Updating XML Views: The Clean Source Theory

Ling Wang, Elke A. Rundensteiner, and Murali Mani

Department of Computer Science
Worcester Polytechnic Institute Worcester, MA 01609
{lingw, rundenst, mmani}@cs.wpi.edu

Abstract. XML has become a primary data model to wrap heterogeneous data sources using views. However, as is commonly known, views and thus also such XML views suffer from the view update problem. In this paper, we characterize the search space of all potential correct update mappings in the context of the source data also being XML. This is achieved by proposing the notion of a *source* and of a *clean source* for a given XML view element. Further, we introduce the *clean source theory* to determine whether a given view update mapping is correct. This now serves as a solid theoretical foundation for developing practical algorithms towards update translatability checking. We also provide a concrete case study that applies this theory to assess the translatability of XML view updates with different update types.

1 Introduction

1.1 Motivation

Most systems support views of the database to enhance logical data independence and to simplify the user interface by shielding users from data that are of no interest to them. With the growing popularity of XML, it has become the primary data model to wrap heterogeneous data sources using views. In addition, native XML management systems [12, 16, 29] and XML-relational systems [7, 15, 29] are beginning to support the creation of XML views that wrap the underlying often large repositories of XML or relational data. For these XML views to become first class citizens, users must be able to apply both query as well as update operations to them. These operations on the virtual views must be translated into corresponding operations on the respective underlying data storage. Most systems support at best querying against such wrapper views, while updating remains largely un-addressed [7, 12, 15, 16].

Update operations against such wrapper views are essential. First, without the capability of updating the stored data, the stored data may become quickly out of date and less valuable. Second, supporting updates through the XML view will provide users with a uniform interface (XML and XQuery), no matter what the underlying data storage is, nor which query and update language to be used on the underlying storage system.

However, supporting updates through views is challenging. The first question is to decide if such a mapping even exists? The second question concerns the ambiguity of the update semantics. In other words, when such a mapping does exist, it may not be unique.

Three practical approaches have been used to address the view update problem in both relational and XML context. One is to regard the underlying database and view as abstract data types, with the update operations explicitly predefined by the DBA [17, 18, 20]. The second determines a unique or a small set of update translations based on the syntax and semantics of the view definition, i.e., a compile-time decision process [1, 5, 6, 13, 14, 22]. The third approach performs run-time translation [21]. It transforms the view update problem into a constraint satisfaction problem (CSP), with an exponential time complexity in the number of constraint variables.

For either of these approaches we need a theory that characterizes precisely the conditions under which a mapping from XML view updates into updates on the base data storage is correct [10, 27, 28]. In fact, the purpose of this paper is to develop such a theory within the context of the XML data model, namely, both view and base are now in XML. Our theory aims to assess whether a mapping from an XML view update to a base XML update is correct. This problem is challenging for the fact that XML is a non-normalized hierarchical data model, which is inherently more complex than the normalized flat relational data model.

The fundamental question about what a correct update mapping entails has been addressed in earlier works [2, 9, 10]. In [2, 9], a complementary theory is proposed that requires a correct mapping to avoid *view side effects* as well as *database side effects*. That is, for a translation to be considered correct, it cannot affect any part of the database that is "outside" the view. This correctness criteria, however, is too restrictive to be practical. [10] relaxes this condition to only require that no *view side effect* occurs. In other words, a translation is correct as long as it corresponds exactly to the specified update, and it does not affect anything else in the view. The second is commonly used by most research projects [1, 5, 6, 13, 14, 22] and commercial systems [3, 8, 19] alike. Our work in this paper also follows this criteria.

1.2 A Running Example

```
<Database>
(p1)        <publisher>
                <pubid>A01</pubid>
                <pubname> McGraw-Hill Inc. </pubname>
(p1.b1)         <book>
                    <bookid>98001</bookid>
                    <title>TCP/IP Illustrated</title>
                    <price>37.00</price>
                    <year>1997</year>
(p1.b1.r1)          <review>
                        <reviewid> 001 </reviewid>
                        <comment>A good book on network.</comment>
                    </review >
(p1.b1.r2)          <review>
                        <reviewid> 002 </reviewid>
                        <comment>Useful for advanced user.</comment>
                    </review >
                </book>
(p1.b2)         <book>
                    <bookid>98003</bookid>
                    <title>Data on the Web</title>
                    <price>48.00</price>
                    <year>2004</year>
                </book>
            </publisher>
(p2)        <publisher>
                <pubid>A02</pubid>
                <pubname> Simon & Schuster Inc. </pubname>
(p2.b1)         <book>
                    <bookid>98002</bookid>
                    <title>Programming in Unix</title>
                    <price>45.00</price>
                    <year>1985</year>
                </book>
            </publisher>
(p3)        <publisher>
                <pubid>B01</pubid>
                <pubname> Prentice-Hall Inc. </pubname>
            </publisher>
</Database>
                                Bib.xml
```

Fig. 1. An example XML document *bib.xml*

Example 1. As a running example, Fig. 3 shows an XML (virtual) view defined by an XQuery (Fig. 2) over an XML document (Fig. 1). For illustration purposes, we encode the complex elements of the XML document and the result view by the Dewey order encoding. Now assume we want to delete a book (`vb1`) from *BookView.xml* (Fig. 3). The following three mappings into updates on the underlying base XML document may be potential solutions:

