

Kurt Bauknecht
A Min Tjoa
Gerald Quirchmayr (Eds.)

LNCS 2738

E-Commerce and Web Technologies

4th International Conference, EC-Web 2003
Prague, Czech Republic, September 2003
Proceedings

658.8406 I61 2003

Autor: International C

Título: E-commerce and Web tec



411675

152.154

PUC-Rio PUCI



Springer

Kurt Bauknecht A Min Tjoa
Gerald Quirchmayr (Eds.)

E-Commerce and Web Technologies

4th International Conference, EC-Web
Prague, Czech Republic, September 2-5, 2003
Proceedings



Springer

Efficient Maintenance of XML Views Using View Correspondence Assertions

Vânia Maria Ponte Vidal¹ and Marco Antonio Casanova²

¹Dept. Computação, Universidade Federal do Ceará
Fortaleza, CE – Brasil
vvidal@lia.ufc.br

²Dept. Informática, PUC-Rio
Rio de Janeiro, RJ – Brasil
casanova@fplf.org.br

Abstract. 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. These views may be materialized to achieve faster data access. The main difficulty with this approach is to maintain the consistency of the materialized view with respect to changes of base data. In this paper, we propose an algorithm for the incremental maintenance of XML views. Our algorithm uses the view correspondence assertions for checking the relevance, for the view, of a base update and computes the changes needed for propagating the update to the view.

1 Introduction

Over the last years, the Web has become the largest environment capable of providing access to heterogeneous data sources and XML came to be the standard for the representation and exchange of data over the Web. As a result, data sources often export XML views over base data [3], [6]. The exported view can be either virtual or materialized. Materialized views improve query performance and data availability, but they must be updated in order to reflect changes in the base source.

Basically, there are two strategies for materialized view maintenance: re-materialization and incremental maintenance. In the re-materialization strategy, view data is re-computed at pre-established times. By contrast, in the incremental maintenance strategy, a mechanism periodically modifies view data to reflect updates to local sources. Incremental view maintenance proved to be an effective solution [1], [5].

The view maintenance problem has been extensively studied for relational and object-oriented databases [1], [4], [7], [8], [9]. Abiteboul et al [2] proposed an incremental maintenance algorithm for materialized views over semi-structured data, considering the graph-based data model OEM and the query language Lorel. El-Sayed et al [5] proposed an algebraic solution approach based on the XAT XML algebra.

In this paper, we propose an algorithm for the incremental maintenance of XML views that uses view correspondence assertions, which define relationships between the view schema and the base data source. We show how to analyze these assertions, at view definition time, to generate information that is used, at run time, to propagate updates to the base data sources to the materialized view. The use of this information brings significant advantage to our approach, when compared to previous approaches to XML view maintenance.

This paper is organized as follows. Section 2 reviews basic concepts and introduces the graphical notation used to represent XML schema types. Section 3 discusses the process of generating view correspondence assertions. Section 4 presents the algorithm for the incremental maintenance of XML views.

2 XML Fundamentals

We adopt a graphical notation [10] to represent the types of a XML schema S . Briefly, the notation uses a tree-structured representation for the types of S , where bold fonts denote the name of the type, “&” denotes references, “@” denotes attributes and “*” denotes multiple occurrences of an element.

Figure 1 shows the types of a XML schema *Bib*, where:

- T_{bib} is the type of the root element and contains a books element of type T_{books} , an authors element of type $T_{authors}$, and an articles element of type $T_{articles}$;
- $T_{authors}$ contains a sequence of zero or more author elements of type T_{author} ;
- T_{author} contains the attribute email and the elements name and area. The type ID, specified for the attribute email, allows a unique identification to be associated with each author element;

The attribute *author_ref* of T_{book} contains a reference to an author element of type T_{author} (& T_{author}).

A *link* represents a relationship between XML Types. There is a link from type T_1 to type T_2 iff: (i) T_1 contains an element e whose type is T_2 , denoted $e : T_1 \rightarrow T_2$; (ii) T_2 contains an element e whose type is T_1 , denoted $e : T_1 \rightarrow T_2$; (iii) T_1 contains an element e whose type is a reference & T_2 to T_2 , denoted $e \rightarrow : T_1 \rightarrow T_2$; (iv) T_2 contains an element e whose type is a reference & T_1 to T_1 , denoted $(e \rightarrow) : T_1 \rightarrow T_2$.

We adopt an extension of XPATH that permits navigating through a reference link. The result of a path expression is a sequence of nodes or primitive values.

Let $e \rightarrow : T_1 \rightarrow T_2$ be a reference link. Given an instance $\$e_1$ of T_1 , the path expression $\$e_1/e \rightarrow$ returns all instances of T_2 referenced in $\$e_1/e$.

