Updateable Views for an XML Query Language*

Hanna Kozankiewicz[1], Jacek Leszczyłowski[1], Kazimierz Subieta[1,2]

[1] Institute of Computer Science Polish Academy of Sciences, Warsaw, Poland
[2] Polish-Japanese Institute of Information Technology, Warsaw, Poland
{hanka, jacek, subieta}@ipipan.waw.pl

Abstract. We present an approach to updateable XML views, which is based on the Stack-Based Approach to query languages. Its novelty consists in introducing procedures overloading generic updating operations on virtual objects into a view definition. This overloading is implicit and users update virtual objects like stored objects. Due to procedures we support full algorithmic power of view definitions, which is essential for writing sophisticated wrappers and mediators. The approach is also relevant to general object-oriented data models.

1 Introduction

A view is a virtual image of data stored in a database that is adapted to user/application requirements. In Web applications views enable one to resolve incompatibilities among distributed/heterogeneous data resources and to integrate them. Hence the topic raises a lot of interest in the Web and XML communities [1].

In this paper we present a novel approach to updateable views, which is based on the Stack-Based Approach (SBA). We deal only with virtual views. Our view mechanism provides a possibility of inclusion information about view updates intents into a view definition. The information has a form of procedures (written in a query language) which dynamically overload generic view (updating) operations. This feature allows the view definer to construct very powerful views which can take algorithmically complete control over actions on stored objects that are induced by updates of virtual objects. The view definer has also all possibilities to avoid unexpected and/or undesired side-effects during updating virtual objects (overupdating, underupdating or warping updating intention).

The rest of the paper is structured as follows. The next section contains basic concepts of the Stack-Based Approach to query languages. Section 3 presents main issues of our approach to XML updateable views. Section 4 concludes.

2 The Stack-Based Approach

The foundation of the Stack-Based Approach [2] is that query languages can be treated as a particular case of programming languages and their semantics

* Supported by the European Commission 5th Framework project ICONS, IST-2001-32429.

should be defined in a similar manner. The approach is abstract and universal what makes it relevant to a general data model. A basic mechanism of SBA is an environment stack (ES), which is used for binding names occurring in queries according to their scopes. ES consists of sections that are sets of *binders* that relate names to run-time entities. A new section is pushed (popped) on (from) ES in accordance with query nesting. A name appearing in a query is bound via ES to a run-time entity. The search for name binding starts from the ES top and ends on the first visible section that contains a corresponding binder.

2.1 The Stack Based Query Language

The Stack Based Query Language (SBQL) is a formalized query language in a spirit of SQL or OQL. It is based on the principle of compositionality. We can create complex queries through combining them from simpler ones by means of unary/binary operators. The simplest query is an atomic query, which can be any name or any literal (e.g. 5, "Hamlet", book). SBQL queries return *q_results* that can be combined by query operators, considered as algebraic or non-algebraic.

Algebraic operators are for example: +, max, or. If a query $q_1 \ \Delta \ q_2$ contains algebraic operator then the order of evaluation of queries q_1 and q_2 is insignificant. Both these queries are evaluated independently and then their results are combined into a final results according to semantics of operator Δ.

Non-algebraic operators are basic for SBQL. If a query $q_1 \ \theta \ q_2$ is build up of a non-algebraic operator θ then, then q_2 is evaluated in the context determined by q_1. This evaluation looks as follows. First, query q_1 is evaluated and it returns a collection *q_result*. Next, for each element $e \in$ *q_result* ES is augmented with a new section containing local environment of e, then q_2 is evaluated, and the section is removed from the stack. Finally, the partial results of these iterations are combined into a final result according to θ. Non-algebraic operators include selection, dependent join, projection/navigation/path expression (dot), $\forall$, $\exists$, ordering, and transitive closures.

2.2 Functions and Views in SBQL

SBA fully supports functions and procedures. They can be defined with or without parameters, can involve local objects, can have side-effects, and can call other function within their bodies (including recursive calls). There is also no restriction on computational complexity of the functions.

In classical approaches views are essentially functions (e.g. [1]) that return references to (stored) objects. View updates use these references as *l-values*. This approach leads to severe limitations of view definition power and to a danger of violating the user updating intents. In our research we follow a totally different approach in which the power is unlimited and the user intents are explicitly defined inside a view definition. Thus our view is a more complex structure than e.g. an SQL view. It consists of a definition of virtual objects, definitions of all the required operations on virtual objects, and definitions of all nested views (e.g. virtual [sub-]attributes of virtual objects).