– delete review (`p1.b1.r1`) from *bib.xml* (1)

```
<BookView>

FOR  $publisher IN document("default.xml")/Database/publisher,
        $book IN $publisher/book
WHERE ($book/price<50.00) AND ($book/year > 1990)
REURN {
        <book>
            $book/bookid, $book/title, $book/price,
                <publisher>
                        $publisher/pubid, $publisher/pubname
                </publisher>,
                FOR $review IN $book/review
                RETURN{
                 <review>
                        $review/reviewid,
                        <comment>
                                $review/comment/text()
                        </comment>
                </review>}
        </book>},

FOR  $p IN document("default.xml")/Database/publisher
REURN {
        <publisher>
                $p/pubid, $p/pubname
        </publisher>}

</BookView>
```

Fig. 2. An XML view definition (*BookView.qlt*) over XML document in Fig. 1

```
<BookView>
(vb1)    <book>
             <bookid>98001</bookid>
             <title>TCP/IP Illustrated</title>
             <price>37.00</price>
(vb1.vp1)  <publisher>
                <pubid>A01</pubid>
                <pubname> McGraw-Hill Inc. </pubname>
              </publisher>
(vb1.vr1)   <review>
                <reviewid> 001 </reviewid>
                <comment> A good book on network.</comment>
              </review >
(vb1.vr2)  <review>
                <reviewid> 002 </reviewid>
                <comment>Useful for advanced user.</comment>
             </review >
         </book>
(vb2)  <book>
              <bookid>98003</bookid>
              <title>Data on the Web</title>
              <price>48.00</price>
(vb2.vp1)   <publisher>
                <pubid>A01</pubid>
                <pubname> McGraw-Hill Inc. </pubname>
              </publisher>
        </book>
(vp1)  <publisher>
            <pubid>A01</pubid>
            <pubname> McGraw-Hill Inc. </pubname>
        </publisher>
(vp2)  <publisher>
            <pubid>A02</pubid>
            <pubname> Simon & Schuster Inc </pubname>
        </publisher>
(vp3)  <publisher>
            <pubid>B01</pubid>
            <pubname> Simon & Schuster Inc </pubname>
        </publisher>
<BookView>
```

Fig. 3. The XML view (*BookView.xml*) defined by query in Fig. 2

– delete publisher (`p1`) from *bib.xml* (2)
– delete book (`p1.b1`) from *bib.xml* (3)

Intuitively we would like to choose the third one. In fact, we can observe that the following two conditions hold: (i) The book (`vb1`) actually does not occur in the *re-generated view*, which is created by re-applying the view query on the updated *bib.xml*. (ii) The XML view we are expecting is equal to the re-generated view. That is, we get the exact view we wanted.

The first choice is not correct since the book information (*bookid, title, price*) still appears in the view. That means condition (i) above is violated. The second choice is not correct either. The re-generated view will not only miss the book (`vb1`) but also the publisher (`vp1`). This is not what we had intended to do. □

The three choices in our example represent three typical categories of translations. The first class of mappings does not even perform the update (choice (1)). The second class of mappings performs the update but causes view side effects (choice (2)). The third class of mappings *correctly performs* the update (choice (3)).

We would like to identify the search space of potential correct update mappings. For efficiency, we would like to minimize this search space as much as possible. Intuitively, the update mapping has to *perform* the view update [10]. In other words, the element we want to delete cannot stay in the re-generated view and the new element we want to insert has to appear in the re-generated view. By this criteria, choice (1) is out of our search space.

Now all the update mappings in our search space can achieve the desired view update. We would also like to filter out choice (2) to avoid the *view side effect*. One way to identify this side effect is to compare the re-generated view with the view we are expecting to identify differences. This then constitutes the side effect. This is however not always practical since the view computation and the view comparison can be rather expensive, and the search space could be potentially very large.

1.3 The Clean Source Theory for Update Translatability

The issues mentioned in the running example, namely (i) how to identify the search space of potential correct translations and (ii) what is the practical way of determining a correct update translation, are fundamental for the XML view update problem.

As the first work to consider the view update problem in the context of XML views defined over XML documents, we aim to address these XML view updating issues. We first propose the concept of *source* of a given XML view element. This characterizes the minimized search space of potential correct update mappings. Then a *clean source theory* is introduced, which utilizes the concept of a *clean source* to determine whether a view update mapping is correct and has no view side effects.

The concepts of *source* and *clean source* have been first proposed in [10] for the relational context. They are used to characterize the conditions under which an update on a relational view published over a single relational table is translatable. Our earlier work in [27] extends this work into a *clean extended source theory*, which helps us to identify the correct update translation in the case when an XML view is defined over the relational database.

The work in this paper is now the first work to consider the view update problem in the context of the pure XML data model, namely, both the view and the base data are in XML. This is challenging problem because of two facts.

- Unlike the relational flat data model, XML is a hierarchical data model. This hierarchical structure can be a deep tree structure. Relational database, however, can only form a shallow tree structure as used by the *default XML views* [7, 15, 29]. Since the view query can freely define the new hierarchical structues, the difference between the view hierarchy and base XML hierarchy can thus potentially have many more translation possibilities. These choices can be "overlap" along the XML base hierarchy. Identifying the correct translation is thus harder to fulfill.
- XML is not normalized with relational databases generally are. The XML data can contain duplications. On the other hand, view query can eliminate this duplications by using the dinstinct-value function. This raise more challenge in the update translatability checking.

To solove these challenges, we need to properly bridge the two flexible hierarchical data structures, namely, the XML view hierarchy and the underlying XML document hierarchy. Using a typical syntax level query rewriting and the *Update Context Binding* of each schema node (Section 2.1), we guarantee to preserve this flexibility in update translation. This theory serves as a solid theoretical foundation for the study of the XML view update problem. For example, it can be used as guideline for developing algorithms of update translatability checking. It can also be used by practical XML view update translation mechanisms, such as commercial systems, as correctness criteria.