Conversely, given an instance $\$e_1$ of T_1 , the path expression $\$e_1/l^{-1}$ returns the instance $\$e_2$ of T_2 such that $\$e_1$ in $\$e_2/l$.

Instances of a type T_1 can be related with instances of a type T_n through the composition of two or more links. Consider links $l_i : T_i \rightarrow T_{i+1}$, for $i=1, \dots, n-1$, where T_i are types of an XML schema. Therefore $\delta = l_1 / l_2 / \dots / l_{n-1}$ is a path of T_1 . Given an instance $\$e$ of T_1 , the path expression $\$e/\delta$ selects a set of elements of type T_n . Hence, the type T_δ of the path δ is T_n .

3 Specifying View Correspondence Assertions

In general, we propose to define a view with the help of a *view schema*, as usual, and a set of *view correspondence assertions* [10], instead of the more familiar approach of defining a query on the data sources. These assertions axiomatically specify how view objects are synthesized from data source objects.

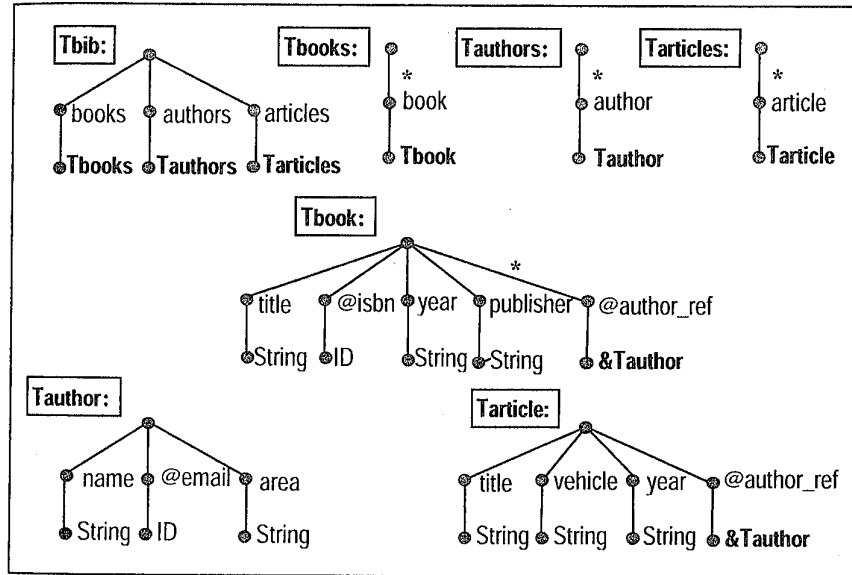


Fig. 1. Graphical representation for the Bib XML Schema

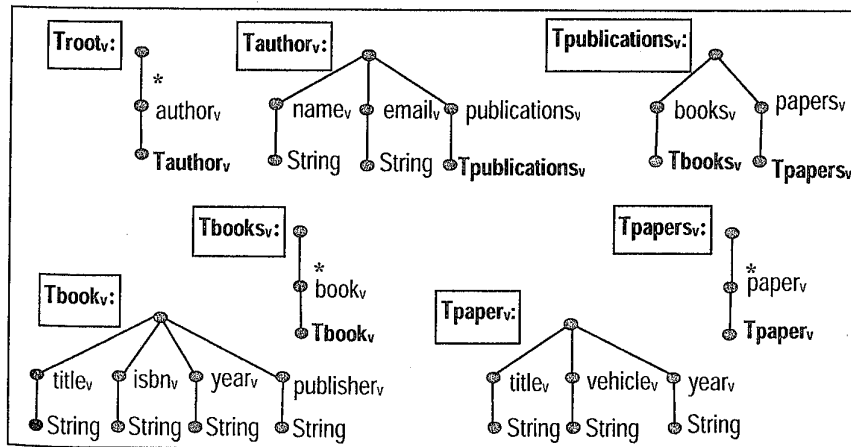


Fig. 2 XML Schema for view V

We exemplify the process, described in [10], for generating the correspondence assertions by matching the schemas V and Bib , represented in Figures 1 and 2. In what follows, let $\$V$, $\$bib$ be global variables corresponding to the XML documents that represent V and Bib , respectively.

The matching process is top down and consists of two steps:

Step 1: We first match the primary elements of V . For example, the *global collection correspondence assertion* (GCCA) [10]:

$$\Psi_1: [\$V/author_v] \equiv [\$bib/authors/author[area = "Database"]]$$

specifies that $\$V/author_v$ and $\$bib/authors/author[area = "Database"]$ denote the same set of real world objects.

