

Towards Automatic Generation of Rules for Incremental Maintenance of XML Views of Relational Data

Vânia Vidal¹, Valdiana Araujo¹, and Marco Casanova²

¹ Dept. Computação, Universidade Federal do Ceará,
Fortaleza, CE - Brasil
{vvidal, valdiana}@lia.ufc.br

² Dept. Informática, PUC-Rio
Rio de Janeiro, RJ - Brasil
casanova@inf.puc-rio.br

Abstract. This paper proposes a two-step approach to define rules for maintaining materialized XML views specified over relational databases. The first step concentrates on identifying all paths of the base schema that are relevant to a path of the view, with respect to an update. Hence, the corresponding views paths can be affected by the update. The second step creates rules that maintain all view paths that can be affected by the update. The paper then discusses how to automatically identify all paths in the base schema that are relevant to a view path with respect to a given update operation and how to create the appropriate maintenance rules.

1 Introduction

The eXtended Markup Language (XML) has quickly emerged as the universal format for publishing and exchanging data on the Web. As a result, data sources often export XML views over base data [7, 11]. The exported view can be either virtual or materialized. Materialized views improve query performance and data availability, but they must be updated to reflect changes to the base source.

Basically, there are two strategies for materialized view maintenance. Re-materialization re-computes view data at pre-established times, whereas incremental maintenance [1, 2, 8, 13, 14, 17] periodically modifies part of the view data to reflect updates on local sources. Incremental view maintenance proved to be an effective solution.

Incremental view maintenance algorithms have been extensively studied for relational and object oriented views [3, 9, 12, 15]. Also, there have been incremental maintenance algorithms for semi-structured views [1, 18, 27], and recent work on maintaining XML views [5, 6, 9, 10, 21, 26]. The techniques of [9, 26] cannot be applied to incremental maintenance of XML views of relational data. [5] proposed two approaches for incremental evaluation of ATGs[4] which is the formalism used for specifying the mapping between a relational schema and a predefined DTD. In

their middleware system, an update monitor detects source updates, and triggers their incremental algorithm to propagate the changes to XML views automatically.

In [19], we proposed an incremental maintenance strategy, based on rules, for XML views defined on top of relational data. The strategy has three major steps: (1) convert the base source schema to an XML schema, so that the view schema and the base source schema are expressed in a common data model [22]; (2) generate the set of view correspondence assertions by matching the view schema and the base source XML schema (the set of view correspondence assertions formally specify the mapping [16,25] between the view schema and the base source schema); and (3) derive the incremental view maintenance rules from the correspondence assertions. Our solution has the following salient points in relation to previous work:

- The views can be stored outside of DBMS.
- Most of the work is done at view definition time. Based on the view correspondence assertions, at view definition time, we are able to: (i) identify all source updates that are relevant to the view; and (ii) define the view maintenance statements required to maintain the view w.r.t a given relevant update. It is important to notice that the view maintenance statements are defined based solely on the source update and current source state, and, hence, no access to the materialized view is required. This is important when the view is maintained externally, because accessing a remote data source may be too slow.
- Our rules can be implemented using triggers. In which case, no middleware system is required, since the triggers are responsible for directly propagating the changes to the materialized view.
- The view maintenance statements propagated by the rules do not require any additional queries over data source to maintain the view. This is important in case where the view is maintained externally.

One can see that the use of rules is a very effective solution for incremental maintenance of external views. However, creating the rules that correctly maintain an XML view can be an extremely complex process. So, there is a need for tools which automate the rule generation process. In this paper we show that, based on the view correspondence assertions, one can generate, automatic and efficiently, all the rules required to maintain a materialized view. Our formalism allows us to formally justify that the rules generated by our approach maintain correctly the view.

The main contributions of the paper are threefold. First, we propose the following improvements to our mapping formalism: (i) we formally specify the conditions for a set of correspondence assertions to fully specify the view in terms of the source and, if so, we show that the mappings defined by the view correspondence assertions can be expressed as XQuery view definitions; (ii) we propose an algorithm that, based on the view correspondence assertions, generates XQuery[24] functions that construct view elements from the corresponding tuples of base source; (iii) we introduce a simple graphical interface that helps user define view correspondence assertions. It is important to say that other proposed mapping formalism are either ambiguous [16] or require the user to declare complex logical mapping [25] which can not be graphically defined.

Second, we provide a formal framework to detect the view paths that are affected by a base update, and to define the rules required for maintaining a view with respect to a given base update. As we will show, this can be carried out automatically and efficiently using view correspondence assertions.

The paper is organized as follows. Section 2 reviews how to convert a relational schema into an XML schema. Section 3 presents our mapping formalism. Section 4 discusses how to specify XML view using correspondence assertions. Section 5 presents our approach for automatic generation of incremental view maintenance rules. Section 6 describes how to automatically identify the paths that are relevant to a view path with respect to an insertion, deletion or update on a base table. Finally, Section 8 contains the conclusions.

2 Mapping Relational Schema to XMLS⁺ Schema

We review in this section how to map a relational schema R into an XML schema S , using the mapping rules described in [22].