Contributions. In general, the contributions of this paper include:

- We characterize the *update translatability* problem in the context of the XML data model, namely, when XML views are defined over XML documents.
- We propose the concept of a *source* to characterize the search space for correct update translations.
- Using the concept of a *clean source*, we propose and prove the *clean source theory* for determining whether a given view update translation is correct.
- We provide concrete case studies illustrating how our proposed theory can be utilized to reason about the correctness of mapping updates for different update types.

Outline. The XML view update problem is formalized in Section 2. In Sections 3 and 4 we propose the *clean source theory* and its key concepts. Section 5 provides concrete case studies on applying our theory for different update types. Section 6 reviews related work while Section 7 provides our conclusions and future directions.

2 Preliminary

2.1 XML Views and Normalization

Let D denote an **XML document** comply with certain XML schema [23]. We associate each element in the XML document with its *schema node type*. An element is of a *simple type* if it is defined by *xsd:element* in the schema. A *tag type* element is defined by *xsd:complexType* but includes only *simple type* elements as its children. Otherwise, an element is of a *complex type*. For example, in the XML document in Fig. 1, both the *publisher* and the *book* are *complex type* elements, while the *review* is a *tag type* element. All the others are *simple type* elements.

The **XML view** V is defined by a **view definition** DEF over D. In our case, DEF is an XQuery expression (e.g. [24]) called a *view query*. In this paper, we assume the view queries follow the XQueryCore semantic [25]. We perform additional rewrite steps to satisfy the following conditions. (i) Each XPath in DEF is a one-step simple navigation. Multi-step XPaths are rewritten into FWR expressions with single-step XPaths. (ii) RETURN clauses include either variable bindings of any type or simple-step navigations of *simple type* and *tag type*.

Fig. 4 lists several examples to illustrate these rewriting steps. Note that we return *$review* in the query, which is a *tag type* element, instead of returning *$book/review*, which is a navigation from *complex type* elements above.

The purpose of this rewriting is to distinguish the schema node from its context by rewriting multi-step XPaths in RETURN clauses. We also want to enumerate all possible paths by rewriting multi-step XPaths in FOR clauses.

Let B_k denote a variable binding defined by FOR clauses in the view definition DEF. Let $I(B_k)$ denote the set of instances of B_k. Let $Vars()$ be a function to extract all variable bindings referred by DEF. For example, $Vars(BookView)$=

Example I:

```
FOR $root IN document("bib.xml")/Database
RETURN
    publisher/book[price < 70.00]/review
```

Example II:

```
FOR $review IN document("bib.xml")
    /Database/publisher/book[price < 70.00]/review
RETURN
    $review
```

$\Longrightarrow$

```
FOR $root IN document("bib.xml")/Database
RETURN {
    FOR $publisher IN $root/publisher
        $book IN $publisher/book
        $review IN $book/review
    WHERE $book/price < 70.00
    RETURN
        $review
}
```

Fig. 4. Examples in view query rewriting

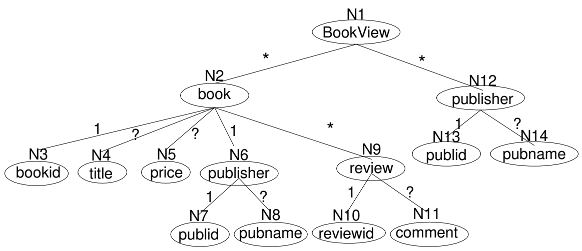

Fig. 5. The schema graph of view defined in Fig. 2

Node ID	UCBinding
N_1	
N_2	\$publisher,\$book
N_6	\$publisher,\$book
N_9	\$publisher,\$book,\$review
N_{12}	\$p

Fig. 6. UCBinding and UPBinding for nodes in Fig. 5

$\{B_1, B_2, B_3, B_4\}$ =$\{$*\$publisher,\$book,\$review,\$p*$\}$. And $I(B_1)$= I(*\$publisher*)=$\{\mathtt{p}_1, \mathtt{p}_2, \mathtt{p}_3\}$.

The XML view schema defined by the view query can be represented with a schema graph (see Fig. 5). It captures the hierarchical structure of the view result, and the cardinality between the schema nodes indicated by ***,?,1** for a (1:n), (1:[0,1]) and (1:1) relationship respectively [6, 11, 28].

We decorate each schema node n with an **Update Context Binding**, denoted by *UCBinding(n)*. It includes all the variable bindings defined up to n in DEF. Fig. 6 shows the *UCBindings* of all the schema nodes in the schema graph (Fig. 5). Note that we do not list the leaf nodes, since they always have exactly the same *UCBindings* as their parents. The *UCBindings* are extensively used in Section 3 to define the *generator* concept.

2.2 Updates and Correct Translation

Let $\mho$ be the domain of all view updates. Let $u \in \mho$ be an update on the view V with an *insertion* adding while a *deletion* removing an element from the XML view. A *replacement* replaces an existing view element with a new one. For illustration purposes, we use update primitives from [6] to represent update operations on the view as well as on the base XML documents.

Definition 1. *An update operation u is a triple (type; ref; delta), where* **type** *is the type of the operation: insert, delete or replace;* **ref** *is a simple path expression expressed using XPath that indicates where the update is to occur; and* **delta** *is the XML tree to be inserted, or in case of a replacement an atomic value, or empty in the case of delete [6].*

Updates are specified using path expressions to point to a set of target nodes in the XML tree at which the update is to be performed. For insertions and replacements, the update must also specify a *delta* containing the new values. [1] Several update examples on the *BookView.xml* are shown in Fig. 7.