Then, we define *matching correspondence assertions* (MCAs) to specify the criteria for performing the matching of elements in two semantically related collections. For example, the MCA

$$\Psi_2: [Tauthor_v, \{email_v\}] \equiv [Tauthor, \{@email\}]$$

specifies that an author $\$a_v$ in $\$V/author_v$ matches an author $\$a$ in $\$Bib/authors/author$ iff $\$a_v/email_v/text()=\$a/@email/text()$.

Step 2: We next specify the correspondence assertions for the sub-elements of V . For example, the *path correspondence assertions* (PCAs) [10], generated by matching the types $Tauthor_v$ and $Tauthor$:

$$\Psi_3: [Tauthor_v/name_v] \equiv [Tauthor/name]$$

$$\Psi_4: [Tauthor_v/email_v] \equiv [Tauthor/@email]$$

$$\Psi_5: [Tauthor_v/publications_v/books_v/book_v] \equiv [Tauthor/(Tbook.author_ref->)^{-1}]$$

$$\Psi_6: [Tauthor_v/publications_v/papers_v/paper_v] \equiv [Tauthor/(Tarticle.author_ref->)^{-1}]$$

These assertions specify relationships between paths of $Tauthor_v$ and $Tauthor$.

Given an instance $\$a_v$ of $Tauthor_v$, if there is an instance $\$a$ of $Tauthor$ such that $\$a_v=\a , then:

- $\$a_v$ contains an element $name_v$ such that $\$a_v/name_v/text()=\$a/name/text()$ (from Ψ_3);
- $\$a_v$ contains an element $email_v$ such that $\$a_v/email_v/text()=\$a/@email/text()$ (from Ψ_4);
- $\$a$ contains an element $publications_v$ such that
- $\$a_v/publications_v/books_v/book_v=\$a/(Tbook.author_ref->)^{-1}$ (from Ψ_5)
- $\$a_v/publications_v/papers_v/paper_v=\$a/(Tarticle.author_ref->)^{-1}$ (from Ψ_6).

In case of Ψ_5 , since $Tbook_v$ is a complex type, we also introduce the following MCA to match instances of $Tbook_v$ with instances of $Tbook$:

$$\Psi_7: [Tbook_v, \{isbn_v\}] \equiv [Tbook, \{@isbn\}]$$

and the following PCAs, obtained by matching $Tbook_v$ and $Tbook$:

$$\Psi_8: [Tbook_v/title_v] \equiv [Tbook/title]$$

$$\Psi_9: [Tbook_v/isbn_v] \equiv [Tbook/@isbn]$$

$$\Psi_{10}: [Tbook_v/year_v] \equiv [Tbook/year]$$

$$\Psi_{11}: [Tbook_v/publisher_v] \equiv [Tbook/publisher]$$

In case of Ψ_6 , since T_{paper_v} is a complex type, we also introduce the MCA

$$\Psi_{12}: [T_{\text{paper}_v}, \{\text{title}_v\}] \equiv [T_{\text{article}}, \{\text{title}\}]$$

and the following PCAs obtained by matching T_{paper_v} and T_{article} :

$$\Psi_{13}: [T_{\text{paper}_v}/\text{title}_v] \equiv [T_{\text{article}}/\text{title}]$$

$$\Psi_{14}: [T_{\text{paper}_v}/\text{vehicle}_v] \equiv [T_{\text{article}}/\text{vehicle}]$$

$$\Psi_{15}: [T_{\text{paper}_v}/\text{year}_v] \equiv [T_{\text{article}}/\text{year}].$$

4 Algorithm for the Incremental Maintenance of XML Views

4.1 Terminology

In this section, we study the maintenance of XML views, defined as follows. Consider V , a view that contains a set of v elements of type T_v , specified by the GCCA

$$\Psi: [\$V/v] \equiv [\$S/\text{PathExp}],$$

where S is the base source and $\text{PathExp} = e_1[\text{selExp}_1]/\dots/e_n[\text{selExp}_n]$, where $e_1/\dots/e_n$ is a path of the root type of S (T_{roots}), and selExp_k , $1 \leq k \leq n$, is a predicate expression [12]. The GCCA Ψ specifies that $\$V/v$ and $\$S/e_1[\text{selExp}_1]/\dots/e_n[\text{selExp}_n]$ denote the same set of real world objects. The *sub-elements* of v are specified by the PCA of T_v with T_{e_n} as discussed in Section 3.

