Preserving Referential Constraints in XML Document Association Relationship Update

Eric Pardede[1], J. Wenny Rahayu[1], David Taniar[2]

[1] Department of Computer Science and Computer Engineering,
La Trobe University, Bundoora VIC 3083, Australia
{ekpardede, wenny}@cs.latrobe.edu.au

[2] School of Business System,
Monash University, Clayton VIC 3800, Australia
David.Taniar@infotech.monash.edu.au

Abstract. In this paper we propose the usage of W3C-standardized query language, XQuery, to accommodate XML Update. Our main aim is to enable the update without violating the semantic constraint of the documents. The focus of the update is on the association relationship. Therefore, we distinguish the relationship type based on the number of participants, cardinality, and adhesion. We also propose the mapping of the association relationship in the conceptual level to the XML Schema. We use XML Schema for structure validation, even though the algorithm can be used by any schema languages.

1 Introduction

In the last few years the interest in storing XML Documents in the native XML Databases (NXD) has emerged rapidly. The main idea is to store the documents in their natural tree form. However, it is well-known that many users still prefer to use DBMS that are based on established data models such as Relational Model for their document storage. One reason is the incompleteness of NXD query language. Many proprietary XML query languages and even W3C-standardized languages still have limitations compared to the Relational Model SQL. One of the most important limitations is the lack of the update operations support [8].

Different NXD applies different strategies for XML updates. Very frequently after update operations, the XML document contains many dangling references, loses the key attribute, has unnecessary duplications, and many other problems that indicate very low database integrity. In our best knowledge, there is no XML query language that has considered the integrity issues that emerge from the update operations. We suggest this as an important issue to raise and to investigate further.

This paper proposes the XQuery [11] for update operations. It is based on the basic operations firstly mentioned in [9]. Our main contribution is that we have included semantic constraints in the operations. It focuses on the update for association relationship, which can be defined as a "reference" relationship between one to another node/document in an XML tree. We will distinguish different association type based on the number of participants and on the cardinality. The XQuery update considers different target contents whether it is a key node or a key reference node. By doing this, we remove the possibilities of low integrity data after each update operation.

For XML update we need structure validation and we propose the use of XML Schema [10]. Thus, a part of the paper is allocated to developing a transformation on the association relationship into XML Schema, in a way that suits our proposed XQuery.

2 XML Document Update: An Overview

There are several strategies for updating XML documents in NXDs [1][8]. At the time of writing, there are three main strategies: (i) use **proprietary update language** that will allow updating within the server [3][7], (ii) use **XUpdate**, the standard proposed by XML DB initiative for updating a distinct part of a document [2][5], and (iii) use **XML API** after retrieve the document out of the database [4]. It is important to mention that none of these are concerned with the semantic constraint of the updated XML document.

Different strategies have limited the database interchangeability. To unite these different strategies, [9] has tried to propose the update processes for XML Documents into an XML language. These processes are embedded into XQuery and thus, can be used for any NXD that has supported this language.

In [9] the update is embedded in the XQuery FLOWR expression. Note the difference between the original XQuery (Fig.1) and its extension [(Fig. 2). The UPDATE clause specifies the node targets that will be updated. Inside the UPDATE clause, we determine the operation types.

```
FOR $binding1 IN path
    expression…
LET $binding := path
    expression…
    WHERE predicate, …
    ORDER BY predicate, …
        RETURN results
```

Fig. 1. XQuery FLOWR Expressions

```
FOR. . .LET. . .WHERE. . .
UPDATE $target{
       DELETE $child|
       INSERT content
           [BEFORE|AFTER $child]|
       REPLACE $child WITH $content|
       {,subOp}*
       }
```

Fig. 2. XQuery UPDATE Extensions

Nonetheless, even with this proposal there is a basic question to answer. We do not know how the update operations can affect the semantic correctness of the updated XML Documents. Specifically for the association relationship, this proposal does not maintain the referential constraint between the associated nodes/documents. This fact has motivated us to take the topic a step further.

3 Association Relationships in XML Document

Association relationship is a “reference” relationship between one object with another object in a system. For an XML document, the object is a node or a document. Semantically, this relationship type can be distinguished by some constraints such as *number of participant type*, *cardinality*, and *adhesion*. These constraints can be easily identified in XML Data Model such as in Semantic Network Diagram [2]. We will show a running example describing different association relationship constraints (see Fig. 3).

