HUX: Handling Updates in XML

Ling Wang, Elke A. Rundensteiner, Murali Mani and Ming Jiang
Worcester Polytechnic Institute, Worcester, MA 01609, USA
{lingw|rundenst|mmani|jiangm}@cs.wpi.edu

1. MOTIVATION

With the growing popularity of XML, it has become the primary data model for creating XML wrapper views and querying the database through them [5, 12, 9]. For example, biologists use XML views to provide the online protein information resource; authorized geography institutes as well as normal GPS users use XML views to represent the world geography information. XML views are also widely used in education systems. Let us consider Assistment system (http://www.assistment.org/) which is developed jointly by Worcester Polytechnic Institute and Carnegie Mellon University and funded by NSF and United States Department of Education. It is a web-based system that offers instruction to students while providing a more detailed evaluation of their abilities to the teacher than is possible under current approaches. This greatly helps teachers assisting students' development and assessing their abilities. This system is currently used in some middle schools in Massachusetts. Fig. 1 shows a part of relational schema of the Assistment system. Using this system, a teacher may be interested in the problem information as shown by the XML wrapper view in Fig. 2 [1].

Although not addressed yet, support of update operations against such wrapper views would be useful. For instance, when the teacher looks at the view, he may want to insert students' new scores for some existing problems; delete some inappropriate or old problems or update a student's score after regrading. However, allowing the teacher to directly update the base tables might not be practical, as he need not be aware of underlying data models, such as the update language or access modes of the underlying storage system. The ideal solution is that the teacher simply describes what the updated view should look like and this update requirement is automatically translated to the underlying database updates. This is also the goal we want to achieve.

9 [1]Like other recent XML systems [5, 8], we use a basic XML view, called *default XML view*, to define one-to-one relational-to-XML mappings. User specific views are defined on top of the default XML view

VLDB '06, September 12-15, 2006, Seoul, Korea.

Update requests through views are difficult in the sense that they have to be translated into "appropriate" updates on the underlying databases. This translation should be handled transparently by the database system, while the effects of translated updates should at the same time meet the user expectations.

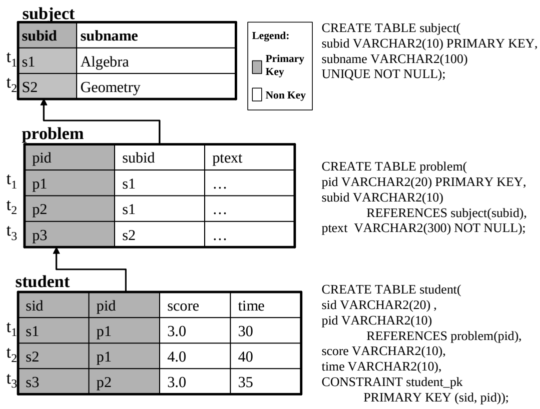


Figure 1: **Relational Database of Running Example**

In general, two problems concerning updating views need to be tackled. First, *update translatability* concerns whether some updates on the base data storage, which may for example be a relational database or a native XML document, can be made to have the same effect as the given update to the view directly without causing any view-side-effect. Second, we need to devise an appropriate *translation strategy*. That is, assuming the view update is indeed translatable, how to map the updates on the XML view into the equivalent tuple-based SQL updates or XML document updates on the base data.

The XML view update problem is more complex than the traditional relational view update problem [2, 6, 7]. Not only do all the problems in the relational context still exist in XML semantics, but we also have to address the new update issues introduced by the XML hierarchical data model and its flexible update language.