Definition 4.1: Let δ_s and δ_p be paths of T_s and T_p . We say that δ_s is *semantically related* to δ_p , denoted $\delta_s \equiv \delta_p$, iff there are paths $\delta_{s1}, \dots, \delta_{sn}$, with $\delta_s = \delta_{s1}/\dots/\delta_{sn}$, and paths $\delta_{p1}, \dots, \delta_{pn}$, with $\delta_p = \delta_{p1}/\dots/\delta_{pn}$, such that

$$[T_s/\delta_{s1}] \equiv [T_p/\delta_{p1}] \text{ and } [T_{s_i}/\delta_{s_{i+1}}] \equiv [T_{p_i}/\delta_{p_{i+1}}], 1 \leq i \leq n-1.$$

In the examples of this section, we use *Bib* schema of Figure 1, the view V in Figure 2, and the view correspondence assertions for V , defined in Section 3.

Example 4.1: From PCA $\Psi_5: [T_{\text{author}_v}/\text{publications}_v/\text{books}_v/\text{book}_v] \equiv [T_{\text{author}}/(\text{Tbook.author_ref-})^{-1}]$, we have that the path $\text{publications}_v/\text{books}_v/\text{book}_v$ of T_{author_v} is semantically related to the path $(\text{Tbook.author_ref-})^{-1}$ of T_{author} .

Definition 4.2: Let V be a view specified by the GCCA

$$\Psi: [\$V/v] \equiv [\$S/e_1[\text{selExp}_1]/\dots/e_n[\text{selExp}_n]].$$

Let $\delta_0 = e_1/\dots/e_n$ where $T_{\delta_0} = T_{e_n}$. We say that a path δ of the root type T_{roots} of S is *relevant* to V iff δ satisfies one of the following conditions:

1. δ is a prefix of δ_0 ,
2. δ is a prefix of δ_0/δ_p , where path δ_p of T_{δ_0} is S.R. to the path δ_v of T_v ($\delta_p \equiv \delta_v$)
3. δ is a prefix of δ_p/δ_q , where $\delta_p = e_1/\dots/e_p$, with $p < n$, and δ_q is a condition path in selExp_p .

Example 4.2: Let $\$bib/authors/author/(Tbook.author_ref\rightarrow)^{-1}$ be a path of the root type of the schema Bib (Tbib). From the GCCA

$$\Psi_1: [\$/author_v] = [\$bib/authors/author[area="Database"]]$$

we have that $\delta_o = \$bib/authors/author$ and $T\delta_o = Tauthor$. Since the path $publications/books/book_v$ of $Tauthor_v$ is semantically related to the path $(Tbook.author_ref\rightarrow)^{-1}$ of $Tauthor$ (see example 4.1), we have that the path

$$\$bib/authors/author/(Tbook.author_ref\rightarrow)^{-1}$$

satisfies condition (2) of Definition 4.2. Thus, the path is relevant to V .

Definition 4.3: Let δ_s and δ_p be paths of T_s and T_p . Suppose that the type $T\delta_s$ of δ_s references the type $T\delta_p$ of δ_p through the link k . Then, we say that δ_s references the path δ_p through k .

Example 4.3: The path $\$bib/books/book$ references the path $\$bib/authors/author$ through the link $author_ref\rightarrow$. This follows from the fact that $Tbook$, the type of the path $\$bib/books/book$, references $Tauthor$, the type of the path $\$bib/authors/author$, through the link $author_ref\rightarrow$.

4.2 The View_Maintainer Algorithm

The View_Maintainer algorithm receives as input the XML source data and an update on base data. As in [2], we describe the algorithm operating on a single update, which can be an insertion, a deletion or a replacement. The insert operation $INSERT(\$target, \$child)$ adds the node $\$child$ as a child of $\$target$. The delete operation $DELETE(\$target, \$child)$ removes the node $\$child$, a child of $\$target$. The replace operation removes an existing subtree and adds a new one in its place. We assume that it can be implemented as a deletion followed by an insertion.

In outline, the major steps of the View_Maintainer algorithm are:

1. Obtain all different paths from the root of S to the updated objects that are relevant to V . If no relevant path exists, stop.
2. For each relevant path δ^* , do
 - 2.1. Select the updated objects, in δ^* , that are relevant to V . An updated object is *relevant* to V when its new state may cause a change to the state of V .
 - 2.2. Generate the set of maintenance statements for the relevant objects.
 - 2.3. Install the maintenance changes in V .

In the following sections we discuss the procedures used in Step 1 and 2.

4.3 The Procedure IdentifyRelevantPath

Let $\tau = updateop(\$target, \$child)$ denote an insertion or deletion operation, where $\$target$ is a node in $\$S/\delta$ and δ is a path of $Troots$.

The procedure $IdentifyRelevantPaths$ receives as input a path δ and updated nodes $\$target$ and $\$child$ and identifies all different paths from the root of S to $\$child$ that is relevant to V (see Definition 4.2). Table 1 shows all possible situations that the

procedure should verify if a relevant path exists. For each relevant path δ^* , the procedure also obtains the set of all pairs $\langle \$t, \$c \rangle$, where $\$c$ is a node in $\$/\delta^*$ that, as a consequence of the update τ , was deleted or inserted as a child of $\$t$. The set of updated nodes in $\$/\delta^*$, denoted UpdatedNodes, is also defined in Table 1.