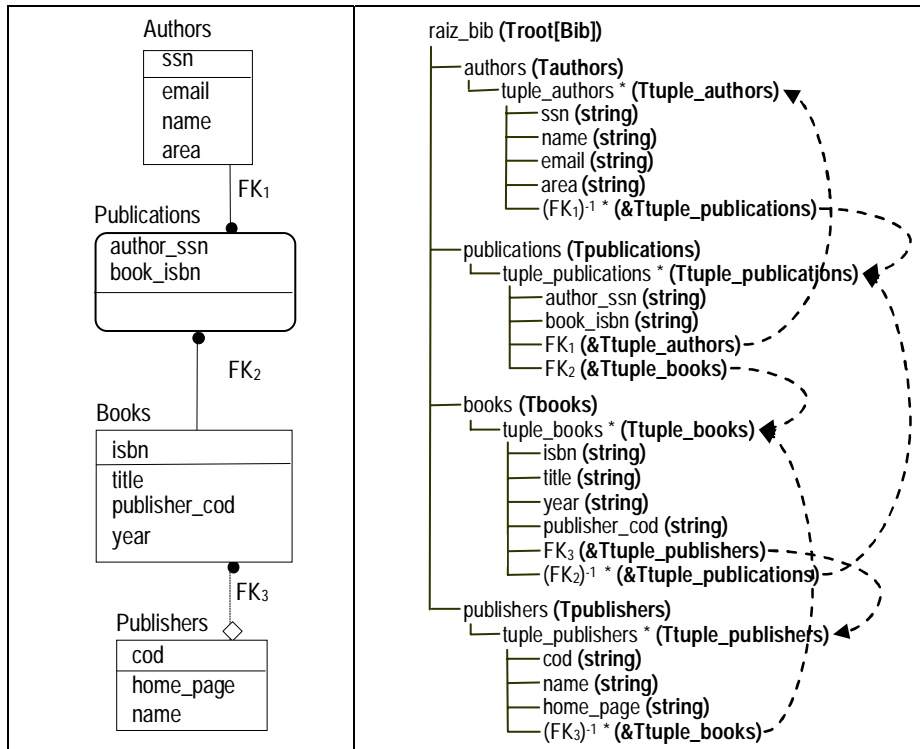


Fig. 1. The relational schema *Bib* and the XMLS⁺ schema *Bib*

Briefly, \mathbf{S} has a root complex type, denoted $\text{Root}[\mathbf{S}]$. For each relation schemes R_i in R , $\text{Root}[\mathbf{S}]$ contains an element R_i of type T_{R_i} . Note that the relation scheme and the XML element have the same name and will be used interchangeably. The complex type T_{R_i} contains a sequence of elements $\text{tuple_}R_i$ of type $T_{\text{tuple_}R_i}$. For each attribute a of R_i of type t_a , the complex type $T_{\text{tuple_}R_i}$ contains an element a of a built-in XML data type compatible with t_a . Finally, for each referential integrity constraint in R , the XML schema \mathbf{S} contains a *key ref* constraint.

The mapping rules guarantee that the relational schema R and the XML schema \mathbf{S} are semantically equivalent, in the sense that each database state of R corresponds to a XML document with schema \mathbf{S} , and vice-versa.

We adopt a graphical notation, denoted XMLS^+ for *Semantic XML Schema* [22], to represent the types of the XML schema \mathbf{S} . Briefly, the notation represents the types of \mathbf{S} as a tree, where type names are in boldface, “&” denotes references, “@” denotes attributes and “*” denotes multiple occurrences of an element.

Fig. 1 shows graphical representations for the relational schema *Bib* and for the XML schema **Bib**. The root type $\text{Root}[\mathbf{Bib}]$ contains an element for each relation in *Bib*. For example, the element *authors*, of type T_{authors} , corresponds to the relation scheme *authors*. T_{authors} contains a sequence of zero or more *tuple_authors* elements of type $T_{\text{tuple_authors}}$, which contains elements *ssn*, *name*, *email* and *area*. The constraint FK_3 of *Bib* is represented by two reference elements in **Bib**: *tuple_books* has a reference element, also labeled FK_3 , with type $\&T_{\text{tuple_publishers}}$ (a reference to $T_{\text{tuple_publishers}}$); and $T_{\text{tuple_publishers}}$ contains a reference element, labeled FK_3^{-1} , with type $\&T_{\text{tuple_books}}$ (a reference to $T_{\text{tuple_books}}$).

Let T_1, \dots, T_n be types of an XML schema. Suppose that T_k contains an element e_k of type T_{k+1} , for $k=1, \dots, n-1$. Then $\delta = e_1 / e_2 / \dots / e_{n-1}$ is a path of T_1 , and T_n is the type of δ . If e_k is mono occurrence, for $k=1, \dots, n-1$, then δ is mono occurrence; otherwise, δ is multi-occurrence. For example, $\delta = \text{books} / \text{tuple_books} / \text{FK}_3$ is a multi-occurrence path of $\text{Root}[\mathbf{Bib}]$.

In what follows, we adopt an extension of XPath [23] that permits navigating through reference elements of XMLS schemas. For example, let σ be a XML document whose schema is **Bib**. The extended path expression $\text{document}(\sigma) / \text{books} / \text{tuple_books} / \text{FK}_3$ is defined by the following XQuery expression: { **For** $\$b$ **in** $\text{document}(\sigma) / \text{Books} / \text{tuple_books}$, **For** $\$p$ **in** $\text{document}(\sigma) / \text{publishers} / \text{tuple_publishers}$ **where** $\$b / \text{publisher_cod} = \p / cod **Return** $\$p$ }.

3 Background

In this section, consider \mathbf{V} and \mathbf{S} XMLS schemas, T_v and T_s types of \mathbf{V} and \mathbf{S} , respectively.

Definition 3.1: A *path correspondence assertion (PCA)* of \mathbf{V} is an expression of the form $[T_v / e] \equiv [T_s / \delta]$, where e is an element of T_v and δ is a possibly null path of type T_s .

Definition 3.2: Let T_v and T_s be types, and P be a set of Path Correspondence Assertions (PCA). We say that P specifies T_v in terms of T_s iff, for any element e in T_v , there is a PCA of the form $[T_v / e] \equiv [T_s / \delta]$ in P , such that:

- (i) If e is mono occurrence, then δ is mono occurrence.
- (ii) If e is multi-occurrence then δ is multi-occurrence.
- (iii) If the type of e is an atomic type, then the type of δ is an atomic type.
- (iv) If the type of e is a complex type, then the type of δ is a complex type.

Definition 3.3 Let P be a set of Path Correspondence Assertions (PCA). We say that P fully specifies T_v in terms of T_s iff: (i) P specifies T_v in terms of T_s ; and (ii) for each PCA in P of the form $[T_v / e] \equiv [T_s / \delta]$, such that e is of complex type T_e , P specifies T_e in terms of T_δ (the type of δ).