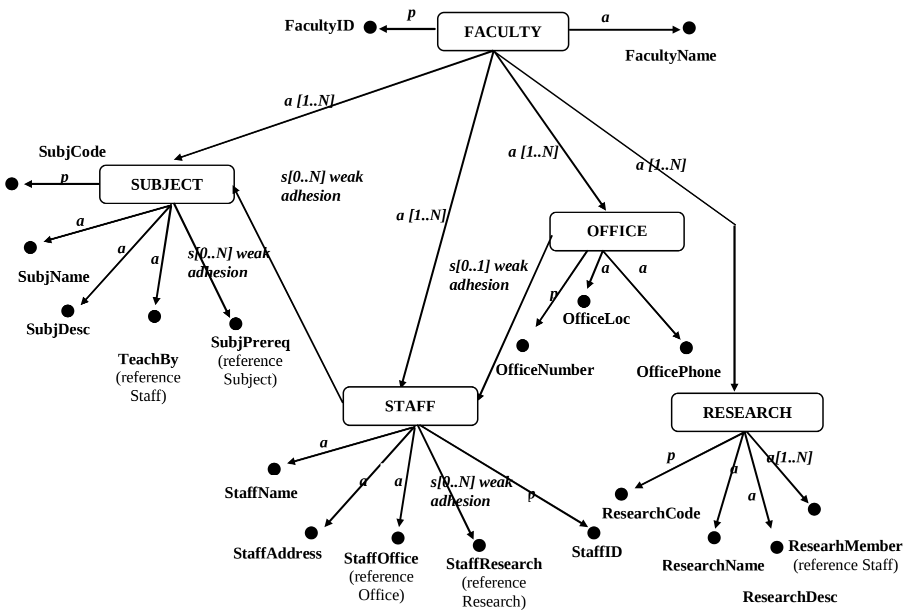

Fig. 3. Association Types in XML Document Tree

The number of participants can be distinguished into unary and binary. In the former, an object refers to another object of the same type. For example, there is one unary relationship, where node *SubjPrereq* in a Subject document refers to another subject. In the binary relationship, the participants come from different types. In our example above, there are: Subject:Staff, Staff:Office, and Staff:Research.

Association cardinality identifies the number of objects of the same type to which a single object can relate. For example, staff can teach no subject at all to many subjects [0..N]. In the diagram the default cardinality is [1..1].

Finally, association adhesion identifies whether the associated objects must or must not coexist and adhere to each other. In the diagram the default adhesion is a strong adhesion. For example, Staff has weak adhesion association to Subject, which means that the former does not require the latter to exist.

4 Mapping Association Relationship to XML Schema

XQuery update will require a schema to validate the document structure. We select XML Schema because as well as its precision in defining the data type, XML Schema has an effective ability to define the semantic constraints of a document. It includes the cardinality, adhesion, and referential integrity constraint depicted in Fig. 3.

First, the cardinality constraint is very straightforward and well-known. We can determine the “minoccurs” and “maxoccurs” after an element declaration. It is a general practice that these constraints are not used for attribute. A particular attribute will appear, at most, once in an object/instance.

Second, the adhesion constraint can easily be indicated by the “use” constraint. A strong adhesion has use “required” and a weak adhesion has use “optional”. However, in practice the “use” constraint can only be attached in attribute. We will find a problem if, for example, the adhesion constraint is applied to an element. Fortunately, we can utilize the cardinality constraint to define the adhesion constraint as well. Cardinality [0..1] or [0..N] indicates a weak adhesion, while [1..1] or [1..N] indicates a strong adhesion.

For a consensus we provide a table combining the two constraints that can be very useful for the next section.

Table 1. Adhesion and Cardinality Constraints in XML Schema

Adhesion	Cardinality	What to Use
Strong	1:1	Attribute (use = “required”)
	1:N	Element (minoccurs=“1” maxoccurs=“unbounded”)
	N:N	Element (minoccurs=“<2.. >” maxoccurs=“unbounded”)
Weak	0:1	Attribute (use = “optional”)
	0:N	Element (minoccurs=“0” maxoccurs=“unbounded”)

Thirdly, for the referential integrity constraint, XML Schema provides two options ID/IDREF and key/keyref. We have chosen to use the latter for several reasons: (i) key has no restriction on the lexical space [10], (ii) key cannot be duplicated and thus facilitate the unique semantic of a key node, and (iii) key/keyref can be defined in a specific element and, thereby, helping the implementer to maintain the reference.

For our design, we propose to create a specific element under the root element called “GroupKey” to store all keys and keyrefs. It is required to track the path of the associated elements once an update is performed. We cannot just declare the key/keyref inside the element where they are appeared. We will experience problems in tracking an associated element since keyref can only trace the key declared in the same element or in its direct predecessor. For example, if we declare the keyref *TeachBy* inside subject element and key *StaffID* inside staff element, the keyref cannot trace the key and thus, if we update a staff, there is no checking done to ensure the subject refers to the particular staff.