Table 1. Cases of the IdentifyRelevantPath procedure

Case	relevant path (δ^*)	UpdatedNodes
(1) $\delta/\text{label}(\$child)$ is relevant to V .	$\delta / \text{label}(\$child)$	$\{ \langle \$target, \$child \rangle \}$
(2) δ references δ_r through k such that $\delta_r / k^{-1} / \text{label}(\$child)$ is relevant to V .	$\delta_r / k^{-1} / \text{label}(\$child)$	$\{ \langle \$target, \$child \rangle \}$
(3) $\delta / \text{label}(\$child)$ references δ_r through k such that δ_r / k^{-1} is relevant to V .	δ_r / k^{-1}	$\{ \langle \$t, \$child \rangle \mid \$t \text{ in } \$child/k \}$
(4) Exists a path δ_p such that $\delta = \delta_p / \delta_q$ where δ_p references a path δ_r through link k and $\delta_r / k^{-1} / \delta_q / \text{label}(\$child)$ is relevant to V .	$\delta_r / k^{-1} / \delta_q / \text{label}(\$child)$	$\{ \langle \$target, \$child \rangle \}$
(5) Exists a path δ_q such that $\delta / \text{label}(\$child) / \delta_q$ references a path δ_r through link k and δ_r / k^{-1} is relevant to V .	δ_r / k^{-1}	$\{ \langle \$t, \$c \rangle \mid \$c \text{ in } \$child / \delta_q \text{ and } \$t \text{ in } \$c/k \}$

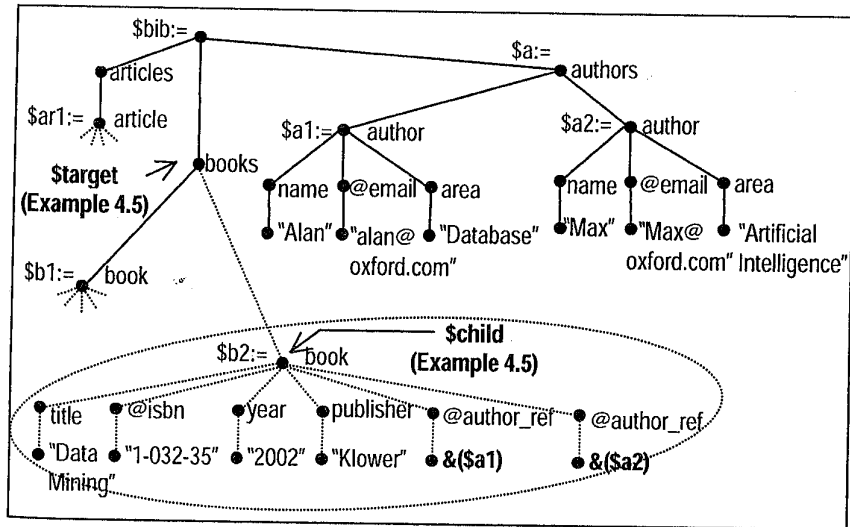


Fig. 3. XML data of Example 4.5

Example 4.5: Suppose that the base update $\tau = \text{INSERT}(\$target, \$child)$ is applied to the XML data in Figure 3, where $\$target$ is a node in $\$/bib/books$ and $\$child$ is the book element $\$/b2$. Therefore, the procedure *IdentifyRelevantPath* returns

$$\delta^* = \$bib/authors/author/(Tbook.author_{1_ref} \rightarrow)^{-1} \text{ and}$$

$$\text{UpdatedNodes} = \{ \langle \$a_1, \$b_2 \rangle, \langle \$a_2, \$b_2 \rangle \}.$$

4.4 The Procedure CheckSelectConditions

The procedure CheckSelectConditions selects the set of pairs $\langle \$t, \$c \rangle$ in updatedNodes which are relevant to V . A pair $\langle \$t, \$c \rangle$ is relevant to V when the new state of the base source may cause a change to the state of V . By analyzing the CAs of V , we establish, at view definition time, the conditions for efficiently checking the relevance of a given pair of updated nodes in a given relevant path. Table 2 shows the conditions for checking the relevance of a pair $\langle \$t, \$c \rangle$ in a relevant path δ^* . The cases in Table 2 correspond to the conditions in Definition 4.2 that the relevant path may satisfy.