Definition 3.4 Let P be a set of PCAs that fully specify T_v in terms of T_s . Let $\$v$ and $\$s$ be instances of T_v and T_s , respectively. We say that $\$v$ matches $\$s$ as specified by P (denoted $\$v \equiv_P \s) iff: For any element e of T_v such that T_e is the type of e and $[T_v / e] \equiv [T_s / \delta]$ is the PCA in P for e (which exists by assumption on P) we have that:

- (i) If δ is not Null, e is mono occurrence and T_e is an atomic type, then $\$v/e = \langle e \rangle \$s / \delta / \text{text}() \langle /e \rangle$
- (ii) If δ is not Null, e is mono occurrence and T_e is a complex type, then $\$v/e \equiv_P \s / δ
- (iii) If δ is not Null, e is multi-occurrence and T_e is an atomic type, then for each $\$a$ in $\$v/e$, there is $\$b$ in $\$s / \delta$ s.t $\$a = \langle e \rangle \$b / \text{text}() \langle /e \rangle$, and for each $\$b$ in $\$s / \delta$, there is $\$a$ in $\$v/e$ s.t $\$a = \langle e \rangle \$b / \text{text}() \langle /e \rangle$

```

Input: types  $T_v$  and  $T_s$ ; set  $P$  of PCAs that fully specifies  $T_v$  in terms of  $T_s$ ;
Output: function  $\tau[T_s \rightarrow T_v](\$s: T_s)$  such that, given an instance  $\$s$  of  $T_s$ ,  $\tau[T_s \rightarrow T_v](\$s)$ 
constructs the content of an instance of  $T_v$  as specified by the PCAs in  $P$ .
 $\tau =$  "declare function  $\tau[T_s \rightarrow T_v](\$s: T_s)$  {";
For each element  $e$  of  $T_v$  where  $[T_v / e] \equiv [T_s / \delta]$ ,  $T_e$  is the type of  $e$ 
and  $T_\delta$  is the type of  $\delta$  Do
  In case of
    Case 1: If  $\delta$  is not Null,  $e$  is mono occurrence and  $T_e$  is an atomic type
       $\tau = \tau + \langle e \rangle \$s / \delta / \text{text}() \langle /e \rangle$  ;
    Case 2: If  $\delta$  is not Null,  $e$  is mono occurrence and  $T_e$  is a complex type, then
       $\tau = \tau + \langle e \rangle \tau[T_\delta \rightarrow T_e](\$s / \delta) \langle /e \rangle$  ;
    Case 3: If  $\delta$  is not Null,  $e$  is multi-occurrence and  $T_e$  is an atomic type, then
       $\tau = \tau + \{ \text{For } \$b \text{ in } \$s / \delta \text{ do return } \langle e \rangle \$b / \text{text}() \langle /e \rangle \}$  ;
    Case 4: If  $\delta$  is not Null,  $e$  is multi-occurrence and  $T_e$  is a complex type, then
       $\tau = \tau + \{ \text{For } \$b \text{ in } \$s / \delta \text{ do return } \langle e \rangle \tau[T_\delta \rightarrow T_e](\$b) \langle /e \rangle \}$  ;
    Case 5: If  $\delta$  is Null, then  $\tau = \tau + \langle e \rangle \tau[T_s \rightarrow T_e](\$s) \langle /e \rangle$  ;
  end case;
end for;
return  $\tau$  ";

```

Fig. 2. Constructor Generation Algorithm

- (iv) If δ is not Null, e is multi-occurrence and T_e is a complex type, then for each $\$a$ in $\$/e$, there is $\$b$ in $\$/\delta$ s.t $\$a \equiv_p \b , and for each $\$b$ in $\$/\delta$, there is $\$a$ in $\$/e$ s.t $\$a \equiv_p \b
- (v) If δ is Null, then $\$/e \equiv_p \s .

Based on Definition 3.4, the algorithm shown in Fig. 2 receives as input the types T_v and T_s , a set \mathbf{P} of PCAs that fully specifies T_v in terms of T_s , and generates the function constructor $\tau[T_s \rightarrow T_v](\$s:T_s)$ such that, given an instance $\$s$ of T_s , $\tau[T_s \rightarrow T_v](\$s)$ constructs the content for an instance $\$/v$ of T_v , such that $\$/v \equiv_p \s .

4 View Specification

In general, we propose to define a view with the help of a view schema, as usual, and a set of view correspondence assertions [20], instead of the more familiar approach of defining a query on the data sources. Without loss of generality, we assume that the type of the root element of a view V , denoted $\text{Trroot}[V]$, has only one element which usually is a multi-occurrence element. Let \mathbf{S} be the XMLS schema of base relational source R , a *view* over \mathbf{S} is a quadruple $V = \langle e, T_e, \Psi, \mathbf{P} \rangle$, where e is the name of the only element of $\text{Trroot}[V]$, T_e is the type of the element e , Ψ is a global collection correspondence assertion which is an expression of the form $[\text{Trroot}[V] / e] \equiv [\text{Trroot}[\mathbf{S}] / R_b / \text{tuple_}R_b [\text{selExp}]]$, where selExp is a predicate expression [23], and \mathbf{P} is a set of path correspondence assertions that fully specifies T_e in terms of $T_{\text{tuple_}R_b}$.

The GCCA Ψ specifies that, given an instance σ_s of $\text{Trroot}[\mathbf{S}]$ and the value σ_v of V in σ_s then, for any $\$a$ in $\text{document}(\sigma_v) / e$, there is $\$b$ in $\text{document}(\sigma_s) / R_b / \text{tuple_}R_b [\text{selExp}]$ such that $\$a \equiv_p \b and, conversely, for any $\$b$ in $\text{document}(\sigma_s) / R_b / \text{tuple_}R_b [\text{selExp}]$, there is $\$a$ in $\text{document}(\sigma_v) / e$ such that $\$a \equiv_p \b ($\$a$ matches $\$b$ as specified by \mathbf{P}).

Consider the XML base source schema **Bib** shown in Fig. 1 and the XML view **View_Bib** where author_v is the only element of $\text{Trroot}[\text{View_Bib}]$ (the type of the root element), and T_{author_v} is the type of the author_v element whose XML schema is shown in Fig. 3. The global collection correspondence assertion matches the author_v element of $\text{Trroot}[\text{View_Bib}]$ with a path from $\text{Trroot}[\mathbf{Bib}]$:

$\Psi: [\text{Trroot}[\text{View_Bib}] / \text{author}_v] \equiv [\text{Trroot}[\mathbf{Bib}] / \text{authors} / \text{tuple_authors} [\text{area}=\text{"Database"}]]$.