In a unary relationship, the key and keyref attribute/element is located under one element. In a binary relationship we have to determine the location of the keyref attribute/element. The adhesion constraint is used for this purpose. Locate the keyref attribute/element under the associated element that has strong adhesion towards the other. In the case where both associated elements are in strong adhesion, the keyref attribute/element will be added in both elements. For example (see Fig.3), in the relationship between staff and research, we add reference *ResearchMember* inside research element that refers to *StaffID*, and reference *StaffResearch* inside research element that refers to *ResearchCode*.

5 Proposed XQuery Update

Now we have shown how to map the conceptual model into logical XML Schema, we can begin to propose the XQuery for Update processes. The update can be differentiated into three main parts: deletion, insertion, and replacement.

5.1 XQuery for Delete Operation

Deletion is the simplest update operation, especially if we want to delete an attribute/element that contains simple data. In this case we only require the binding to the parent and the child. We do not need to perform predicate checking on instances. However, if the attribute and element also have roles as key/keyref the checking on instances is required to maintain the referential integrity constraint.

For our proposed XQuery, we differentiate the operation for key and keyref since we put focus on the association relationship. For deletion however, there is no constraint checking required for keyref node since it is treated like simple data content.

The following XQuery shows functions used for predicates in key deletion. Function *checkKeyRefNode* checks whether the target key node has keyref node referred to it. This function highlights one reason why we select key/keyref instead of ID/IDREF. Now the implementer can check only the path where the keyref is defined. Since we have grouped all key and keyref declarations in the “GroupKey” element, we only need to check the “GroupKey” element.

Under a condition (see the function), the *checkKeyRefNode* function proceeds to function *checkKeyRefInstance*. This function checks that there is no instance that has keyref attribute/element referring to the target key content. This is the reason we also need to pass the content of the target key node. If the function returns false, we cannot perform the deletion.

```
FUNCTION checkKeyRefNode($gKBindingPath, $keyName, $keyContent) RETURN
BOOLEAN {
    LET $gkRef := $gKBindingPath/keyref/[@refer]
    RETURN
        IF $gkRef = $keyName
           THEN getKeyRefInstance($gKBindingPath, $keyName, $keyContent)
           ELSE TRUE}
```

```
FUNCTION   checkKeyRefInstance($gKBindingPath,   $keyName,   $keyContent)
RETURN BOOLEAN
{
FOR $keyRefBinding IN $gKBindingPath/keyref[@refer=$keyname]
LET $keyRefName:=$gKBindingPath/keyref/@name
    $keyRefPath:=$keyRefBinding/selector/xpath
    $keyRefInstance:=$keyRefPath[@$keyRefName, "$keyContent"]
RETURN
    IF exists ($keyRefInstance)
       THEN FALSE
       ELSE TRUE}

Example:
FOR $g IN document("Faculty.xml")/Faculty/GroupKey
    $p IN document("Faculty.xml")/Faculty/Subject(@SubjCode = "ADB41")
    $c IN $p/SubjCode
LET $cContent := “ADB21”
UPDATE $p{
WHERE checkKeyRef($g, "SubjCode", $cContent)
       UPDATE $p{
              DELETE $p (:delete the key and the siblings:)
                     }}}
```

The example following the functions shows the key deletion *Subjcode* in document “Faculty.xml” of instance with *SubjCode* equals to “ADB41”. We know that there is a node *SubjPrereq* that refers to *SubjCode*. Thus, if there is any instance where *SubjPrereq* value refers to *SubjCode* “ADB41”, the deletion should be restricted.

Note that if the *checkKeyRef* function returns TRUE, we will delete whole elements and not only the key *SubjCode*. This is because we do not have trigger-based delete, which will delete the whole element if the key is deleted.

5.2 XQuery for Insert Operation

Unlike deletion, insertion requires the constructor of a new attribute or new element. Beside the XQuery for key insertion, we have to propose insertion for keyref as well, to make sure that the new keyref actually refers to an existent key.

Function *checkKeyDuplicate* returns TRUE if the target is not duplicating an existing instance. The example following the function shows the checking if we want to insert a *SubjCode* with content “ADB41”.

```
FUNCTION checkKeyDuplicate($bindingPath) RETURN BOOLEAN
{
RETURN
    IF EXISTS($bindingPath)
       THEN FALSE
       ELSE TRUE}

Example:
FOR $g IN document("Faculty.xml")/Faculty(@FacID = "FSTE")
    $p IN $g/Subject(@SubjCode="ADB41")
LET $c:= SubjCode
    $cContent:=”ADB41”
UPDATE $p{
WHERE checkKeyDuplicate($p)
       UPDATE $p{
              INSERT new_att($c, $cContent)
```

```
                        }}
```

For keyref insertion, we propose three additional functions. The first is for the referential integrity constraint and the following two are for the cardinality constraint. The cardinality is required since a keyref can actually refer to more than one key instance.