Example 4.6: Consider the relevant path $\$bib/authors/author/(Tbook.author_ref \rightarrow)^{-1}$. From the GCCA $[\$V/author,] \equiv [\$bib/authors/author[area="Database"]]$ we have that $\delta_0 = \$bib/authors/author$; thus, $\$bib/authors/author/(Tbook.author_ref \rightarrow)^{-1}$ meets case 2 in Table 2. According to Table 2, a pair of updated nodes $\langle \$t, \$c \rangle$ in the relevant path $\$bib/authors/author/(author_ref \rightarrow)^{-1}$ is relevant to V iff $\$t/area="Database"$ or $old(\$t)/area="Database"$. For the update τ in Example 4.5, where UpdatedNodes = $\{ \langle \$a_1, \$b_2 \rangle, \langle \$a_2, \$b_2 \rangle \}$, we have:

1. $\langle \$a_1, \$b_2 \rangle$ is relevant to V since $\$a_1/area="Database"$ and
2. $\langle \$a_2, \$b_2 \rangle$ is not relevant to V , since $\$a_2/area="Artificial Intelligence"$ and $old(\$a_2)/area="Artificial Intelligence"$.

4.5 The Procedure GenerateMaintenanceStatement

The procedure GenerateMaintenanceStatement generates, for each pair of relevant updated nodes, the set of updates required for maintaining V . Table 3 defines the maintenance statement required for a relevant pair $\langle \$t, \$c \rangle$ in relevant path δ^* with respect to an insertion operation. The cases in Table 3, correspond to the conditions in Definition 4.2. Due to space limitation, we will discuss only insertion operations.

Example 4.7: Let $\langle \$t, \$c \rangle$ be a relevant pair in the relevant path $\$bib/authors/author/(Tbook.author_ref \rightarrow)^{-1}$. As of case 2 of Table 3, the updates required for the maintenance of V , with respect to the insertion of $\$c$ as a child of $\$t$, are:

IF $\$t/area="Database"$ and $old(\$t)/area="Database"$ /* case 2.1 */

{INSERT($\$target_v, \$child_v$) where:

- (i) $\$target_v = \$V/author_v[e-mail=\$t/email]/publications/books$.
The selExp $e-mail=\$t/e-mail$ selects the author in $\$V/author$ matching $\$t$, and is defined based on the MCA
 $\Psi_2: [Tauthor_v, \{email_v\}] \equiv [Tauthor, \{email\}]$.
- (ii) $\$child_v$ is a book $_v$ element generated by call CreateChild("book $_v$ ", Tbook $_v$, $\$c$, Tbook) which creates, based on the the PCAs of Tbook $_v$ and Tbook, a book $_v$ element corresponding to the book element $\$c$. From the PCAs of Tbook $_v$ and Tbook, CreateChild("book $_v$ ", Tbook $_v$, $\$c$, Tbook) returns:
 - $\langle book_v \rangle \langle title_v \rangle \$c/title/text() \langle /title_v \rangle$ (from Ψ_8)
 - $\langle isbn_v \rangle \$c/isbn/text() \langle /isbn_v \rangle$ (from Ψ_9)
 - $\langle year_v \rangle \$c/year/text() \langle /year_v \rangle$ (from Ψ_{10})
 - $\langle publisher_v \rangle \$c/publisher/text() \langle /publisher_v \rangle$ (from Ψ_{11})

```

    </bookv>}
  IF  $\$/\text{area}=\text{"Database"}$  and  $\text{old}(\$/\text{area}=\text{"Database"}$  /* case 2.2 */
    {INSERT( $\$/\text{child}_v$ ) where  $\$/\text{child}_v$  is an  $\text{author}_v$  element generated by call
    CreateChild("authorv", Tauthorv,  $\$/$ , Tauthor), which creates, based on the
    the PCAs of Tauthorv and Tauthor, the  $\text{author}_v$  element corresponding to the
    author element  $\$/$ }
  IF  $\$/\text{area} \neq \text{"Database"}$  and  $\text{old}(\$/\text{area}=\text{"Database"}$  /* case 2.3 */
    {DELETE ( $\$/\text{child}_v$ ) where  $\$/\text{child}_v = \$/\text{author}_v[\text{e-mail}=\$/\text{e-mail}]$ }

```

Example 4.8: Let τ be the update in Example 4.5. The relevant pair $\langle \$a_1, \$b_2 \rangle$ in $\$/\text{bib}/\text{authors}/\text{author}/(\text{Tbook.author_ref} \rightarrow)-1$ meets case 2.1 in Example 4.7. The procedure generates the maintenance statement, INSERT($\$/\text{target}_v, \$/\text{child}_v$) where:

- (i) $\$/\text{target}_v = \$/\text{author}[\text{e-mail}=\text{"alan@oxford.com"}]/\text{publications}/\text{books}$
- (ii) $\$/\text{child}_v = \langle \text{book}_v \rangle \langle \text{title}_v \rangle \text{"Data Mining"} \langle \text{title}_v \rangle$
 $\langle \text{isbn}_v \rangle 1-00305 \langle \text{isbn}_v \rangle \langle \text{year}_v \rangle 2002 \langle \text{year}_v \rangle$
 $\langle \text{publisher}_v \rangle \text{Klower} \langle \text{publisher}_v \rangle \langle \text{book}_v \rangle$.