Ψ specifies that given an instance σ_{bib} of $\text{Trroot}[\mathbf{Bib}]$ and the value $\sigma_{\text{view_bib}}$ of **View_Bib** in σ_{bib} then:

```

 $\sigma_{\text{view\_bib}} = \langle \text{root\_view\_bib} \rangle$  For  $\$a$  in  $\text{document}(\sigma_{\text{bib}})/\text{authors}/\text{tuple\_author}[\text{area}=\text{"Database"}]$ 
    return  $\langle \text{author}_v \rangle \tau[T_{\text{tuple\_authors}} \rightarrow T_{\text{author}_v}] (\$a) \langle /\text{author}_v \rangle$ 
 $\langle / \text{root\_view\_bib} \rangle$ 

```

Fig. 3 shows the path correspondence assertions of **View_Bib**, which fully specifies T_{author_v} in terms of $T_{\text{tuple_authors}}$. These assertions are generated by: (1) matching the elements of T_{author_v} with elements or paths of the base type $T_{\text{tuple_authors}}$; and (2) recursively descending into sub-elements of T_{author_v} to define their correspondence.

The definition of the view correspondence assertions is supported by the **VBA** (View By Assertion) tool, which features a simple graphical interface. The process starts with the user loading source and target schemas into VBA. The user can then graphically connect elements of the source type with elements or paths of the target type, as indicated in Fig. 3.

The function $\tau[T_{\text{tuple_authors}} \rightarrow T_{\text{author}_v}] (\$a)$ returns the content of the `author_v` element that matches the `tuple_authors` element `$a`, as specified by Ψ_1 to Ψ_4 . Functions $\tau[T_{\text{tuple_books}} \rightarrow T_{\text{book}_v}]$ and $\tau[T_{\text{tuple_publishers}} \rightarrow T_{\text{publisher}_v}]$ are likewise defined using Ψ_5 to Ψ_8 and Ψ_9 to Ψ_{11} , respectively:

```

declare function  $\tau[T_{\text{tuple\_authors}} \rightarrow T_{\text{author}_v}] (\$a \text{ as } T_{\text{tuple\_authors}}) \{$ 
  <ssn_v>{\$a/ssn/text()}</ssn_v> <name_v>{\$a/name/text()}</name_v>
  <email_v>{\$a/email/text()}</email_v>
  { For \$b in \$a / (FK1)-1 / FK2 return <book_v><\tau[Ttuple_books→Tbook_v]($b) </book_v> }
declare function  $\tau[T_{\text{tuple_books}} \rightarrow T_{\text{book}_v}] (\$b \text{ as } T_{\text{tuple_books}}) \{$ 
  <isbn_v>{\$b/ isbn /text()}</isbn_v> <title_v>{\$b/title/text()} </title_v>
  {For \$p in \$b / FK3 return <publisher_v><\tau[Ttuple_publishers→Tpublisher_v]($p) </publisher_v>}}

```

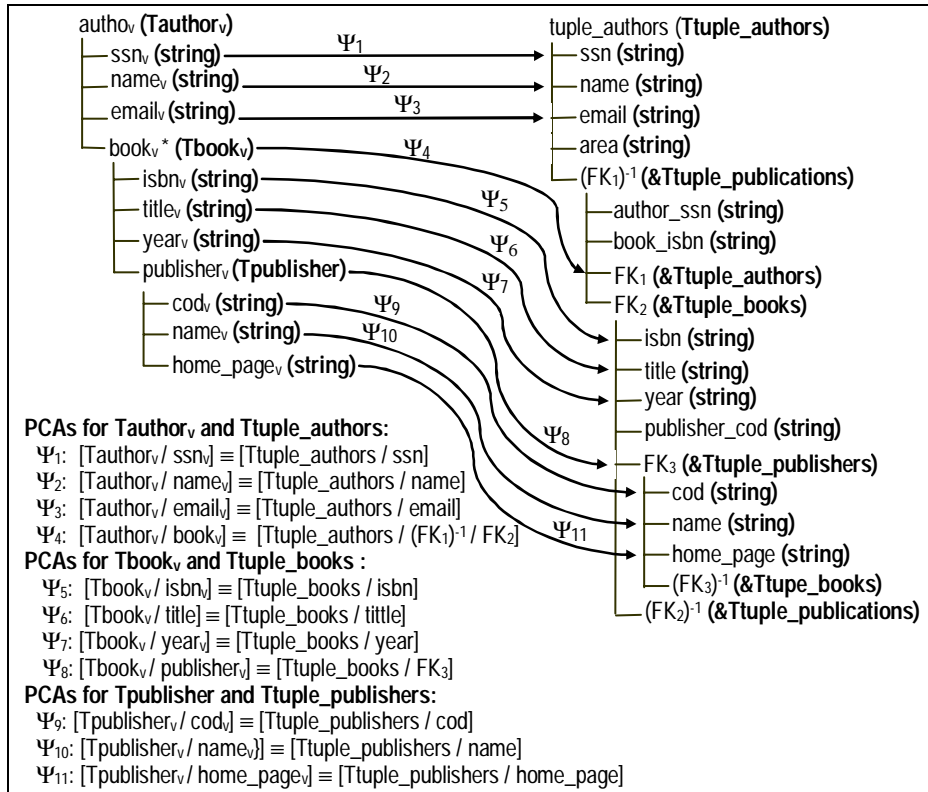


Fig. 3. PCAs for View_Bib

```

declare function  $\tau$ [Ttuple_publishers→Tpublisherv]( $\$p$  as Ttuple_publishers){
  <codv> { $\$p$ /cod/text()}</codv> <namev> { $\$p$ /name/text()} </namev>
  <home_pagev> { $\$p$ /home_page/text()} </home_pagev>};

```

As we have shown, the set of view correspondence assertions fully specify the view in terms of the data source, in the sense that the correspondence assertions of View_Bib (see Fig. 3) define a functional mapping from instances of the source schema **Bib** to instances of the view schema **View_Bib**.

5 Automatic Generation of Rules

In this section, first we present a sequence of definitions and propositions, and then we discuss our approach for automatic generation of incremental view maintenance rules.

