  2. Change the attribute value – to what? (**the problem**) For example, updating the department attribute of this object to something that is different from the current department, such as *obj.department* := *accounting*.

In this example the two possible ways to translate the hypertext operation into database operations affect only the base collection of the current node. This is similar to the view update problem in database views. We will see in the next section that, in general, we must also take into account the navigation path that has been followed to reach the current node. In fact, updating some objects that have been visited during this navigation may have an effect on the current node.

3.2 Local and dynamic (navigation) semantics of nodes

A node in a hypertext view always represents a selected set of objects that belongs to a database collection. The local semantics of a node N with parameters $p_1, \cdots, p_k$ is given by its selection expression $S[p_1, \cdots, p_k, self]$ which is a first order logic formula with free variables $p_1, \cdots, p_k$ (the node parameters) and self (self represents an object of the base collection.) [FGN98]. According to this semantics, the contents of a node instance $N[a_1, \cdots, a_k]$ is the set $\{x \in \text{base collection} \mid S[p_1 = a_1, \cdots, p_k = a_k, self = x] = true\}$. Thus, in this local perspective, the only way to change the contents of $N[a_1, \cdots, a_k]$ is to insert, delete or update objects in the base collection of N. However, there is another perspective, which consists in looking at the way parameter values were obtained. These values are fixed when traversing the link that lead to N from the previous node, say $M[q_1, \cdots, q_r]$. Suppose that this link has the form, href $N[e_1, ..., e_k]$. Its source anchor is within an item of M and the expressions $e_1, \cdots, e_k$ depend on (1) the parameters $q_1, \cdots, q_r$ of M (2) (the attribute values of) the object o, represented by this item (in M). Thus the selection formula of M can be rewritten by substituting $e_i(q_1, \cdots, q_r, o)$ for each $p_i (1 \leq i \leq k)$ in S. Now S is an expression with free variables $q_1, \cdots, q_r, o$, and self. So, from a navigational perspective, we can say that, when coming from M, the content of N depends on an object represented in M and the parameters $q_1, \cdots, q_r$ of M. Since the parameters $q_1, \cdots, q_r$ of M may themselves be derived from the parameters and objects of another linked node, the selection formula of N can be further extended by traversing the navigation path backwards.
Example: Consider the following database schema and node schemas :
class Employee (empno: int, ename: String, job: String)
class Department (deptno: int, dname: String, location: String, staff: set(Employee))
class Team (tname: String, sport: String, members: set(Employee))
collections EMPS : set(Employee), DEPTS : set(Department), TEAMS : set(Team)

```
node emps [e :  set(Employee)]
    ...
    from EMPS
    selected by self in e
```

```
node departments_in[loc]
    ...
    href emps[staff] // staff is a set valued attribute
    ...
    from DEPTS
    selected by location = loc
```

The local semantics of emps[e] is *self in EMPS* and *self in e*. If we substitute *o.staff* for *e* in this formula, we obtain *self in EMPS* and *self in o.staff*. This formula shows that the contents of *emps* depends on the *staff* attribute of an object *o* of *DEPTS*. Since the parameter *loc* of *departments_in* does not appear in the actual parameter list of the link to *emps*, we are sure that the contents of *emps* does not depend on other objects of the database. Had we come to *emps* from a node

```
node team[except :  set(Employee)]
    ...
    href emps[members − except]
    from TEAMS
    ...
```

the navigational semantics of *emps* would have been *self in EMPS* and *self in o.members − except*. In this case it depends on the attribute members of an object *o* of *TEAMS* and on the parameter *except*. This parameter's value, in turn, could depend on another object of the database. The navigational semantics shows that the ambiguity problem is not limited to the local selection expression of a node but that it extends backwards along the navigation path. This extension stops whenever the parameters of a node do not occur as actual parameter in the link expression. This can be checked statically by examining the definitions of each node. Thus, in order to change the content of *emps*, we can either modify *EMPS* or modify *o.members*. The dynamic semantics extends backwards as long as the node formal parameters are passed/used in the node reference; i.e., the node schema is of the form:

```
node N [···, p, ···]
      ...
      href M [···, p, ···]
from
```

So, the dynamics semantics is completely determined when we reach a node that does not transmit its parameters forward to referenced nodes. There are potentially several dynamic semantics since there can exist several paths to reach a node. The important point is that these semantics can be found by statically examining the node definitions. The dynamic semantics may extend infinitely if there are circuits (loops) with parameter transmissions as is

```
node NR [···, p, q, ···]
      ...
      href NR [···, s, q, ···]
```

here, the parameter, q is passed along the cyclic navigation in the NR nodes. However, this infinite navigational path can be represented by regular expressions and only object from (before the loop) can be transmitted outside of the loop.