[1] In our update translatability study, we only consider deletion and insertion. The replacement can be represented with a deletion followed by an insertion.

```
(type = insert;
ref = /book[bookid="98003"];
delta = {<review>
            <reviewid>001</reviewid>
            <comment>good</comment>
        </review>})

(type = delete;
ref = /book[title="TCP/IP Illustrated"])
```

Fig. 7. Examples of update primitives on *BookView.xml*

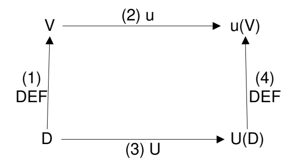

Fig. 8. Correct translation of view update to relational update

A **valid view update** is an insert, delete or replace operation that satisfies all constraints in the view schema, which can be either pre-defined or inferred from the view definition or the XML schema. Unless otherwise stated, all the updates considered in this paper are assumed to be valid.

Definition 2. *Let D be an XML document and V be a view defined over D by the view definition DEF. An XML document update sequence U on D is a* **correct translation** *of a valid update u on V iff (i) $u(DEF(D)) = DEF(U(D))$ and (ii) if $u(DEF(D)) = DEF(D) \Rightarrow U(D) = D$ (see [10]).*

A correct translation means the "rectangle" rules shown in Fig. 8 hold. Intuitively, this implies that the translated XML document updates exactly perform the view update, namely, without view side effects. In addition, if an update operation does not affect the view, then it should not affect the XML base either. This guarantees that any modification of the relational base is indeed done for the sake of the view. For example, the deletion of the book with bookid="98002" will not have a correct translation since this book does not appear in the view. The second criteria is guaranteed if the translation is done by query composition as commonly used by XML view management system [12, 16, 29, 7, 15]. Hence it generally can be achieved.

3 Source and Clean Source

To identify the search space and indicate the correct translations, we need a mapping from the view elements to the underlying XML document elements. This can be achieved by the definition of *UCBinding* of schema nodes (Section 2). First of all, the elements of instances of the *UCBinding* form the *generator* of the view element as defined below.

Definition 3. *Given an XML document D and an XML view V defined by DEF over D. Let v be a view element of V, whose schema node is n. Let $g = \{e_k \mid \forall B_k \in UCBinding(n), e_k \in I(B_k)\}$ and $DEF(g)$ generates v. We say g is the* **generator** *of v, and $\forall e_i \in g$ is a* **source-element** *in D of v.*

Intuitively, the *generator* includes the underlying XML document parts, which all together affect the occurrence and the content of v. Clearly, for a given view element v, it has one and only one generator. As an example, consider the *book* element (`vb1`) in *BookView.xml* of Fig. 3. Its schema node is $N2$ in Fig. 5. We have $UCBinding(N2) = \{\$publisher, \$book\}$ from Fig. 6. The generator of the book (`vb1`) in *BookView.xml* is g={`p1, p1.b1`} in *bib.xml*.

Below we define the *source* of a set of view elements V^0. Intuitively, a *source* includes the base XML part which are sufficient for the disappearance of V^0.

Definition 4. *Let V^0 be a set of view elements in a given XML view V. Let $G(V^0)$ be the set of generators of V^0 defined by $G(V^0) = \{g \mid g$ is a generator of a view-element in $V^0\}$. For each $g \in G(V^0)$, let $H(g)$ be some nonempty subset of g. Then $\cup_{g \in G(V^0)} H(g)$ is a* **source** *in D of V^0.*

For example, in *BookView.xml* (Fig. 3), consider V^0 as all the *reviews* of the *book* (`vb1`). We have $G(V^0) = \{g_1, g_2\}$, where $g_1 =$ {`p1, p1.b1, p1.b1.r1`}, $g_2 =$ {`p1, p1.b1, p1.b1.r2`}. Then $H(g_1)_1 =$ {`p1`}, $H(g_1)_2 =$ {`p1.b1`}, $H(g_1)_3 =$ {`p1.b1.r1`}. And $H(g_2)_1 =$ {`p1`}, $H(g_2)_2 =$ {`p1.b1`}, $H(g_2)_3 =$ {`p1.b1.r2`}. Any combination of $H(g_1)_i$ and $H(g_2)_j$ will be a source of V^0. For example, $S_1 =$ {`p1`}, $S_2 =$ {`p1.b1`} and $S_3 =$ {`p1.b1.r1, p1.b1.r2`}.

Any source of V^0 can achieve the delete of V^0. The translation achieving insertion of V^0 also has to be a source of V^0. The set of *sources* forms the *search space* for correct translations of deleting from or inserting V^0 into the view.

We aim to determine which ones are correct translations by Definition 2 in Section 2. The *clean source* below answers this question. Intuitively, update translation on a clean source, which is characterized by a source that is only referenced by the given view element itself, is a correct translation.

Definition 5. *Let D be an XML document. Let V^0 be a set of view elements in a given XML view V and S be a source in D of V^0. S is a* **clean source** *in D of V^0 iff $(\forall v \in V - V^0)$, $(\exists g)$ such that g is a generator in $(D - S)$ of v. Or, equivalently, S is a clean source in D of V^0 iff $(\forall v \in V - V^0)(S \cap g = \emptyset)$, where g is the generator of v.*

For instance, in *BookView.xml* of Fig. 3, consider V^0 as the book element (`vb1`). Two of its sources are $S_1 =$ {`p1.b1`} and $S_2 =$ {`p1`} in *bib.xml*. S_1 is a clean source of V^0, while S_2 is not. The reason is that S_2 is also a source of the view element *book* (`vb2`) in *BookView.xml*, which is outside of the book (`vb1`). In other words, if we achieve an update by deleting its publisher (`p1`), there will be a view side effect on the book (`vp2`)). This explains the reason of rejecting choice (2) in Example 1.