In the rest of this section, let **S** be the XMLS schema of a base relational source *R* and $V = \langle e, T_e, \Psi, P \rangle$ be a view over **S**, where $\text{Trroot}[V]$ contains a sequence of elements e of type T_e and the GCCA is of the form:

$$\Psi: [\text{Trroot}[V] / e] \equiv [\text{Trroot}[S] / R_b / \text{tuple_Rb}[\text{selExp}]]$$

Definition 5.1: A path δ_v of T_v is *semantically related* to a path δ_b of $T_{\text{tuple_Rb}}$, denoted $\delta_v \equiv \delta_b$, iff there are paths $\delta_{v_1}, \dots, \delta_{v_n}$ in **V** and paths $\delta_{b_1}, \dots, \delta_{b_n}$ in **S** such that $\delta_v = \delta_{v_1} / \dots / \delta_{v_n}$, $\delta_b = \delta_{b_1} / \dots / \delta_{b_n}$, $[T_v / \delta_{v_i}] \equiv [T_b / \delta_{b_i}] \in P$ and $[T_{v_i} / \delta_{v_{i+1}}] \equiv [T_{b_i} / \delta_{b_{i+1}}] \in P$, for $1 \leq i \leq n-1$.

Definition 5.2: A path δ_s of $\text{Trroot}[S]$ is *relevant* to a path of $\text{Trroot}[V]$ in the following cases:

- Case 1. $\delta_s = R_b / \text{tuple_Rb}$ is relevant to path e of $\text{Trroot}[V]$.
- Case 2. $\delta_s = R_b / \text{tuple_Rb} / a$ is relevant to path e of $\text{Trroot}[V]$, where a is an element or an attribute in selExp .
- Case 3. Let δ_s and δ_v be paths of $T_{\text{tuple_Rb}}$ and T_e , respectively, such that $\delta_v \equiv \delta_s$, then $R_b / \text{tuple_Rb} / \delta_s$ is relevant to e / δ_v .

A path of $\text{Trroot}[S]$ is *relevant* to V iff it is relevant to a path of $\text{Trroot}[V]$.

Definition 5.3: A path δ of $\text{Trroot}[S]$ is *relevant* to V iff δ is relevant to a path of $\text{Trroot}[V]$.

Definition 5.4: Let $\sigma = (\mathbf{N}, \mathbf{A})$ be an instance of a type **T** and let $\delta = e_1 / \dots / e_n$ be a path of **T**.

(i) $G[\delta, \sigma]$ is a directed graph $(\mathbf{N}', \mathbf{A}')$, a subgraph of σ , where \mathbf{N}' contains all nodes in \mathbf{N} that are visited when the path δ is traversed starting from the root, and all nodes in \mathbf{N} that are descendants of a node e_n in $\text{document}(\sigma) / e_1 / \dots / e_n$. $G[\delta, \sigma]$ represents the value of δ in σ .

(ii) $H[\delta, \sigma]$ is a directed graph $(\mathbf{N}'', \mathbf{A}'')$, a subgraph of $G[\delta, \sigma]$, where \mathbf{N}'' contains all nodes in \mathbf{N} that are visited when the path δ is traversed starting from the root.

In what follows, let μ be an update over a table of *R*, and U be a list of updates against the view *V*.

Definition 5.5: Let δ_s be a path of $\text{Troot}[\mathbf{S}]$. We say that δ_s can be affected by μ iff there are instances σ and σ' of $\text{Troot}[\mathbf{S}]$ such that σ and σ' are the instances immediately before and after μ , respectively, and $H[\delta_s, \sigma] \neq H[\delta_s, \sigma']$.

Definition 5.6: Let δ_v be a path of $\text{Troot}[\mathbf{V}]$. We say that δ_v can be affected by μ iff there are instances σ and σ' of $\text{Troot}[\mathbf{S}]$ such that σ and σ' are the instances immediately before and after μ , respectively, and σ_v and σ'_v are the values of V in σ and σ' , respectively, and $H[\sigma_v, \delta_v] \neq H[\sigma'_v, \delta_v]$.

Proposition 5.1: A path δ_v of $\text{Troot}[\mathbf{V}]$ can be affected by μ iff there is a path δ_s of $\text{Troot}[\mathbf{S}]$ such that δ_s is relevant to δ_v and δ_s can be affected by μ .

Definition 5.7 We say that U correctly maintains V w.r.t. μ iff for any pair of instances of $\text{Troot}[\mathbf{S}]$, σ_s and σ'_s , if σ_s and σ'_s are the instances immediately before and after μ , respectively, σ_v and σ'_v are the values of V in σ_s and σ'_s , respectively, and $\sigma''_v = U(\sigma_v)^1$ then $\sigma''_v = \sigma'_v$.

Definition 5.10: Let δ_v be a path of $\text{Troot}[\mathbf{V}]$. We say that U correctly maintains δ_v w.r.t. μ iff for any pair of instances of $\text{Troot}[\mathbf{S}]$, σ_s and σ'_s , if σ_s and σ'_s are the instances immediately before and after μ , respectively, σ_v and σ'_v are the values of V in σ_s and σ'_s , respectively, and $\sigma''_v = U(\sigma_v)$ then $G[\delta_v, \sigma''_v] = G[\delta_v, \sigma'_v]$.

Proposition 5.2: If U correctly maintains all paths δ_v of $\text{Troot}[\mathbf{V}]$ w.r.t. μ , then U correctly maintains V .

Proposition 5.3: Let $\delta_v = \delta_1 / \delta_2$ be a path of $\text{Troot}[\mathbf{V}]$. Suppose that δ_1 can be affected by μ . If U correctly maintains δ_1 w.r.t. μ , then U correctly maintains δ_v .

In our approach, the process of generating the maintenance rules for V consists of the following steps:

1. Obtain the set $\mathcal{A}[\mu, V]$ of all paths δ_s such that: (i) δ_s is relevant to a path δ_v of $\text{Troot}[\mathbf{V}]$; (ii) δ_s is affected by μ ; and, (iii) there is no path δ' such that δ' is a proper prefix of δ_s and $\delta' \in \mathcal{A}[\mu, V]$. $\mathcal{P}[\mu, V]$ denote the set of paths that are relevant to V with respect to a base update μ over S .
2. For each path δ_v of $\text{Troot}[\mathbf{V}]$ where there is δ_s in $\mathcal{A}[\mu, V]$ such that δ_s is relevant to δ_v , generate the rule for maintaining δ_v with respect to μ . Note that, if $\mathcal{A}[\mu, V] = \emptyset$, then μ is not relevant to V and no rule is generated.

