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
Efficient Maintenance of XML Views
Using View Correspondence Assertions

Vânia Maria Ponte Vidal\textsuperscript{1} and Marco Antonio Casanova\textsuperscript{2}

\textsuperscript{1} Dept. Computação, Universidade Federal do Ceará
Fortaleza, CE – Brasil
vvidal@lia.ufc.br
\textsuperscript{2} Dept. 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 at al [5] proposed an algebraic solution approach based on the XAT XML algebra.

© Springer-Verlag Berlin Heidelberg 2003
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 \( Blb \), where:

- \( Blb \) is the type of the root element and contains a books element of type \( Tbooks \), an authors element of type \( Tauthors \), and an articles element of type \( Tarticles \);
- \( Tauthors \) contains a sequence of zero or more author elements of type \( Tauthor \);
- \( Tauthor \) 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 \( Tbook \) contains a reference to an author element of type \( Tauthor (Tauthor) \).

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) \( T_1 \) contains an element \( e \) whose type is \( T_1 \), denoted \( e^\dagger T_1 \rightarrow T_2 \); (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 \( (e \rightarrow)^\dagger T_1 \rightarrow T_2 \).

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 \( Se_1 \) of \( T_1 \), the path expression \( $e_1/e \) returns all instances of \( T_2 \) referenced in \( $e_1/e \).

Conversely, given an instance \( $e_2 \) of \( T_2 \), the path expression \( $e_2/1 \) returns the instance \( $e_2 \) of \( T_2 \) such that \( $e_1 \) in \( $e_2/1 \).

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 \( 1_i:T_1 \rightarrow T_{i+1} \), for \( i=1,...,n-1 \), where \( T_i \) are types of an XML schema. Therefore \( \delta_i = 1_1/\ldots/1_i/\ldots/1_{n-1} \) 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 \( T_0 \) of the path \( \delta \) is \( T_0 \).
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.
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]:

$$\Psi_1: [V/author,] = [Bib/authors/author[area = "Database"]]$$

specifies that $V/author$ 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

$$\Psi_2: [Tauthor, \{email,\}] = [Tauthor, \{@email\}]$$

specifies that an author $Sa$, in $V/author$, matches an author $Sa$ in $Bib/authors/author$ iff $Sa/email/text() = Sa/@email/text()$.

**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 $Tauthor$ and $Tauthor$:

$$\Psi_3: [Tauthor, name,] = [Tauthor, name]$$

$$\Psi_4: [Tauthor, email,] = [Tauthor, @email]$$

$$\Psi_5: [Tauthor, publications/books/book,] = [Tauthor, (Tbook.author_ref->)^{-1}]$$

$$\Psi_6: [Tauthor, publications/papers/paper,] = [Tauthor, (Tarticle.author_ref->)^{-1}]$$

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

Given an instance $Sa$ of $Tauthor$, if there is an instance $Sa$ of $Tauthor$ such that $Sa, Sa$, then:

- $Sa$ contains an element $name$, such that $Sa, name/text() = Sa/name/text()$ (from $\Psi_3$);
- $Sa$ contains an element $email$, such that $Sa, email/text() = Sa/@email/text()$ (from $\Psi_4$);
- $Sa$ contains an element publications, such that
  - $Sa, publications/books/book, = Sa/(Tbook.author_ref->)^{-1}$ (from $\Psi_5$);
  - $Sa, publications/papers/paper, = Sa/(Tarticle.author_ref->)^{-1}$ (from $\Psi_6$).

In case of $\Psi_5$, since $Tbook$ is a complex type, we also introduce the following MCA to match instances of $Tbook$ with instances of $Tbook$:

$$\Psi_7: [Tbook, \{isbn,\}] = [Tbook, \{@isbn\}]$$

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

$$\Psi_8: [Tbook, title,] = [Tbook, title]$$

$$\Psi_9: [Tbook, isbn,] = [Tbook, @isbn]$$

$$\Psi_{10}: [Tbook, year,] = [Tbook, year]$$

$$\Psi_{11}: [Tbook, publisher,] = [Tbook, publisher]$$
In case of $\Psi_3$, since $\text{Tpaper}$ is a complex type, we also introduce the MCA

$$\Psi_3: [\text{Tpaper}, \{\text{title},\}] = [\text{Tarticle}, \{\text{title}\}]$$

and the following PCAs obtained by matching $\text{Tpaper}$ and $\text{Tarticle}$:

$$\Psi_3: [\text{Tpaper}/\text{title},\] = [\text{Tarticle}/\text{title}]$$
$$\Psi_4: [\text{Tpaper}/\text{vehicle},\] = [\text{Tarticle}/\text{vehicle}]$$
$$\Psi_5: [\text{Tpaper}/\text{year},\] = [\text{Tarticle}/\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

$$\Psi: [SV/v ] = [SS/\text{PathExp}],$$

where $S$ is the base source and $\text{PathExp} = e_1[/selExp_1]/.../e_n[/selExp_n]$, where $e_1/.../e_n$ is a path of the root type of $S$ (Trees), and $selExp_k, 1 \leq k \leq n$, is a predicate expression [12]. The GCCA $\Psi$ specifies that $SV/v$ and $SS/\text{PathExp}$ have the same set of real world objects. The sub-elements of $v$ are specified by the PCA of $T_v$ with $T_e$, as discussed in Section 3.

Definition 4.1: Let $\delta_o$ and $\delta_p$ be paths of $T_o$ and $T_p$. We say that $\delta_o$ is semantically related to $\delta_p$, denoted $\delta_o \equiv \delta_p$, iff there are paths $\delta_o, ..., \delta_{o_n}$, with $\delta_o = \delta_{o_1}/.../\delta_{o_n}$, and paths $\delta_p, ..., \delta_{p_m}$, with $\delta_p = \delta_{p_1}/.../\delta_{p_m}$, such that

$$[T_o/\delta_{o_1}] = [T_p/\delta_{p_1}],$$
$$[T_o/\delta_{o_{i+1}}] = [T_p/\delta_{p_{i+1}}], 1 \leq i \leq m-1.$$

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 $\Psi_5: [\text{Tauthor/publications/books/book}] = [\text{Tauthor/ (Tbook/author_ref->)}]$, we have that the path $\text{publications/books/book}$ of $\text{Tauthor}$ is semantically related to the path $\text{(Tbook/author_ref->)}$ of $\text{Tauthor}$.

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

$$\Psi: [SV/v ] = [SS/c/[selExp_1]/.../e_n/[selExp_n]].$$

Let $\delta_o = e_1/.../e_n$ where $T_0 = T_e$. We say that a path $\delta$ of the root type $T_o$ of $S$ is relevant to $V$ if $\delta$ satisfies one of the following conditions:

1. $\delta$ is a prefix of $\delta_o$,
2. $\delta$ is a prefix of $\delta_p/\delta_q$, where path $\delta_p$ of $T_o$ is S.R. to the path $\delta_q$ of $T_v$ ($\delta_p \equiv \delta_q$),
3. $\delta$ is a prefix of $\delta_p/\delta_q$, where $\delta_p = e_1/.../e_p$, with $p < n$, and $\delta_q$ is a condition path in $\text{selExp}_p$. 
Example 4.2: Let \$bib/authors/author/(Tbook.author_ref→)^{-1}\$ be a path of the root type of the schema Bib (\$bib\). From the GCCA

\[ \Psi_{T_{\text{V}}}: [SV/\text{author},]=[$\text{bib}/\text{authors}/\text{author}[\text{area}=\text{"Database"}]] \]

we have that \( \delta_{\text{s}}=\text{bib}/\text{authors}/\text{author} \) and \( T\delta_{\text{s}}=\text{Tauthor} \). Since the path publications/books/book of \text{Tauthor} is semantically related to the path \( (Tbook.author_ref→)^{-1} \) of \text{Tauthor} (see example 4.1), we have that the path

\[ \text{bib}/\text{authors}/\text{author}/(\text{Tbook.author_ref→})^{-1} \]

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

Definition 4.3: Let \( \delta_{\text{s}} \) and \( \delta_{\text{p}} \) be paths of \( T_{\text{s}} \) and \( T_{\text{p}} \). Suppose that the type \( T\delta_{\text{s}} \) of \( \delta_{\text{s}} \) references the type \( T\delta_{\text{p}} \) of \( \delta_{\text{p}} \) through the link \( k \). Then, we say that \( \delta_{\text{s}} \) references the path \( \delta_{\text{p}} \) through \( k \).

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

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 \text{INSERT}(\text{$\text{target}$, \$child$}) adds the node \$child$ as a child of \$target$. The delete operation \text{DELETE}(\text{$\text{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 \( \delta' \), do
   2.1. Select the updated objects, in \( \delta' \), 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 \( \tau = \text{UPDATE}(\text{$\text{target}$, \$child$}) \) denote an insertion or deletion operation, where \$target$ is a node in \$S/\delta$ and \( \delta \) is a path of \text{Troots}.