4 Clean Source Theory

We now establish a connection between the concept of a *clean source* introduced above and *update translatability* of different update types by introducing a series of theorems. Theorems 1 and 2 together form the **Clean Source Theory**. While Lemmas 1, 2 and 3 are used to prove Theorems 1 and 2.

We assume a view update u is a valid view update and expressed in update primitives (Definition 1 in Section 2). Using two examples below, we first explain the clean source theory intuitively.

Example 2.

```
u:      ( type=delete; ref = /book/review )                    ⟹

U1:     ( type=delete; ref = /publisher/book[price<50, year>1990]/review )
U2:     ( type=delete; ref = /publisher/book[price<50, year>1990])
U3:     ( type=delete; ref = /publisher )
```

In Example 2, the view update u deletes all the reviews. Thus $V^0 =$ {`vb1.vr1, vb1.vr2`} in the *BookView.xml*. This view update can be translated into three different base updates over *bib.xml*, namely, U1, U2 and U3. We say that U1 is a correct translation, since it deletes a clean source of V^0. That is, $S_1 =$ {`p1.b1.r1, p1.b1.r2`}. $U2$ deletes the source $S_2 =$

Example 3.

```
u: ( type = insert; ref = root;
     delta = <book>
                <bookid>98004</bookid>
                <title>Oracle</title>
                <price>48.00</price>
                <publisher>
                  <pubid>A04</pubid>
                  <pubname> Wesley </pubname>
                </publisher>
             </book> )
```

$\Longrightarrow$

```
U: ( type = insert; ref = root;
     delta = <publisher>
                <pubid>A04</pubid>
                <pubname> Wesley </pubname>
                <book>
                   <bookid>98004</bookid>
                   <title>Oracle</title>
                   <price>48.00</price>
                </book>
             </publisher> )
```

{`p1.b1`}. S_2 is not clean since it is also the source of book element (`vb1`) in *BookView.xml*, which belongs to $V - V^0$. Similarly, U_3 deletes the source $S_3 =$ {`p1`}, which is not clean. Therefore, intuitively an update translation of a deletion is correct as long as it deletes a clean source of the view part.

In Example 3, the view update u inserts a new book element (denoted as `vb3`) into *BookView.xml*. The translation U is not correct since it inserts a publisher (denoted as `p4`) into *bib.xml*, but this source $S =${`p4`} is not a clean source of `vb3`. The reason is that a new publisher (denoted as `vp4` in *BookView.xml*) will appear in the view, since S now also serves as a generator of `vp4`. Therefore, intuitively an update translation of insertion is correct as long as it inserts a clean source of the view part to be inserted, but does not form the generator for any other view element, namely, to insert a source-element for any other view element.

Following this intuition, we now explain the *clean source theory* formally. Let D denote an XML document. Let V^0 be a set of XML view elements in a given XML view V and let $v \in V^0$ indicate that v is a view element in V^0.

Lemma 1.
(a) S is a source in D of V^0 iff $DEF(D - S) \subseteq V - V^0$.
(b) S is a clean source in D of V^0 iff $DEF(D - S) = V - V^0$.

Proof. (a) *If.* Suppose $DEF(D - S) \subseteq V - V^0$ but S is not a source in D of V^0. Let $G(V^0)$ be the set of generators of V^0. From Definition 4, $\exists g \in G(V^0)$ be a generator of $v \in V^0$, such that $(\forall e_i \in g) \Rightarrow e_i \notin S$. That is, $e_i \in D - S$. Thus $v \in DEF(D - S)$. But, g is a generator of $v \in V^0$. That is $v \notin V - V^0$. Hence, we have $v \in DEF(D - S)$ and $v \notin V - V^0$. This is a contradiction with the hypothesis that $DEF(D - S) \subseteq V - V^0$.

Only if. Suppose S is a source in D of V^0 but $DEF(D - S) \nsubseteq V - V^0$. Then, $\exists v$ such that $(v \in DEF(D - S)) \wedge (v \in V^0)$. This implies that there is a generator g of $v \in V^0$ such that $\{e_i \mid e_i \in g\} \cap S = \emptyset$, contradicting the hypothesis that S is a source in D of V^0.

(b) *If.* Suppose $DEF(D - S) = V - V^0$ but S is not a clean source in D of V^0. From (a), S is a source in D of V^0. By Definition 5, $(\exists v \in V - V^0)$ such that there is no generator $g \in (D - S)$ of v. Hence we have $v \notin DEF(D - S)$, a contradiction.

Only if. Assume that S is a clean source in D of V^0. By (a), $DEF(D - S) \subseteq V - V^0$. Assuming $V - V^0 \nsubseteq DEF(D - S)$, that is, $(\exists v \in V - V^0)$ such that $(v \notin DEF(D - S))$. Then there is no generator $g \in (D - S)$ of v. Hence, by Definition 4, there is no source in $(D - S)$ of $v \in V - V^0$. This contradicts the hypothesis that S is a clean extended source in D of V^0. $\square$

Lemma 2. *Given an XML view V defined by DEF over XML document D. Let u and U be updates on V and D (respectively). Let $v \in V$. Then (U deletes a source of v and U does not insert a source-element of v) iff $v \notin DEF(U(D))$.*

Proof. Let $D' = U(D)$ and $T = D - D'$. Let $I'(B_x) \subset U(D)$ denote the updated instance of $I(B_x)$.