From Proposition 5.2, we have that the rules correctly maintain a view V iff they correctly maintain all paths of V . So, our strategy for proving that the rules created by the above process correctly maintain V with respect to an update μ consists in demonstrating that: (i) all updates generated by the rules correctly maintain all paths δ_v of $\text{Troot}[\mathbf{V}]$ that can be affected by μ , and (ii) all other paths cannot be affected by the updates generated by rules. From Propositions 5.1 and 5.3, we have to generate rules for maintaining each path δ_v where there is δ_s in $\mathcal{P}[\mu, V]$ such that δ_s is relevant to δ_v , which are the paths corresponding to the rules generated in Step 2 above.

¹ $U(\sigma_v)$ returns the new value of view V after applying U to σ_v

Let R be a table of base source R , Theorem 5.1 establishes sufficient condition to compute the set $\mathcal{A}[\mu, V]$ when μ is an insertion or deletion over R , and Theorem 5.2 when μ is an update on attributes of R .

Theorem 5.1: Let μ be an insertion or deletion operation over R .

Case 1: Suppose that $R = R_b$. Then, we have $\mathcal{A}[\mu, V] = \{ R_b / \text{tuple_}R_b \}$

Case 2: Suppose that $R \neq R_b$. Let δ_s be a path of $\text{Troot}[S]$. Then, a path $\delta_s \in \mathcal{A}[\mu, V]$ iff $\delta_s = R_b / \text{tuple_}R_b / \delta_2 / FK_1^{-1} / \delta_1$, where δ_1 and δ_2 can be null, FK_1 is a foreign key of R , δ_s is relevant to V and there is no path δ' such that δ' is a proper prefix of δ_s and $\delta' \in \mathcal{A}[\mu, V]$.

Theorem 5.2: Let μ be an update of the attribute a on R , and δ_s be a path of $\text{Troot}[S]$.

Case 1: Suppose that $R = R_b$. Then, a path $\delta_s \in \mathcal{A}[\mu, V]$ iff

Case 1.1: $\delta_s = R_b / \text{tuple_}R_b / a$ where $R_b / \text{tuple_}R_b / a$ is relevant to V .

Case 1.2: $\delta_s = R_b / \text{tuple_}R_b / FK / \delta_2$ where FK is a foreign key of R_b such that a is an attribute of FK , δ_s is relevant to V and there is no path δ' such that δ' is a proper prefix of δ_s and $\delta' \in \mathcal{P}[\mu, V]$.

Case 2: Suppose that $R \neq R_b$. Suppose that $R \neq R_b$. Then, a path $\delta_s \in \mathcal{A}[\mu, V]$ iff

Case 2.1: $\delta_s = R_b / \text{tuple_}R_b / \delta / FK / a$ where δ can be null, FK is a foreign key that references R and δ_s is relevant to V .

Case 2.2: $\delta_s = R_b / \text{tuple_}R_b / \delta / FK^{-1} / a$ where δ can be null, FK is a foreign key of R and δ_s is relevant to V .

Case 2.3: $\delta_s = R_b / \text{tuple_}R_b / \delta_1 / FK / \delta_2$, where δ_2 can be null, FK is a foreign key of R such that a is an attribute of FK , δ_s is relevant to V and there is no path δ' such that δ' is a proper prefix of δ_s and $\delta' \in \mathcal{P}[\mu, V]$.

Case 2.4: $\delta_s = R_b / \text{tuple_}R_b / \delta_1 / FK^{-1} / \delta_2$, where δ_1 e δ_2 can be null, FK is a foreign key of R such that a is an attribute of FK , δ_s is relevant to V and there is no path δ' such that δ' is a proper prefix of δ_s and $\delta' \in \mathcal{P}[\mu, V]$.

Example 1: Consider the view **View_Bib** defined in Section 3 and let μ be the update of the attribute `publisher_cod` of table `Books`. From Case 3 of Definition 5.2 we have that $\delta = \text{authors} / \text{tuple_authors} / FK_1^{-1} / FK_2 / FK_3$, where FK_3 is a foreign key of `Books`, is relevant to view path `author_v / book_v / publisher_v`. From Case 2.3 of Theorem 5.2 we have that $\delta \in \mathcal{A}[\mu, \text{View_Bib}]$. Fig. 4 shows the rule for maintaining path `author_v / book_v / publisher_v` w.r.t μ . In this rule $\$view_bib$ represents the current instance of $\text{Troot}[\text{View_Bib}]$, $\$new$ and $\$old$ represents the old and new state of the updated `tuple_book` element, respectively. According to the rule, the updates required for maintaining `author_v / book_v / publisher_v` reduce to: For each `book_v` element in $\$view_bib / \text{author}_v / \text{book}_v$ that matches $\$new$ then: (i) delete the sub-element `publisher_v` that matches the element `tuple_publishers` in $\$old / FK_3$ and (ii) insert the sub-element `publisher_v` that matches the element `tuple_publishers` in $\$new / FK_3$. The rule can be automatically generated based on the rule template shown in Fig. 5. This template should be applied in case where the relevant path satisfy conditions of Case 2.3 of Theorem 5.2. The complete family of rule templates for insertion, deletion and update operation, covering all cases of Theorems 6.1 and 6.2, can be found in [20].

```

when UPDATE OF publisher_cod ON Books then
  let $v_n in $view_bib / author_v / book_v where $v_n / isbn_v / text() = $new / isbn / text()
  for $d in $old / FK_3 do
    let $v_{n+1} in $v_n / publisher_v where $v_{n+1} / cod_v / text() = $d / cod / text()
    delete $v_{n+1} as child of $v_n;
  for $i in $new / FK_3 do
    $c := <publisher_v>τ [T_{tuple_publishers} → T_{publisherv}] ($i) </publisher_v>
    insert $c as a child of $v_n;

```

Fig. 4. Rule for maintaining path author_v/ book_v/ publisher_v w.r.t μ

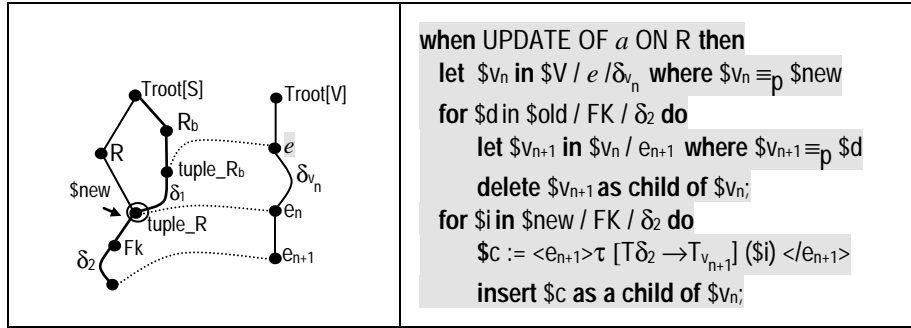


Fig. 5. Rule for maintaining the view V with respect to an update of an attribute a of R satisfying the following conditions: (i) $R \neq R_b$ and (ii) the path $R_b / \text{tuple_}R_b / \delta_1 / FK / \delta_2$ is relevant to path $e / \delta_{v_n} / v_{n+1}$ of $\text{Trout}[V]$, where Fk is a foreign key of R and $R_b / \text{tuple_}R_b / \delta_1 \equiv_p e / \delta_{v_n}$ (case 2.3 of Theorem 5.2).