3 Updateable XML Views

In this section we sketch the main concepts of our approach that is described in detail in [3]. Our view is not defined by a single query, but consists of two parts. The first part determines some values or references to stored objects that are the basis for building up virtual objects. The second part (re)defines operations that can be performed on the virtual objects.

The first part of the definition is a procedure that returns a set of entities called *seeds* that are used for making up virtual objects. The second part includes procedures that (re)define generic operations (update, delete, dereference, insert) that can be performed on virtual objects identified by seeds. A seed is a parameter of these operations. Let us analyze an example view definition for an XML file containing bibliography. XML root element is denoted by the tag *bib*, and XML subnodes contain information on books (XML tag *book*). Each book node has the subnodes *title* and *author*. The example view is as follows:

```
create view LemBookTitleDef {
  virtual objects LemBookTitle {
    return (bib.book where author = "Lem") as b; }
  on_retrieve do { return capitalize(b.title); }
  on_update new_title do {
    if theUserHasUpdatePermission() then b.title = new_title; }
  on_delete do { if theUserHasUpdatePermission() then delete b; } }
```

The view returns titles of Lem's books in the database. It defines: the dereference operation that returns a capitalized book's title; the protected operation of update that changes the title to a new one (operation has the parameter *new_title*); and the protected operation of deletion that deletes a given book from the database. Note that operation of object insertion is undefined, thus it is disallowed. The view can be called in an example request that fixes a typo in a title:

```
for each LemBookTitle as bt do
  if bt = "SOLRIS" then bt := "Solaris";
```

Note that the request implicitly calls two view operations: *on_retrieve* when it performs the comparison = (which forces dereference of bt), and *on_update* when it performs :=. The example illustrates the important property of views known as *view transparency*, which claims no syntactic and pragmatic differences in operations on virtual and stored objects. In our case the transparency concerns not only querying but also updating operations. The users formulating requests need not to distinguish between stored and virtual data applying the same query syntax to both cases.

3.1 Processing of View Updates

To distinguish virtual objects from stored objects we have introduced the concept of *virtual identifier* – a counterpart of a stored object identifier. Such a virtual

identifier is distinguishable from a stored object identifier and contains the seed of a given virtual object and the identifier of a corresponding view definition.

When a view is invoked in a query it returns a set of virtual identifiers. Next, when a system tries to perform update operation with a virtual identifier as an *l-value*, it recognizes that it deals with the virtual object and hence, it calls the proper update operation from the view definition. To enable that, a virtual identifier must contain both a seed (which is a parameter of view updating procedures passed implicitly via ES) and the identifier of the view definition.

3.2 View Nesting

The presented method allows for view nesting with an unlimited number of nesting levels. Sub-views are seen as (sub) attributes of virtual objects. This feature requires an extension of virtual identifier for sub-views that has to contain information on ancestors (in nesting) of the view because the information can be useful for the view definer to refer to the ancestors during writing view updating procedures.

3.3 Parameters and Recurrence

Views can have parameters that (similarly to function's parameters) can be arbitrary queries. A method of parameter passing is determined according to syntax of the view definition: we provide both *call-by-value* and *call-by-reference*. Our view mechanism also supports recurrence as a side effect of SBA and its stack-oriented semantics.

4 Conclusions

We have presented a new approach to updateable XML views that gives to a view user the possibility to take full control over what happens with data accessible through the view. The approach is consistent and implementable. We have already implemented SBQL with functions for XML native databases based on the DOM model and now implementation of the presented view mechanism is under way. We believe that the presented idea creates great possibilities that have not been even considered by other approaches, thus is worth attention from the XML research community.

References

1. S.Abiteboul. On Views and XML. Proc. of PODS Conf., 1999, 1-9
2. K.Subieta, Y.Kambayashi, J.Leszczyłowski. Procedures in Object-Oriented Query Languages. VLDB 1995: 182-193
3. H.Kozankiewicz, J.Leszczyłowski, J.Płodzień, and K.Subieta. Updateable Object Views. Institute of Computer Science, Polish Academy of Sciences, Report 950, October 2002