U deletes a source of $v \in V$
$\Longleftrightarrow T$ is a source in D of v
$\Longleftrightarrow DEF(D - T) \subseteq V - v$ (lemma 1)
$\Longleftrightarrow v \notin DEF(D - T)$
$\Longleftrightarrow v \notin DEF(D \cap D')$ since $D - T = D \cap D'$
$\overset{(1)}{\Longleftrightarrow}$ There is no generator of v in $(D \cap D')$.

U does not insert a source element of $v \in V$
$\overset{(2)}{\Longleftrightarrow} \forall B_x \in Vars(DEF)$, $\forall e_i \in I'(B_x) - I(B_x)$, $\not\exists e_j \in I'(B_y) - I(B_y)$ where $B_y \in Vars(DEF)$, $x \neq y$, such that $(e_1, ..., e_p)$ is a generator of v.

(1) and (2) hold iff there is no generator in $U(D)$ of v. The proposition then follows. □

Lemma 3. *Let u, U, V, D be as in Lemma 2. Let $v \in dom(V) - V$. Then U inserts source-elements of v iff $v \in DEF(U(D))$.*

Proof. U inserts source-elements of v
$\Longleftrightarrow (\exists B_x \in Vars(DEF), \exists e \in (I'(B_x) - I(B_x))$
(e is a source element in $U(D)$ of v)
$\overset{(1)}{\Longleftrightarrow} (\exists g = (e_1, ..., e_p) \in U(D))$($g$ is a generator of v).
$\Longleftrightarrow v \in DEF(I'(B_1), ..., I'(B_n)) = DEF(U(D))$.

(1) is proven as below:
If. Follows directly from Definition 4.
Only If. Assume that $g = (e_1, ...e_p)$ is a generator of v, but $\forall B_x \in Vars(DEF)$, $e_i \in I(B_x)$ instead of $e_i \in I'(B_x)$. Then $g \in \prod_{B_x \in UCBinding(n)}(I(B_x))$ and so $v \in DEF(I(B_1), ..., I(B_n))$, that is $v \in DEF(D)$, a contradiction with $v \in DEF(U(D))$. □

The following theorems form the core of the *clean source theory*. The intuition is that the XML document updates correctly translate a view update if and only if it deletes or inserts a *clean source* of the view element to be updated. In other words, a deletion or insertion is translatable as long as there is a clean source of the view element being deleted or inserted.

Theorem 1. *Let u be the deletion of a set of view elements $V^d \subseteq V$. Let τ be a translation procedure, $\tau(u, D) = U$. Then τ* **correctly translates** *u to D iff U deletes a clean source of V^d.*

Proof. By lemma 1(b), U deletes a clean source of V^d
$\Longleftrightarrow DEF(I(B_1) - T_1, ..., I(B_n) - T_n) = V - V^d = u(V)$
$\Longleftrightarrow DEF(U(D)) = u(V)$, since $I(B_i) - T_i = I'(B_i)$
$\Longleftrightarrow \tau$ correctly translates u to U. □

By Definition 2, a correct delete translation is one without any view side effect. This is exactly what deleting a clean source guarantees by Definition 5. Thus Theorem 1 follows.

Theorem 2. *Let u be the insertion of a set of view elements V^i into V. Let $V^- = V - V^i$, $V^u = V^i - V$. Let τ be a translation procedure, $\tau(u, D) = U$. Then τ* **correctly translates** *u to D iff (i) $(\forall v \in V^u)$(U inserts a source-element of v) and (ii) $(\forall v \in dom(V) - (V^u \cup V^-))$(U does not insert a source element of v).*

Proof. By Lemma 3, condition (i) iff $V^u \subseteq DEF(U(D))$.
Also, since $type(u) = insert$ and $type(U) = type(u)$, $DEF(U(D)) \supseteq V \supseteq V^-$.
Hence, $V^u \cup V^- \subseteq DEF(U(D))$.
By Lemma 3, condition (ii) iff $(dom(V) - (V^u \cup V^-)) \cap (DEF(U(D))) = \emptyset$.
Hence, $DEF(U(D)) \subseteq V^u \cup V^-$.
Thus, condition (i) and condition (ii) iff $DEF(U(D)) = V^- \cup V^u = u(V)$. That is τ correctly translates u to U. □

Since $dom(V) - (V^u \cup V^-) = (dom(V) - (V^i \cup V)) \cup (V^i \cap V)$, Theorem 2 indicates a correct insert translation is one without any duplicate insertion (insert a source of $V^i \cap V$) and any extra insertion (insert a source of $dom(V) - (V^i \cup V)$). That is, it inserts a clean source for the new view element.

5 Update Translatability Case Study

Using the *BookView.xml* defined in Fig. 2, we now perform case studies for different update types.

5.1 Case Study on Delete Operation

We describe the application of Theorem 1 using additional examples. In Example 4, u deletes all publishers in the *BookView.xml*. $U1$ is the only choice in our search space, which deletes a source S={`p1`} in *bib.xml*. But it is not a correct translation since S is not a clean source (S is also a source for view element (`vp1`) in *BookView.xml*).

Example 4.
```
 u:  ( type=delete; ref = /book/publisher )
U1: ( type=delete; ref = /publisher[book/price<50 and book/year>1990])  (NO)
```

In Example 5, u deletes all books in the *BookView.xml*. The search space includes two translations $U1$ and $U2$. $U1$ deletes a source $S1$={`p1.b1, p1.b2`} in *bib.xml*. It is a correct translation since $S1$ is a clean source. However, $U2$ does not correctly translate u. It deletes a source $S2$={`p1`} in *bib.xml*, but $S2$ is not a clean source. The reason is that $S2$ is also a source of another publisher element (`vp1`) in *BookView.xml*.