6 Identifying Relevant Paths

In this section, let S be a XMLS schema of base relational source R and R be a table of R . Consider a view $V = \langle e, T_v, \Psi, P \rangle$ defined over S , as in Section 4. To automatically compute the set of all paths of $\text{Trout}[S]$ which are relevant to V , we use the *relevant path tree* of V , denoted $\mathcal{A}[V]$.

Fig. 6 shows the relevant path tree for the View_Bib defined in Section 3. For each path $\text{root}, e_1, \dots, e_n$ in $\mathcal{A}[\text{View_Bib}]$, there is a path $e_1 / \dots / e_n$ of $\text{Trout}[\text{Bib}]$ which is a prefix of a path relevant to View_Bib, and vice-versa. If e_n is a black node, then $e_1 / \dots / e_n$ is relevant to View_Bib. Hence, paths authors / tuple_authors and authors / tuple_authors / area are relevant to View_Bib, and the path authors is not relevant, since it is a prefix of the relevant path authors / tuple_authors.

Fig. 6 also shows the assertions that originated nodes and edges. For example, from the GCCA Ψ , authors / tuple_authors and authors / tuple_authors / area are relevant to View_Bib (cases 1 and 2 of relevant path). From Ψ and Ψ_1 , authors / tuple_authors /

ssn is relevant to View_Bib (case 3). From Ψ and Ψ_4 , authors / tuple_authors / (FK1)⁻¹ / FK2 is relevant to View_Bib (case 3).

The relevant paths for a given base update operation can be automatically identified using the relevant path tree as we illustrate in the following example.

Example 2: Let μ be an insertion or deletion on table Publications. From $\mathcal{A}[\text{View_Bib}]$, we have that the path $\delta = \text{authors} / \text{tuple_authors} / (\text{FK}_1)^{-1} / \text{FK}_2$, where FK₁ is a foreign key of Publications, is relevant to path author_v / book_v of Troot[view_Bib]. From Case 2 of Theorem 5.1, we have that $\delta \in \mathcal{A}[\mu, \text{View_Bib}]$.

Example 3: Let μ be an update of the attribute area of table Authors. From $\mathcal{A}[\text{View_Bib}]$, we have that $\delta = \text{authors} / \text{tuple_authors} / \text{area}$ is relevant to path author_v of Troot[view_Bib]. From Case 1.1 of Theorem 5.1, we have that $\delta \in \mathcal{A}[\mu, \text{View_Bib}]$.

Example 4: Let μ be update of the attribute home_page of table Publishers. From $\mathcal{S}[\text{View_Bib}]$, we have that $\delta = \text{authors} / \text{tuple_authors} / (\text{FK}_1)^{-1} / \text{FK}_2 / \text{FK}_3 / \text{home_page}$ where FK₃ is a foreign key that references Publishers, is relevant to path author_v / book_v / publisher_v / home_page_v of Troot[View_Bib]. From Case 2.1 of Theorem 5.2, we have that the path $\delta / \text{home_page} \in \mathcal{A}[\mu, \text{View_Bib}]$.

Example 5: Let μ be update of the attribute publisher_cod of table Books. From $\mathcal{S}[\text{View_Bib}]$, we have that $\delta = \text{authors} / \text{tuple_authors} / (\text{FK}_1)^{-1} / \text{FK}_2 / \text{FK}_3$, where FK₃ is a foreign key of Books that references Publishers such that publisher_cod is an attribute of FK₃, is relevant to path author_v / book_v / publisher_v of Troot[View_Bib]. From Case 2.3 of Theorem 5.2, we have that $\delta \in \mathcal{A}[\mu, \text{View_Bib}]$.

The full details of the algorithms to identify, for a view V, the set $\mathcal{A}[\mu, V]$ to insertions, deletions and update operations can be found in [20].

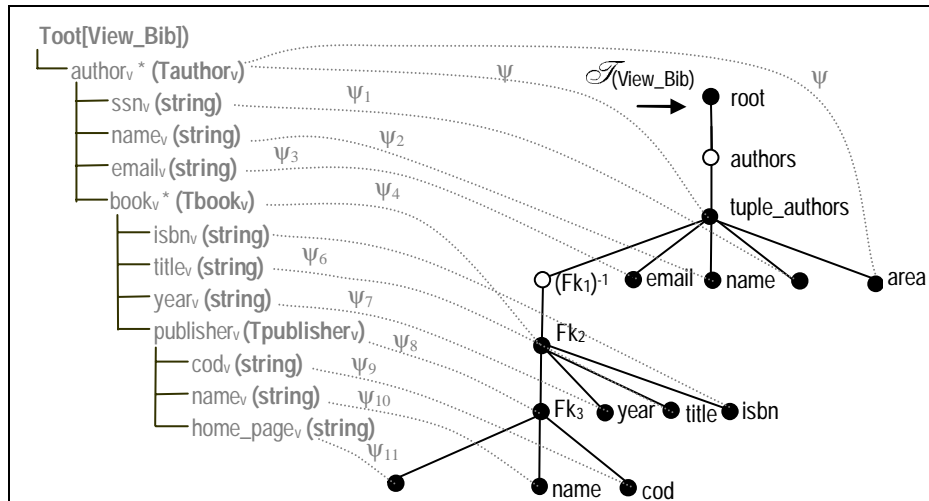


Fig. 6. Relevant path tree of View_Bib

7 Conclusions

We argued in this paper that the view correspondence assertions fully specify the view in terms of the data source, in the sense that they define a mapping from instances of the source schema to instances of the view schema.