The new state of the materialized view V , after the updates, is shown in Figure 4.

5 Conclusions

We proposed an algorithm for the incremental maintenance of XML views using view correspondence assertions. We showed how to analyze these assertions, at view definition time, to generate information that is used, at run time, to propagate updates to the base data sources to the materialized view. The use of this information brings significant advantage to our approach, when compared to previous approaches to XML view maintenance.

Table 2. Checking the relevance of $\langle \$/, \$c \rangle$ in the relevant path δ^*

Case	Conditions
Case1: δ^* is a prefix of δ_0 , where: (i) $\delta^* = e_1 / \dots / e_p, p \leq n$; (ii) $\$/$ in $\$/\text{e}_1 / \dots / \text{e}_{p-1}$; and (iii) $\$/c$ in $\$/\text{e}_1 / \dots / \text{e}_p$	exists $\$/e_1, \dots, \$/e_{p-2}$, ancestors of $\$/$, where $\$/e_k$ in $\$/\text{e}_1 / \dots / \text{e}_k, 1 \leq k \leq p-2$, such that $\text{selExp}_k(\$/e_k) = \text{true}, 1 \leq k \leq p-2$, and $\text{selExp}_{p-1}(\$/) = \text{true}$ and $\text{selExp}_p(\$/c) = \text{true}$.
Case2: δ^* is a prefix of δ_0 / δ_p , where: (i) $\delta_p = \delta_{p1} / \dots / \delta_{pm}$ is S.R. to the path $\delta_v = \delta_{v1} / \dots / \delta_{vm}$ by the PCAs $T_{\delta_0 / \delta_{p1}} \equiv T_v / \delta_{v1}$ and $T_{\delta_{pi} / \delta_{pi-1}} \equiv T_{\delta_i} / \delta_{i-1}, 1 \leq i \leq m-1$; (ii) $\delta_{pm} = f_1 / \dots / f_t / f_{t+1} / \dots / f_r$; and (iii) $\$/$ in $\$/\delta_0 / \delta_{p1} / \dots / \delta_{pm-1} / f_1 / \dots / f_t$, and $\$/c$ in $\$/\delta_0 / \delta_{p1} / \dots / \delta_{pm-1} / f_1 / \dots / f_t / f_{t+1}, 1 \leq t \leq r$	exists $\$/e_1, \dots, \$/e_n$, ancestors of $\$/$, where $\$/e_1$ in $\$/\text{e}_1$ and $\$/e_k$ in $\$/e_{k-1} / \text{e}_k, 2 \leq k \leq n$, such that $\text{selExp}_k(\$/e_k) = \text{true}, 1 \leq k \leq n-1$, and $(\text{selExp}_n(\$/e_n) = \text{true}$ or $\text{selExp}_n(\text{old}(\$/e_n)^{\text{old}}) = \text{true})$. (*) $\text{old}(\$/e_n)$ refers the old state of $\$/e_n$
Case3: δ^* is a prefix of δ_p / δ_q where (i) $\delta_p = e_1 / \dots / e_n, p < n$, and (ii) δ_q is a condition path in selExp_p	exists $\$/e_1, \dots, \$/e_p$, ancestors of $\$/$, such that $\$/e_k$ in $\$/\text{e}_1 / \dots / \text{e}_k, 1 \leq k \leq n$, and $\text{selExp}_k(\$/e_k) = \text{true}, 1 \leq k \leq p-1$

Table 3. Maintenance statements for insertions operations (INSERT(\$t, \$c))