Example 5.
```
 u:  ( type=delete; ref = /book )
U1: ( type=delete; ref = /publisher/book[price<50 and year>1990])   (YES)
U2: ( type=delete; ref = /publisher )                                (NO)
```

In Example 6, u deletes the publisher in the *BookView.xml* with pubid="A01". $U1$ is the only choice in our search space, which deletes a source S = {`p1`} in *bib.xml*. But it is not a correct translation since S is not a clean source. S is also a source for view elements (`vb1`) and (`vb2`) in *BookView.xml*. However, a similar deletion in Example 7 has a correct translation. The reason is that the publisher with pubid="B01" in *bib.xml* is a clean source for the view element (`vp3`).

Example 6.

```
u:  ( type=delete; ref = /publisher[pubid='A01'] )
U1: ( type=delete; ref = /publisher[pubid='A01'] )                (NO)
```

Example 7.

```
u:  ( type=delete; ref = /publisher[pubid='B01'] )
U1: ( type=delete; ref = /publisher[pubid='B01'] )                (YES)
```

5.2 Case Study on Insert Operation

We now describe the application of Theorem 2 using additional examples. In Example 8, the view update u inserts a new review to the book (`vb1`) in *BookView.xml* as (`vb1.vr3`). U is the only choice in the search space. It inserts the review into the book (`p1.b1`) in *bib.xml* as `p1.b1.r3`. This is a correct translation since this new review (`p1.b1.r3`) is a clean source of the new review (`vb1.vr3`) in the view.

Example 8.

```
u: ( type = insert;
     ref = /book[title='TCP/IP Illustrated'];
     delta = <review>
               <reviewid>003</reviewid>
               <comment> good </comment>
             </review> )
```

$\Longrightarrow$

```
U: ( type = insert;
     ref = /publisher/book
          [price<50 and year>1990
           and title='TCP/IP Illustrated'];
     delta = <review>
               <reviewid>003</reviewid>
               <comment> good </comment>
             </review> )
```

Example 9.

```
u: ( type = insert;
     ref = root;
     delta = <publisher>
               <pubid>A04</pubid>
               <pubname> Wesley </pubname>
             </publisher> )
```

$\Longrightarrow$

```
U: ( type = insert;
     ref = root;
     delta = <publisher>
               <pubid>A04</pubid>
               <pubname> Wesley </pubname>
             </publisher> )
```

In Example 9, the view update u inserts a new publisher for the book (`vb1`) in *BookView.xml* as (`vp4`). U is the only choice in the search space. It inserts the new publisher into the *bib.xml* as `p4`. This is a correct translation since this new publisher `p4` is a clean source of the inserted publisher (`vp4`) in the view.

6 Related Work

Related Work on Update Translatability. The view update translatability problem has been studied in depth for the relational data model. An abstract formulation of the update translation problem is given by the *view complementary theorem* in [9, 2]. It uses the complement of a view to resolve the ambiguity in mapping between old and new database states. Based on the notion of a *clean source*, [10] presents an approach for determining the existence of update translations by performing a semantic analysis of the view definition.

The first interesting extension of these existing relational works is to tackle the problem of updating XML views published over a relational database [5, 6, 26–28]. Intuitively, this is more complex than their issues of a pure relational view update. Not only do all the problems in the relational context still exist in the XML semantics, but we also have to address the new update issues introduced by the XML hierarchical data model and the flexible view definition language.

Recent works [5, 6] study the XML update problem over relational databases using a *nested relational algebra*. They assume the view is always *well-nested*, that is, joins are through keys and foreign keys, and nesting is controlled to agree with the integrity constraints and to avoid duplication. The update over such a view is thus always translatable.

Our earlier work [26] studies the update translatability of XML views over the relational database in the "round-trip" case, which is characterized by a pair of reversible lossless mappings for (i) loading the XML documents into the relational database, and (ii) extracting an XML view identical to the original XML document back out of it. We prove that the view updates in this case are always translatable. In [27, 28] we extend [10] as a *clean-extended source theory* and provide a schema-based XML view update translatability checking algorithm. It particularly addresses new challenges related to the decision of translation existence when no restrictions have been placed on the defined view. That is, in general, conflicts are possible and a view cannot always be guaranteed to be well-nested (as assumed by [5, 6, 26]).

Our work in this paper further addresses the view update problem for XML views defined over an XML document. This in general is more complex due to dealing with a non-normalized XML data model in both the view and base. The proposed theory is comprehensive but practical with a large spectrum of XML views and XQuery updates being considered [2].

About Update Translation Approach. Works in this direction are different from above. They assume all the view updates considered are translatable and thus they are not concerned with verifying whether they indeed result in side effects in views. Their focus points are solving the ambiguity issue and finding the "best" translation.

[1, 13, 14] study the view update translation mechanism for SPJ queries on relations that are in BCNF. These works have been further extended for object-based views in [4]. [22] studies the performance of executing the translated updates over the relational database.

Commercial database systems, such as Oracle, DB2 and SQL-Server, also provide XML support. **Oracle** XML DB [3] provides SQL/XML as an extension to SQL, using functions and operators to query and access XML content as part of normal SQL operations, and also to provide methods for generating XML from the result of an SQL Select statement. The **IBM DB2** XML Extender [8] provides user-defined functions to store and retrieve XML documents in XML columns, as well as to extract XML elements or attribute values. However, neither IBM nor Oracle support update operations. [19] introduces XML view updates in **SQL-Server2000**, based on a specific *annotated schema* and update language called *updategrams*. Instead of using update statements, the user provides a before and after image of the view. The system computes the difference between the image and generates the corresponding SQL statements to reflect changes on the relational database.