4 Our proposed approach

We have shown in the previous section that there are basically two sources of ambiguity when updating node contents:

- there may exist several navigation paths to reach a node
- each navigation path determines a dynamic semantics which is a first order expression possibly involving several database objects. So there are potentially several possibilities to modify the truth value of this expression, and hence the content of the node.

Thus, for each update operation (Cut, Paste, Insert, etc.), and for each possible navigation path, it is necessary to fix the actual database action (Insert, Delete, Update).
To specify these actions we add to each node schema the following statement
on <operation>
    when path ends with <navigation path> do <database action>
    when path ends with ···
The navigation path is a sequence of node instances, selected item/object, and selected link. It take the following form:
<node name> [<parameter variables>] (<item variable>) → ···
The item variable represents the starting item of the traversed link. Since an item represents a database object, this variable can be used in the definition of the database action.
Example : In the node schema emps[e : set(Employee)] of the previous example we can add the following operation definitions:
on cut
    when departments_in[lc](o) → do
        o.staff := o.staff - self
    when n[...](o1) → team[ex](o2) → do
        o2.members := o2.members - self; o1.on_leave := o1.on_leave + self
If the hypertext structure contains cycles it may happen that there are infinitely many possible navigation path that reach a particular node. However, it has been shown that these paths constitute a regular set, so they may be represented by a regular expression. In case of cycle, the <navigation path> of the when clause may thus be a regular expression.

4.1 Implementing the hypertext operations

In what follows we show all the possible ways to implement the hypertext operations on a hypertext view.
Cut: There are basically two way to cut an item of a node.

- Remove the corresponding object from the base collection of the node.

- Update the value of this object and/or the value of one or several objects that appear in the dynamic semantics formula of this node. Note that it is not always possible to update the object values so as to make the selection formula false.

So we can say that there are three levels of implementation :

- change the base collection (Delete)
- change the corresponding object's value only (local)
- change other object values (navigation)

In any case, the object must be inserted into the clipboard collection, it will thus appear in the clipboard node which shows this collection.
Paste: Let o be the object of the clipboard that we want to paste in node N. If o does not belong to the base collection B of N it must first be inserted into B. Then either the value of o and/or the value of one or several objects that appear in the dynamic semantics formula must be changed. Once again, this operation may fail if it is impossible to find object values that make the selection formula true.
Insert(new_item): Inserting a new item consists in

- creating a new database object and setting its attribute values according to the give item.
- pasting that object in the node (see above)

The difference with pasting lies in the fact that the new object value may not be change (in fact, a user does not expect the system to modify what he or she just entered).
Update(element of an item): Since an item correspond to a database object, updating an item element amounts to update one or several attributes of that object. Ambiguities may arise if several attributes are involed in the element specification. If the element specification depends on a node parameter, it may be necessary to update an object represented in another node.

5 Conclusion and future directions

We have presented an approach to the view update problem in hypertext views on databases. We identified ambiguity patterns that depend on hypertext navigation, and thus differs from the well-known database view update problem. The proposed approach to solve these ambiguities is based on the navigation semantics of nodes, which takes into account the navigation path that lead to a particular node.
We are currently implementing a system for the specification and dynamic generation of updatable hypertext views. This system comprises

- a node compiler that translate textual node schemas into node definition objects. These definitions are then stored in the database dictionary.
- a node server that generate node contents in response to node requests, keeps tracks of the navigation history, and translates hypertext operations in database operations according to the navigation history and node specifications

There are numerous modeling opportunities for such an approach in various fields, e.g. the representation and editing of complex electronic documents stored in databases, database applications to be accessed through the Web, shopping basket like applications for electronic commerce.

6 References

[B92] A View Mechanism for Object-Oriented Databases, EDBT'92, Elisa Bertino
[FGN98] Languages and Tools to Specify Hypertext Views on Databases, WebDB 1998, Gilles Falquet, Jacques Guyot, Luka Nerima
[LRV] O_2, an Object-Oriented Data Model, ACM SIGMOD 1988, Christophe Lecluse, Philippe Richard, and Fernando Velez
[M84] A relational database view update translation mechanism, VLDB 1984, Yoshifumi Masunaga
[SLT91] Updatable Views in Object-Oriented Databases, Proceedings of the 2nd DOOD, 1991, Marc H. Scholl, Christian Laasch, and Markus Tresch
[S98] Incremental Maintenance of Hypertext Views, WebDB 1998, Giuseppe Sindoni