Case	Maintenance Statements
Case 1 (as in Table 2)	FOR \$e _n in \$c / e _{p-1} {selExp _{p-1} } / ... / e _n {selExp _n } DO INSERT(\$target _v , \$child _v) where: \$target _v = \$V and \$child _v = CreateChild ^(*) ("v", T _v , \$e _n , T _{δo}) (*creates an element v corresponding to \$e _n)
Case 2 (as in Table 2)	Case 2.1: selExp _n (\$e _n) = true and selExp _n (old(\$e _n)) = true FOR \$f _i in \$c / f _{v-2} / ... / f _i DO {INSERT(\$target _v , \$child _v) where: \$child _v = CreateChild("v _m ", T _{v_m} , \$f _i , T _{f_i}) and \$target _v = \$V / step ₁ / step ₁ / ... / step _{m-1} where (i) step ₀ = v {selExp _i } ^(*) ; (ii) step _k , 1 ≤ k ≤ m-1, is defined as follows: IF δ _k is single-valued {step _k = δ _k } ELSE {step _k = δ _k {selExp _k } ^(*) }; (*selExp _v is defined based on the MCA of T _v with T _{δo} . Given the MCA [T _v , {y ₁ , ..., y _i }] = [T _{δo} , {x ₁ , ..., x _i }], selExp _v = [y ₁ = \$e _n /x ₁ AND ... AND y _i = \$e _n /x _i]. (**) Given [T _{δ_k} , {y ₁ , ..., y _i }] = [T _{δ_k} , {x ₁ , ..., x _i }], the MCA of T _{δ_k} with T _{δ_k} , and \$p _k the ancestor of \$target s.t \$p _k in \$S/δ ₁ /.../δ _{p_k} , then selExp _{p_k} = [y ₁ = \$p _k /x ₁ AND ... AND y _i = \$p _k /x _i]
	Case 2.2: selExp _n (\$e _n) = true and selExp _n (old(\$e _n)) = false INSERT (\$V, \$child _v) where \$child _v = CreateChild ("v", T _v , \$e _n , T _{δo}).
	Case 2.3: selExp _n (\$e _n) = false and selExp _n (old(\$e _n)) = true DELETE (\$V, \$child _v) where \$child _v = v {selExp _i }
Case 3 (as in Table 2)	Case 3.1: selExp _p (\$e _p) = true and selExp _p (old(\$e _p)) = false For \$e _n in \$e _p / e _{p-1} {selExp _{p-1} } / ... / e _n {selExp _n } do INSERT (\$V, \$child _v) where \$child _v = CreateChild ("v", T _v , \$e _n , T _{δo}) Case 3.2: selExp _p (\$e _p) = false and selExp _p (old(\$e _p)) = true For \$e _n in \$e _p / e _{p-1} {selExp _{p-1} } / ... / e _n {selExp _n } do DELETE (\$V, \$child _v) where \$child _v = v {selExp _i }

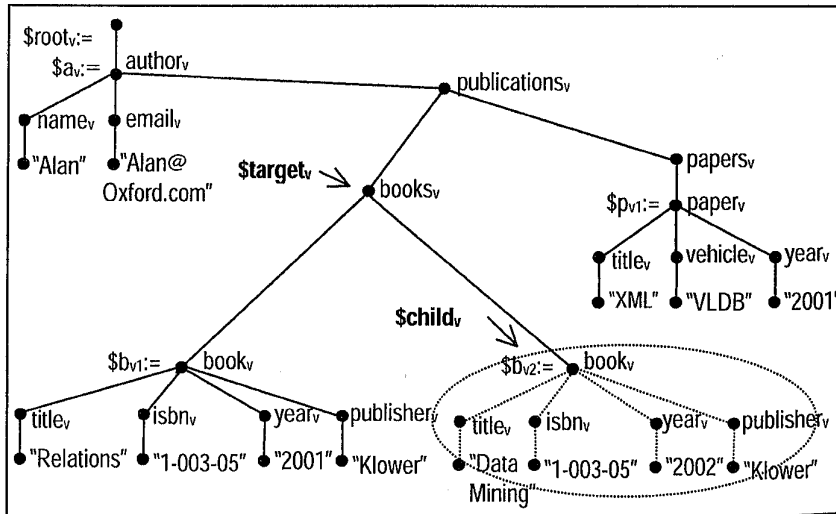


Fig. 4. The new state of the view

References

- [1] Ali, M.A., Fernandes, A.A., Paton, N.W.: Incremental Maintenance for Materialized OQL Views. In Proc. DOLAP (2000) 41–48
- [2] 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
- [3] 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
- [4] Ceri, S., Widom, J.: Deriving production rules for incremental view maintenance. In Proceedings of the International Conference on Very Large Databases (1991) 577–589
- [5] 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)
- [6] Fernandez, M., Morishima, A., Suci, D., Tan, W.: Publishing Relational Data in XML: the SilkRoute Approach. *IEEE Trans on Computers*, 44(4). (2001) 1–9
- [7] Gupta, A., Mumick, I.S.: Maintenance of Materialized Views: Problems, Techniques, and Applications. In *IEEE Bulletin on Data Engineering*, 18(2).(1995)3–18
- [8] Gupta, A., Mumick, I.S., Subrahmanian, V.S.: Maintaining Views Incrementally. In SIGMOD (1993) 157–166
- [9] 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)
- [10] 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).