7 Conclusion

In this paper, we tackle the problem of updating XML views published over XML data. We propose a *clean source theory* to determine whether a view update mapping is correct. We currently have a simple technique for checking update translatability that iterates over the set of sources, and tries to check if it forms a clean source. In the future, we would like to develop efficient algorithms to reduce this search space. Our clean source theory can then be used to establish the correctness of such algorithms.

[2] The XML views handled by our clean source theory include general FLWR expressions [25], with the exception of if/then/else expressions; user-defined functions and aggregate functions, such as max(), count(), etc.

References

1. A. M. Keller. The Role of Semantics in Translating View Updates. *IEEE Transactions on Computers*, 19(1):63–73, 1986.
2. F. Bancilhon and N. Spyratos. Update Semantics of Relational Views. In *ACM Transactions on Database Systems*, pages 557–575, Dec 1981.
3. S. Banerjee, V. Krishnamurthy, M. Krishnaprasad, and R. Murthy. Oracle8i - The XML Enabled Data Management System. In *ICDE*, pages 561–568, 2000.
4. T. Barsalou, N. Siambela, A. M. Keller, and G. Wiederhold. Updating Relational Databases through Object-Based Views. In *SIGMOD*, pages 248–257, 1991.
5. V. P. Braganholo, S. B. Davidson, and C. A. Heuser. On the Updatability of XML Views over Relational Databases. In *WEBDB*, pages 31–36, 2003.
6. V. P. Braganholo, S. B. Davidson, and C. A. Heuser. From XML view updates to relational view updates: old solutions to a new problem. In *VLDB*, pages 276–287, 2004.
7. M. J. Carey, J. Kiernan, J.Shanmugasundaram, E. J. Shekita, and S. N. Subramanian. XPERANTO: Middleware for Publishing Object-Relational Data as XML Documents. In *The VLDB Journal*, pages 646–648, 2000.
8. J. M. Cheng and J. Xu. XML and DB2. In *ICDE*, pages 569–573, 2000.
9. S. S. Cosmadakis and C. H. Papadimitriou. Updates of Relational Views. *Journal of the Association for Computing Machinery*, pages 742–760, Oct 1984.
10. U. Dayal and P. A. Bernstein. On the Correct Translation of Update Operations on Relational Views. In *ACM Transactions on Database Systems*, volume 7(3), pages 381–416, Sept 1982.
11. M. F. Fernandez, A. Morishima, D. Suciu, and W. C. Tan. Publishing Relational Data in XML: the SilkRoute Approach. *IEEE Data Engineering Bulletin*, 24(2):12–19, 2001.
12. H. Jagadish, S. Al-Khalifa, L. Lakshmanan, A. Nierman, S. Paparizos, J. Patel, D. Srivastava, and Y. Wu. Timber: A native xml database. In *VLDB*, 2002.
13. A. M. Keller. Algorithms for Translating View Updates to Database Updates for View Involving Selections, Projections and Joins. In *Fourth ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, pages 154–163, 1985.
14. A. M. Keller. Choosing a View Update Translator by Dialog at View Definition Time. In *VLDB*, pages 467–474, 1986.
15. M. Fernandez et al. SilkRoute: A Framework for Publishing Relational Data in XML. *ACM Transactions on Database Systems*, 27(4):438–493, 2002.
16. N. May, S. Helmer, and G. Moerkotte. Nested queries and quantifiers in an ordered context. In *Proc. of the Int. Conf. on Data Engineering (ICDE)*, pages 239–250, 2004.
17. S. Rielau. INSTEAD OF Triggers - All Views are Updatable! http://www-106.ibm.com/developerworks/db2/library/techarticle/0210rielau/0210rielau.html.
18. L. A. Rowe and K. A. Shoens. Data abstraction, views and updates in rigel. In *ACM SIGMOD*, 1979.
19. M. Rys. Bringing the Internet to Your Database: Using SQL Server 2000 and XML to Build Loosely-Coupled Systems. In *VLDB*, pages 465–472, 2001.
20. K. C. Sevcik and A. L. Furtado. Complete and compatible sets of update operations. In *ICMOD*, 1978.
21. H. Shu. Using Constraint Satisfaction for View Update Translation. In *European Conference on Artificial Intelligence*, 1998.
22. I. Tatarinov, Z. G. Ives, A. Y. Halevy, and D. S. Weld. Updating XML. In *SIGMOD*, pages 413–424, May 2001.
23. W3C. XML Schema. http://www.w3.org/XML/Schema.
24. W3C. XQuery: A Query Language for XML. http://www.w3.org/TR/xquery/, February 2001.
25. W3C. XQuery 1.0 Formal Semantics. http://www.w3.org/TR/query-semantics/, June 2003.
26. L. Wang, M. Mulchandani, and E. A. Rundensteiner. Updating XQuery Views Published over Relational Data: A Round-trip Case Study. In *XML Database Symposium*, pages 223–237, 2003.

27. L. Wang and E. A. Rundensteiner. On the Updatability of XQuery Views Published over Relational Data. In *ER*, 2004.
28. L. Wang and E. A. Rundensteiner. Updating XML Views Published Over Relational Databases: Towards the Existence of a Correct Update Mapping. Technical Report WPI-CS-TR-04-19, Computer Science Department, WPI, 2004.
29. X. Zhang, K. Dimitrova, L. Wang, M. EL-Sayed, B. Murphy, L. Ding, and E. A. Rundensteiner. RainbowII: Multi-XQuery Optimization Using Materialized XML Views. In *Demo Session Proceedings of SIGMOD*, page 671, 2003.