Function *checkKeyInstance* passes the keyref name and content, and then checks the key being referred. Function *getMaxOccurs* and *checkMaxOccurs* are used to calculate the maxoccurs constraint of a keyref node. If the particular instance has the maximum number of keyref, we will disable the insertion of another keyref content.

```
FUNCTION  checkKeyInstance($gKBindingPath,  $keyRefName,  $keyRefContent)
RETURN BOOLEAN
{
FOR $keyRefBinding IN $gKBindingPath/keyref[@name = $keyRefName],
LET $keyName:=$gkeyRefBinding/@refer,
    $keyPath:=$gKeyRefBinding/selector/xpath,
    $keyInstance:=$keyPath[@$keyName, "$keyRefContent")
RETURN
   IF EXISTS($KeyInstance)
      THEN TRUE}

FUNCTION getMaxOccurs($xsName, $parentName, $childName) RETURN INTEGER
{
FOR $pDef IN document($xsName)//xs:element[@name=$parentName],
    $cDef IN $pDef/xs:element[@name=$childName],
LET $cMaxOccurs:=$cDef/maxoccurs,
RETURN $cMaxOccurs}

FUNCTION checkMaxOccurs($bindingPath, $cMaxOccurs) RETURN BOOLEAN
{
LET $instanceOccurs:=count($bindingPath)
RETURN
    IF $cMaxOccurs >= $instanceOccurs + 1
       THEN FALSE
       ELSE TRUE}
```

5.3 XQuery for Replace Operation

Since replacement can be seen as a combination of deletion and insertion, we can reuse the functions we have already described in the last two sub-sections. In fact, we do not need the functions for cardinality constraints during keyref replacement as well.

For replacement of a key, we have to check whether the new key content does not duplicate any existing instance. For replacement of a keyref, we just need to check whether the new keyref content refers to a valid instance. No cardinality checking is required since to accommodate a new keyref, we have deleted another keyref. XQuery below shows the example of replacement for keyref *TeachBy* element.

```
FOR $g IN document("Faculty.xml")/Faculty/GroupKey
    $p IN document("Faculty.xml")/Faculty/Subject(@SubjCode = "ADB41")
    $c IN $p/TeachBy
LET $cName := TeachBy
    $cContent := “WR01”
UPDATE $p{
WHERE checkKeyInstance($g, $cName, $cContent)
       UPDATE $p{
```

```
                REPLACE $c WITH <TeachBy>WR01</TeachBy>
                       }}
```

6 Conclusion

In this paper, we propose extending XQuery to preserve semantic constraints during an XML update. The update operations are divided into deletion, insertion, and replacement. The focus in the paper is on the association relationship type, thus each operation considers the key and key reference target node. By doing this, we can avoid database anomalies that might occur using conventional XQuery.

Since Update requires a document structure validation, we also propose the transformation of the relationship into XML Schema. The constraints captured are cardinality, adhesion, and referential integrity constraint.

With this extension, XML query languages (in the form of XQuery) are becoming more powerful. Concurrently, it can increase the potential of using tree-form XML repository such as Native XML Database.

References

1. Bourett, R.: XML and Databases. http://www.rpbourret.com/xml/XMLAndDatabases.htm, (2003)
2. Feng, L., Chang, E., Dillon, T.S.: A Semantic Network-Based Design Methodology for XML Documents. ACM Trans. Information System, Vol. 20, No. 4. (2002) 390-421
3. Ipedo.: Ipedo XML Database, http://www.ipedo.com/html/products.html, (2004)
4. Jagadish, H. V., Al-Khalifa, S., Chapman, A., Lakhsmanan, L. V. S., Nierman, A., Paprizos, S., Patel, J. M., Srivastava, D., Wiwattana, N., Wu, Y., Yu, C.: TIMBER: A native XML database. VLDB Journal, Vol. 11, No. 4. (2002) 279-291
5. Meier, W.M.: eXist Native XML Database. In Chauduri, A.B., Rawais, A., Zicari, R. (eds): XML Data Management: Native XML and XML-Enabled Database System. Addison Wesley (2003) 43-68
6. Robie, J.: XQuery: A Guided Tour. In Kattz, H. (ed.): XQuery from the Experts. Addison Wesley (2004) 3-78
7. SODA Technology.: SODA. http://www.sodatech.com/products.html, (2004)
8. Staken, K.: Introduction to Native XML Databases. http://www.xml.com/pub/a/2001/10/31/nativexmldb.html, (2001)
9. Tatarinov, I., Ives, Z.G., Halevy, A. Y., Weld, D. S.: Updating XML. ACM SIGMOD (2001) 413-424
10. Vlist, E. V-D.: XML Schema, O'Reilly, Sebastopol (2002)
11. W3C: XQuery 1.0: An XML Query Language. http://www.w3.org/TR/xquery, (2001)
12. XML DB: XUpdate – XML Update Language, http://www.xmldb.org/xupdate/, (2004)