We demonstrate our XML view updating system named **HUX** (Handling Updates in XML). It provides a reliable and efficient solution for both aspects of the XML view update problem. HUX is complementary and also compatible with much of recent effort in XML query support by both academic prototype systems and commercial DBMS vendors, and thus we expect that HUX can be used commonly as an advanced feature for various view based applications.

```
<ProblemView>
FOR $subject IN document("subject.xml")/subject/row,
    $problem IN document("problem.xml")/problem/row
WHERE $subject/subid = $problem/subid
RETURN
   <ProblemInfo>
         $problem/ptext,
         <subject>$subject/subname</subject>
         FOR $student IN document("student.xml")/student/row
         WHERE $student/pid = $problem/pid
         RETURN
             <StudentInfo>
                   $student/sid,
                   $student/score,
                   $student/time
             </StudentInfo>
   </ProblemInfo>
</ProblemView>
```

(a)

```
</ProblemView>
  <ProblemInfo>
     <ptext>  x + 4 = 10. x = ?  </ptext>
     <subject>  Algebra </subject>
      <StudentInfo>
            <sid> s1  </s1>
            <score> 3.0 </score>
            <time> 30</time>
      </StudentInfo>
      <StudentInfo>
            <sid> s2  </s1>
            <score> 4.0 </score>
            <time> 40 </time>
     </StudentInfo>
  </ProblemInfo>

  <ProblemInfo>
     <ptext>  2 * x = 7. x = ?  </ptext>
     <subject>  Algebra </subject>
      <StudentInfo>
            <sid> s3  </s1>
            <score> 3.0 </score>
            <time> 35</time>
      </StudentInfo>
  </ProblemInfo>

  <ProblemInfo>
        <ptext>  For the three angles in a
triangle, one is 50°, one is 60°, what is
the rest?
        </ptext>
        <subject>  Geometry </subject>
  </ProblemInfo>
</ProblemView>
```

(b)

Figure 2: XQuery Views over Relational Database in Fig. 1

2. HUX: THE FULL-FLEDGED XML VIEW UPDATING FRAMEWORK

The framework of HUX system is shown in Fig. 3. The *U-Filter* subsystem first addresses update translatability issue by performing a three-step check to detect all the potential conflicts in both schema [14, 15] and data [15], which can cause an update to be untranslatable. Second, *U-Translator* is used to search correct update translations over the base data storage when it is mappable [16]. In particular, we focus on the following:

- Resolve the mismatch between the XML hierarchical view model and the base data model.
- Handle flexible XML view updates, namely updates can happen on any XML view element instead of just root element.
- Support efficient order-sensitive update translation.

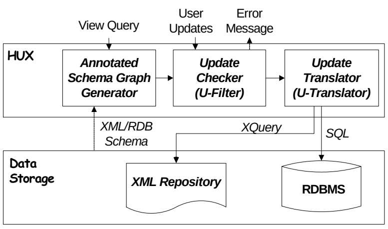


Figure 3: The Framework of XML View Updating System

To achieve these, several challenges have to be addressed as briefly illustrated by the following examples.

First, updates over a given view can be problematic for several different reasons, such as violating constraints, causing view side effects or conflicting with the underlying data. To avoid translation costs caused by forced rollbacks to undo faulty update translations, or even worse, a faulty translation, an update needs to be carefully checked before the translation starts.

EXAMPLE 1. *To delete only the ptext of a subject is not translatable since the ptext of the subject relation is NOT NULL.*

EXAMPLE 2. *To delete the subject of the first ProblemInfo element from the ProblemView is not translatable. The reason is there is a foreign key from problem relation to the subject relation in the underlying relational database. When the subject is deleted, the corresponding problem tuple has to be either also deleted, or the pid of the problem is replaced with NULL, depending on the deletion policy defined by the foreign key constraints. However, neither of these two are correct because they both would cause the corresponding problem elements to no longer appear in the view. We thus say that this update is not translatable since it causes a view side effect in the form of an unintended view deletion. We can draw the conclusion for each subject element; therefore schema knowledge decides the un-translatability of the updates.*

EXAMPLE 3. *To delete a StudentInfo of a certain ProblemInfo from the ProblemView is always translatable. The reason is each student tuple contributes to one StudentInfo only and there is no other view element that refers to it. We can draw the same conclusion for each StudentInfo element; thus schema knowledge guarantees the update always translatable.*

EXAMPLE 4. *To delete a certain ProblemInfo is always translatable by deleting the corresponding tuple in problem relation. This is guaranteed because every problem tuple contributes to only one ProblemInfo. However, there are also other possible ways to achieve the updates. For example, for the third ProblemInfo, we can delete the the second tuple in subject to delete it, because this subject tuple only contributes to this view element and won't cause any side effects. On the contrary, the other two ProblemInfo can not be deleted by deleting the first tuple in subject, because it contributes to two view elements.*

From the last example, we observe the fact even though we can use schema knowledge to decide whether an update is translatable or not, we sometimes need concrete data to find all the correct translations.