We showed that, based on the view correspondence assertions, we can automatically and efficiently identify all view paths that can be affected by a base update. Using view correspondence assertions, we can also define the rule required for maintaining a view path with respect to a given base update. Finally, we indicated how rules can be automatically generated from rule templates.

It is important to notice that, in our approach the rules are responsible for directly propagating the changes to the materialized view. Therefore no algorithm for view maintenance is required. The effectiveness of our approach is assured for externally maintained views because: the view maintenance statements are defined at view definition time; no access to the materialized view is required to compute the view maintenance statements propagated by the rules; and the propagated view maintenance statements do not require any additional queries over the data source to maintain the view.

We are currently working on the development of a tool to automate the generation of incremental view maintenance rules. The tool uses a family of rule templates which are used to generate the specific rules for maintaining the view paths that can be affected by a given base update. We have already defined the rule templates for most types of object preserving views.

References

1. Abiteboul, S., McHugh, J., Rys, M., Vassalos, V., Wiener, J.L.: Incremental Maintenance for Materialized Views over Semistructured Data. In Proceedings of the International Conference on Very Large Databases. New York City (1998) 38-49.
2. Ali, M.A., Fernandes, A.A., Paton, N.W.: Incremental Maintenance for Materialized OQL Views. In Proc. DOLAP (2000) 41-48.
3. Ali, M.A., Fernandes, A.A., Paton, N.W.: MOVIE: an incremental maintenance system for materialized object views, *Data & Knowledge Engineering*, v.47 n.2, p.131-166, November 2003.
4. Benedikt, M., Chan, C. Y., Fan, W., Rastogi, R., Zheng, S., Zhou, A.: DTD-directed publishing with attribute translation grammars. In VLDB, 2002.
5. Bohannon, P., Choi, B., Fan, W.: Incremental evaluation of schema-directed XML publishing, Proceedings of the 2004 ACM SIGMOD international conference on Management of data, June 13-18, 2004, Paris, France.
6. Byron Choi, Wenfei Fan, Xibei Jia, Arek Kasprzyk: A Uniform System for Publishing and Maintaining XML Data. VLDB 2004: 1301-1304.
7. Carey, M.J., Kiernan, J., Shanmugasundaram, J., Shekita, E.J., Subramanian, S.N.: XPERANTO: Middleware for Publishing Object-Relational Data as XML Documents. In *The VLDB Journal*(2000) 646-648.

8. Ceri, S., Widom, J.: Deriving productions rules for incremental view maintenance. In Proceedings of the International Conference on Very Large Databases (1991) 577-589 4
9. Dimitrova, K., El-Sayed, M., Rundensteiner, E. A.: Order-sensitive view maintenance of materialized XQuery views. In ER, 2003.
10. EL-Sayed, M., Wang, L., Ding, L., Rundensteiner, E.: An algebraic approach for Incremental Maintenance of Materialized Xquery Views. In Proceedings of Fourth International Workshop on Web Information and Data Management. McLean, USA (2002).
11. Fernandez, M., Morishima, A., Suciu, D., Tan, W.: Publishing Relational Data in XML: the SilkRoute Approach. IEEE Trans on Computers, 44(4). (2001) 1-9.
12. Gluche, D., Grust, T., Mainberger, C., Scholl, M.: Incremental updates for materialized OQL views, in: Proc. DOOD, 1997, pp. 52-66.
13. Gupta, A., Mumick, I.S., Subrahmanian, V.S.: Maintaining Views Incrementally. In SIGMOD (1993) 157-166.
14. Gupta, A., Mumick, I.S.: Maintenance of Materialized Views: Problems, Techniques, and Applications. In IEEE Bulletin on Data Engineering, 18(2). (1995) 3-18.
15. Gupta, A., Mumick, I.S.: Materialized Views. MIT Press, 2000.
16. Hernández, M.A., Miller, R.J, Haas, L.M.: Clio: A Semi-Automatic Tool For Schema Mapping. SIGMOD Conference 2001.
17. Kuno, H.A., and Rundensteiner, E.A.: Incremental Maintenance of Materialized Object-Oriented Views in MultiView: Strategies and Performance Evaluation. IEEE Transaction on Data and Knowledge Engineering, 10(5):768-792 (1998).
18. Suciu, D.: Query decomposition and view maintenance for query languages for unstructured data. In Proceedings of VLDB, pages 227--238, 1996.
19. Vidal, V.M.P., Casanova, M.A, Araujo, V.S.: Generating Rules for Incremental Maintenance of XML Views of Relational Data. Proceedings of the 9th ACM International Workshop on Web Information and Data Management, WIDM 2003 139-146.
20. Vidal, V.M.P., Casanova, M.A., Araujo, V.S: Automatic Generation of Rules for the Incremental Maintenance of XML Views over Relational Data. Technical Report. Universidade Federal do Ceará, Janeiro, 2005.
21. Vidal, V.M.P., Casanova, M.A.: Efficient Maintenance of XML Views Using View Correspondence Assertions. In Proceedings of the 4th International Conference on Electronic Commerce and Web Technologies. Praga, Czech Republic (to appear).
22. Vidal, V.M.P., Vilas Boas, R.: A Top-Down Approach for XML Schema Matching. In Proceedings of the 17th Brazilian Symposium on Databases. Gramado, Brazil (2002).
23. World-Wide Web Consortium: *XML Path Language (XPath)*: Version 1.0 (November 1999). <http://www.w3.org/TR/xpath> (visited on June 25th, 2005).
24. World-Wide Web Consortium: *XML Query Language (XQuery)*: Version 1.0 (November 1999). <http://www.w3.org/TR/xquery/> (visited on June 25th, 2005).
25. Yu, C., Popa, L.: Constraint-Based XML Query Rewriting For Data Integration. SIGMOD Conference 2004: Paris, France, June 2004, pp. 371-382.
26. Yuan Fa, Ya Bing Chen, Tok Wang Ling, Ting Chen: Materialized View Maintenance for XML Documents. WISE 2004: 365-371.
27. Zhuge Y., Garcia-Molina, H.: Graph structured views and their incremental maintenance. In Proceedings of ICDE, pages 116-125, 1998.