The procedure \text{IdentifyRelevantPaths} receives as input a path \( \delta \) 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 \( \delta' \), the procedure also obtains the set of all pairs \(<s_l, s_c>\), where \( s_c \) is a node in \( S/S' \) that, as a consequence of the update \( \tau \), was deleted or inserted as a child of \( s_l \). The set of updated nodes in \( S/S' \), denoted \( UpdatedNodes \), is also defined in Table 1.

<table>
<thead>
<tr>
<th>Case</th>
<th>relevant path (( \delta' ))</th>
<th>UpdatedNodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( \delta / \text{label}(S\text{child}) ) is relevant to ( V ).</td>
<td>( \delta / \text{label}(S\text{child}) )</td>
</tr>
<tr>
<td>2</td>
<td>( \delta ) references &amp; through ( k ) such that ( \delta / k^{-1} \text{label}(S\text{child}) ) is relevant to ( V ).</td>
<td>( \delta / k^{-1} \text{label}(S\text{child}) )</td>
</tr>
<tr>
<td>3</td>
<td>( \delta / \text{label}(S\text{child}) ) references &amp; through ( k ) such that ( \delta / k^{-1} ) is relevant to ( V ).</td>
<td>( \delta / k^{-1} )</td>
</tr>
<tr>
<td>4</td>
<td>( \delta _c ) exists a path ( \delta _c ) such that ( \delta / \delta _c / \delta _c ) where ( \delta _c ) references a path ( \delta ) through node ( k ) and ( \delta / k^{-1} \text{label}(S\text{child}) ) is relevant to ( V ).</td>
<td>( \delta / k^{-1} \text{label}(S\text{child}) )</td>
</tr>
<tr>
<td>5</td>
<td>( \delta _c ) exists a path ( \delta _c ) such that ( \delta _c / \text{label}(S\text{child}) \delta _c ) references a path &amp; through link ( k ) and ( \delta / k^{-1} ) is relevant to ( V ).</td>
<td>( \delta / k^{-1} )</td>
</tr>
</tbody>
</table>

![Fig. 3. XML data of Example 4.5](image)

**Example 4.5:** Suppose that the base update \( \tau = \text{INSERT}(<\text{target}, S\text{child}> \) is applied to the XML data in Figure 3, where \( S\text{target} \) is a node in \( S\text{bib}/\text{books} \) and \( S\text{child} \) is the book element \( S\text{book} \). Therefore, the procedure \( \text{IdentifyRelevantPath} \) returns \( \delta' = S\text{bib}/\text{authors}/\text{author}(\text{Tbook.}\text{author._ref} \rightarrow )^{-1} \) and \( UpdatedNodes = \{<\$a_1, S_b>, <\$a_2, S_b>\} \).
4.4 The Procedure CheckSelectConditions

The procedure CheckSelectConditions selects the set of pairs <$l$, $c$> in updatedNodes which are relevant to $V$. A pair <$l$, $c$> 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 <$l$, $c$> in a relevant path $\delta^*$. 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 $\$bib/authors/author(Tbook.author_ref->)$. From the GCCA [$V/author]=[$bib/authors/author[area="Database"]$], we have that $\delta^*=$\$bib/authors/author; thus, $\$bib/authors/author(Tbook.author_ref->)$ meets case 2 in Table 2. According to Table 2, a pair of updated nodes <$l$, $c$> in the relevant path $\$bib/authors/author(author_ref->)$ is relevant to $V$ iff $\$area="Database"$ or old($l$)/area="Database". For the update $\tau$ in Example 4.5, where UpdatedNodes=$\{<$s_1$, $b_1$>, <$s_2$, $b_2$>$\$, we have:

1. <$s_1$, $b_1$> is relevant to $V$ since $s_1$/area="Database" and
2. <$s_2$, $b_2$> is not relevant to $V$, since $s_2$/area="Artificial Intelligence" and old($s_2$)/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 <$l$, $c$> in relevant path $\delta^*$ 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 <$l$,$c$> be a relevant pair in the relevant path $\$bib/authors/author(Tbook.author_ref->)$. 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 $l$, are:

IF $l$/area="Database" and old($l$)/area="Database" /* case 2.1 */
{INSERT($\text{Target}_1$, $\text{Child}_2$) where:
(i) $\text{Target}_1=$$V/author, [e-mail=$l$/email]/publications/books,
The selExp e-mail=$l$/e-mail selects the author in $V/author matching $l$,
and is defined based on the MCA
$\Psi_2=[Tauthor, [email, ]]=Tauthor, [email]].$
$<$book, $>$
$<$title, $>$ $c$/title/text()</title> (from $\Psi_6$
$<$isbn, $>$ $c$/isbn/text()</isbn> (from $\Psi_7$
$<$year, $>$ $c$/year/text()</year> (from $\Psi_8$
$<$publisher, $>$ $c$/publisher/text()</publisher> (from $\Psi_9$)
Example 4.8: Let \( \tau \) be the update in Example 4.5. The relevant pair \(<\$a, \$b>\) in \($bib/authors/author/(Tauthor/author_ref->)\) meets case 2.1 in Example 4.7. The procedure generates the maintenance statement, INSERT($SV$, Schildv) where:

(i) Schildv = $SV/author[e-mail=alan@oxford.com]/publications/books
(ii) $SV/author[e-mail=alan@oxford.com]/publications/books

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 \(<\$t, \$c>\) in the relevant path \( \delta^* \)

<table>
<thead>
<tr>
<th>Case</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 1: ( \delta^* ) is a prefix of ( \delta_v ) where:</td>
<td></td>
</tr>
<tr>
<td>i) ( \delta^* = \delta_1 \ldots \delta_n ), postal;</td>
<td></td>
</tr>
<tr>
<td>(ii) ( \delta^* = \delta_1 \ldots \delta_n ), postal;</td>
<td></td>
</tr>
<tr>
<td>(iii) ( \delta^* = \delta_1 \ldots \delta_n ), postal;</td>
<td></td>
</tr>
<tr>
<td>Case 2: ( \delta^* ) is a prefix of ( \delta_v ) where:</td>
<td></td>
</tr>
<tr>
<td>(i) ( \delta_1 = \delta_2 \ldots \delta_v ), postal, ( \delta ) is S.R. to the path:</td>
<td></td>
</tr>
<tr>
<td>(ii) ( \delta_1 = \delta_2 \ldots \delta_v ), postal, ( \delta ) is S.R. to the path:</td>
<td></td>
</tr>
<tr>
<td>(iii) ( \delta_1 = \delta_2 \ldots \delta_v ), postal, ( \delta ) is S.R. to the path:</td>
<td></td>
</tr>
<tr>
<td>Case 3: ( \delta^* ) is a prefix of ( \delta_v ) where:</td>
<td></td>
</tr>
<tr>
<td>i) ( \delta^* = \delta_1 \ldots \delta_n ), postal, and</td>
<td></td>
</tr>
<tr>
<td>(ii) ( \delta^* = \delta_1 \ldots \delta_n ), postal, and</td>
<td></td>
</tr>
<tr>
<td>(iii) ( \delta^* = \delta_1 \ldots \delta_n ), postal, and</td>
<td></td>
</tr>
</tbody>
</table>
### Table 3. Maintenance statements for insertions operations (INSERT(St, Sc))

<table>
<thead>
<tr>
<th>Case</th>
<th>Maintenance Statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 1</td>
<td>FOR $s_0$ in $Sc/f_2$/$f_3$/.../$f_n$ DO INSERT(Target, $Schild$, $Schild$) where: $Target$ = $S$ and $Schild$ = CreateChild($&quot;v&quot;$, $T_0$, $S_0$, $T_0$)</td>
</tr>
<tr>
<td>Case 2</td>
<td>FOR $s_0$ in $Sc/f_2$/$f_3$/.../$f_n$ DO INSERT(Target, $Schild$, $Schild$) where: $Schild$ = CreateChild($&quot;v&quot;$, $T_0$, $S_0$, $T_0$) and $Target$ = $S$/$step_0$/.../$step_n$ where (i) $step_0$ = $v$ (setExp); (ii) $step_0$. $1$ where $i &lt; n$, is defined as follows: IF $\delta_s$ is single-valued (step = $\delta_s$) ELSE (step = $\delta_s$ [setExp]); (iv) (setExp) is defined based on the MCA of $T_0$ with $T_0$. Given the MCA $[T_0, [y_1, ..., y_n]] = (T_0, [x_1, ..., x_n])$, setExp = $y_1$, AND ... AND $y_n$; (v) $y_1$, AND ... AND $y_n$ = $S$/$step_0$/.../$step_n$; then setExp = $y_1$, AND ... AND $y_n$ = $S$/$step_0$/.../$step_n$.</td>
</tr>
<tr>
<td>Case 3</td>
<td>FOR $s_0$ in $Sc/f_2$/$f_3$/.../$f_n$ DO INSERT(Target, $Schild$, $Schild$) where $Schild$ = CreateChild($&quot;v&quot;$, $T_0$, $S_0$, $T_0$)</td>
</tr>
</tbody>
</table>

Fig. 4. The new state of the view
References