3. U-FILTER: A LIGHT-WEIGHT XML VIEW UPDATE CHECKER

U-Filter is a component of our system and used to check whether a view update is translatable [14, 15]. As shown by Fig. 4, U-Filter first performs two steps of schema-level (and thus very inexpensive) checking. Only when necessary, more expensive checking requiring the base data to be accessed is employed.

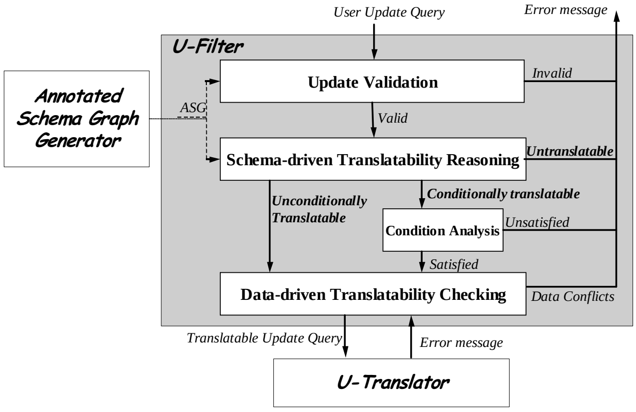


Figure 4: The U-Filter sub-system

The first step, called *update validation*, identifies whether the given view update is valid according to the *view schema*. The problem in Example 1 is identified by this step. This view schema can either be pre-defined or be inferred from the view definition query and the base relational schema knowledge. It is modeled using a graph named *Annotated Schema Graph*, and generated similar to the *view forest* in SilkRoute [12]. Given that a lot of work has already been done in the literature on schema validation [4, 13], here we focus on two questions closely related with the view update issue: (i) How to extract the view schema from the view query and the relational schema? (ii) Which of the extracted constraints in the view schema should the validation procedure consider?

In the second step, called *schema-driven translatability reasoning*(STAR), updates determined to be valid by Step 1 are further examined. The potential view side effects are checked, which can be caused by different reasons, including foreign key constraints conflicting with the view structure, or, base data duplication in the view. This compile-time check only utilizes the view query and the relational schema. Example 3 and 2 are identified here. Techniques in this step is included in our earlier work [14, 15]. [14] extends [7] into a *clean-extended source theory* for XML views; this serves as the criteria for determining whether a given translation is correct. [15] focuses on identifying the factors deciding the translatability of deletions over XML views.

There are some updates that may have potential side effects need to further checked. For instance, in Example 4, in order to make sure whether deleting from *subject* is a correct translation or not, we need to check how many view elements a certain *subject* contributes to. In our third step, the run-time *data-driven translatability checking*, such facts will be identified by issuing *probe queries*. As an example, the probe query for Example 4 is: { *SELECT COUNT(*) FROM subject WHERE subject.sid = problem.sid AND problem.pid = "p3"*}. As the result is 1, the second *subject* tuple contributes to only one *ProblemInfo* element. This check can only be resolved by examining actual base data. This is typically rather expensive. Hence it is practical to employ it only at last, when the prior steps have been considered.

Moreover, for an order-sensitive XML view using a FLWOR expression, an order-sensitive update can delete an existing view element in a certain position or insert a new view element into a certain position of the view. This update can cause new data-related update translatability issues in terms of the update position itself being invalid. Special order-specific probe queries are utilized to verify the legality of the deleting or inserting position.

4. U-TRANSLATOR: AN EFFICIENT UPDATE TRANSLATION MECHANISM

Now assume a view update is not filtered out by U-Filter and thus is found to be translatable. Then we tackle the question of how best to translate updates on the XML view into equivalent tuple-based SQL updates or XML document updates on the base data. This requires understanding the ways in which individual view update requests may be satisfied by updates over the underlying data storage. In some cases, there will be precisely one way to perform the database update that results in the desired view update. However, in most cases, the new view state may correspond to several different database states. Consequently, the question of choosing a particular database state arises. Of the multiple database states, we would like to choose one that is "as close as possible" under some measure to the original database state, namely, to minimize the effect of the view update on the database [3, 10].

We design a set of criteria to distinguish between a "good" and a "bad" translation that extends the validity criteria established for relational view updates [10, 11, 1, 3]. Conceptually an enumeration of all possible valid translations of each view update on the view should be examined. For practical reasons, we do not instantiate this enumeration in our system. We merely use it to define the space of alternatives for correct update translations.

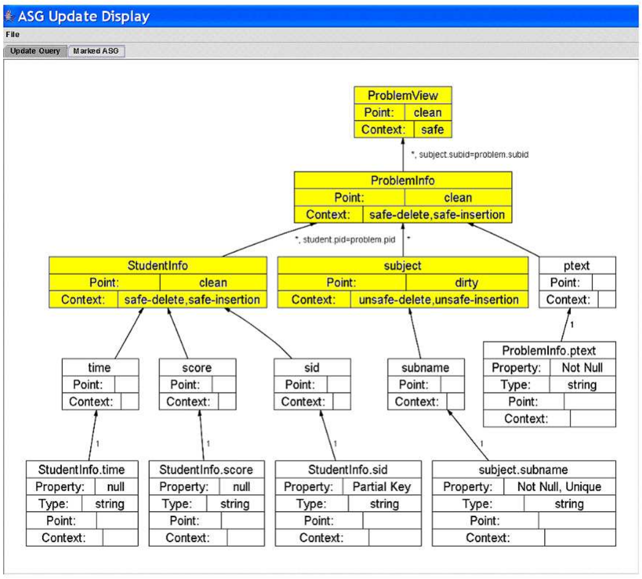
When the underlying data storage is a relational database, the mismatch between the two update languages (XQuery FLWU updates on the view versus SQL queries on the base) is carefully dealt with to pick efficient update translation. New optimization techniques on generating efficient base updates, which consider the performance impacts imposed by for example using some partially materialized index, have also been incorporated into our system. The implementation is based on the XML algebra tree (XAT) of Rainbow XQuery engine.

We have developed an order sensitive query optimization technique when an XML view is defined over relational databases [16]. U-Translator utilizes this general approach for supporting order-sensitive XQuery-to-SQL translation that works irrespective of the chosen XML-to-relational data mapping and the selected order-encoding method. Special probe queries over the XML base are employed, which extract information to generate a proper order code in the relational database or position in the XML document for the newly inserted view element.

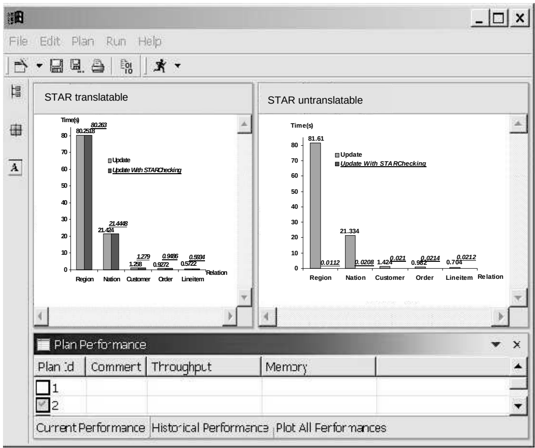
5. DEMONSTRATION

We demonstrate HUX in two different scenarios, namely when the XML view is defined over (i) a relational database and (ii) an XML document. We provide various XML views for teachers and students. For example, a teacher can manage current problems for the subject he is teaching by maintaining (updating) the problems listed under that subject through *SubjectViews*. He can also insert or delete students' scores of a certain problem in *ProblemViews*. The student can thus get refreshed information including problems he has finished, time he used and scores he got through *StudentViews*. All of these services rely on the capability of updating through views. HUX performs careful update translatability checking by *U-Filter* to assure that the database is properly maintained. HUX also provides efficient service through *U-Translator* to translate the user update into correct database updates, even for heavy update workloads.

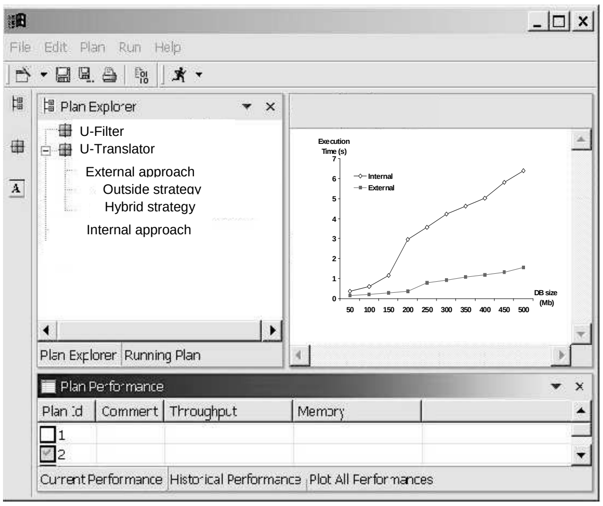
U-Filter. We demonstrate our three-step update translatability checking of U-Filter, which make extensive use of the *Annotated Schema Graph* (below). As an example, to check the translatability of update in Example 2, the ASG is pre-marked as shown in screen snapshot below. Since the *subject* node is marked as *(dirty|unsafe-delete)*, the update on deleting *subject* only is indicated to be not translatable [14, 15].



We monitor the performance of different steps over different update cases. The schema-based checking time is almost negligible in both the successfully translatable case and the failed untranslatable case.



U-Translator. We show how a given view update is translated into a sequence of SQL update statements. We also illustrate the performance of the generated update statements when different update translation policies are used. Especially, when order is considered, we will show strategies for optimization of order-sensitive database updates. We monitor the execution performance of different update translation "plan" (below), which provides valuable information about which translation strategy should be selected.



Acknowledgment. We would like to thank Song Wang, Maged ElSayed and Xin Zhang for their contributions to this project in terms of ideas and systems. We would also like to thank David Krolick, Alex Perry for their contributions for implementation help.

6. 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] T. Barsalou, N. Siambela, A. M. Keller, and G. Wiederhold. Updating Relational Databases through Object-Based Views. In *SIGMOD*, pages 248–257, 1991.

[4] M. Benedikt, C. Y. Chan, W. Fan, and R. Rastogi. DTD-Directed Publishing with Attribute Translation Grammars. In *VLDB*, pages 838–849, 2002.

[5] 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.

[6] S. S. Cosmadakis and C. H. Papadimitriou. Updates of Relational Views. *Journal of the Association for Computing Machinery*, pages 742–760, Oct 1984.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] A. M. Keller. Choosing a View Update Translator by Dialog at View Definition Time. In *VLDB*, pages 467–474, 1986.

[12] M. Fernandez et al. SilkRoute: A Framework for Publishing Relational Data in XML. *ACM Transactions on Database Systems*, 27(4):438–493, 2002.

[13] M. Murata, D. Lee, M. Mani, and K. Kawaguchi. Taxonomy of xml schema languages using formal language theory. In *ACM TOIT*, 2005.

[14] L. Wang and E. A. Rundensteiner. On the Updatability of XQuery Views Publised over Relational Data. In *ER*, pages 795–809, 2004.

[15] L. Wang, E. A. Rundensteiner, and M. Mani. Updating XML Views Published Over Relational Databases: Towards the Existence of a Correct Update Mapping. In *DKE Journal*, 2005 to appear.

[16] L. Wang, S. Wang, B. Murphy, and E. A. Rundensteiner. Order Sensitive XQuery Processing over Relational Sources: An Algebraic Approach. In *IDEAS*, 2005.