XML-RL Update Language

Li Lu, Mengchi Liu, and Guoren Wang

School of Computer Science, School of Mathematics and Statistics,
Carleton University, Ottawa, Ontario, Canada, K1S 5B6
llu@math.carleton.ca,{mengchi, wanggrg}@scs.carleton.ca

Abstract. Supporting for updating XML documents has recently attracted interest. This paper presents a novel declarative XML update language, which is an extension of the XML-RL query language. We define XML-RL update syntax and a set of primitive update operations to fully evolve XML into universal data representation and sharing format.

1 Introduction

Over the past several years, a few languages have been proposed to support Updating XML data. Such as Lorel[1], XML-GL[2], CXQuery[3], XQuery Update[4] and XUpdate[5]. But all these languages are based on so-called graph-based or tree-based data models, the level of which are too lower for end users to comprehend. This causes those update languages more complicated. Therefore, a novel data model for XML is proposed in [6], in which XML data is modelled in a way similar to complex object models. Based on this higher level data model, a rule-based declarative XML query language, XML-RL, is suggested in [7]. It proposes a high level data model for XML that allows users to view XML data as complex objects. There are five kinds of objects: (1) element objects representing elements with tag and value pair; (2) attribute objects representing attributes with name and value pair; (3) tuple objects representing the relationships among elements and attributes in XML; (4) list objects representing multiple values of an attribute or element; (5) lexical objects representing the constant values of an element or an attribute.

Based on the complex object model, The XML-RL query language is proposed. It is composed of rules, each of which has two parts, *querying* and *constructing*. The *querying* part is for querying while the *constructing* part is for constructing the result of the query. The structure is as follows:

Querying $qexp_1$, ..., $qexp_n$
Constructing cexp

where $qexp_1$,..., $qexp_n$ are querying expressions and *cexp* is resulting constructing expression. The following query is to find all faculties whose position is professor.

Querying (URL)/department/$f(faculty)[position⇒professor]
Constructing (URL)/department/$f

For saving space, we use URL instead of real url address.

2 XML-RL Update Language

An XML-RL update rules are composed of two parts: *querying* and *updating*. The *querying* part consists of a set of query expressions, which are used to query one or more XML documents. The result of the query could be bound to variables to hold XML objects. These variables can be used in the *updating* part as an updated object, updating context, and as the content of the update. The *updating* part consists of a set of update expressions, each of which is used to modify one or more XML objects from one or more XML documents. The syntax is as following:

Querying $qexp_1$, ..., $qexp_m$
Updating $uexp_1$, ..., $uexp_n$

where $qexp_1$, ..., $qexp_m$ are querying expressions and $uexp_1$, ..., $uexp_n$ are updating expressions.

We propose a set of primitive update operations, including *insertion*, *deletion* and *replacement* of atomic and complex XML objects.

2.1 Insertion

There are three primitive insertion operations, *insert-before*, *insert-after* and *insert-into*. The first two operations are used to insert an XML object to the selected XML object as its preceding or following sibling for ordered XML data. The last one is used to insert an XML object as a child at arbitrary position in the selected XML object for unordered XML data, or as a last child for ordered XML data. The following example inserts three new elements for a faculty element using two insertion operations.

Querying (URL)/department/faculty/
$n(name)[$f[firstName⇒Ayse], lastName⇒Alaca]
Updating insert-after($n) : workPhone⇒(613)226-5600,
insert-after($f) : middleName⇒Pear,
insert-before($f) : nickname⇒Beer

After updating, the *workPhone* element object is added after the *name* element, the *middleName* element object is added between *firstName* and *lastName*, and the nickname element object is added before the *firstName* element.

2.2 Deletion

A deletion operation is used to remove objects from an XML document. The deleted objects are usually the ones returned by the querying part, either objects or their values. We cannot remove the name of an existing object. When the value of an element is deleted, the value of object will be set to null. The following example is used to delete a *faculty* element.

Querying (URL)/department/$f(faculty)[name⇒[firstName⇒Ayse]]
Updating delete $f

When an element object is deleted, its sub-elements and attributes are removed recursively. If the object is referred by other objects, a cascading deletion is performed.

2.3 Replacement

The replacement operation is used to modify the existing objects. The modified object must be returned by the querying part before the replace operation can be applied. The replace operation can replace name of an object, the value of an object, or both, which depends on what the binding variable refers to. The following two examples are used to replace an object value and an object.

Querying (URL)/department/faculty⇒$f[$f⇒null]
Updating Replace ($f): (@id⇒F500, name⇒(firstName⇒Nency, lastName⇒White), position⇒instructor)

In the above update statement, the value of all null faculty objects is replaced with a complex values. The following example is to update complex objects.

Querying (URL)/department/$f(faculty⇒null)
Updating Replace ($f): faculty⇒(@id⇒F500, name⇒(firstName⇒Nency, lastName⇒White), position⇒instructor)

This update statement has the same function as the above one, but it updates the null faculty objects rather than the null value of the faculty objects, all other XML update languages cannot support the replacement of complex objects and their values because they cannot express complex objects and complex values due to the low level data models used.

2.4 Copy

Our language support copy an existing object from one document to another one, or from one place to another place in the same document. Following example suppose a faculty Smith is working in a department, and his data is at: URL1. Now assume he will also work in another department which address is at URL2. Therefore, we need to copy his data from the first department to the second department.

Querying (URL1)/department/$f1(faculty)/[name⇒[lastName⇒Smith]]
(URL2)/department/$f2(faculty)/[name⇒[lastName⇒Maculler]]
Updating insert-after($f2) : $f1

After updating, the data about Professor Smith is copied into the second department as the following sibling of Professor Maculler.

2.5 Mixed Updates

For mixed updates, we only allow delete objects in the current scope. In other words, mixed with multiple level updates can not delete parents of the current scope. This prevents situations in which users delete parents and then try to

update on their children. In the following example, we insert a new attribute and change the value of position to a faculty, delete a student element in the same department.

Querying (URL)/department/$f(faculty)[name⇒[firstName⇒Ayse,
lastName ⇒ Alaca]]/position⇒$p,
(URL)/department/$s(student)[name⇒[firstName⇒ Mery,
lastName ⇒ Lee]]
Updating insert-into($f) : @status⇒full time,
replace($p) : Professor,
delete $s

3 Conclusion

In this paper, we presented a novel XML update language. Our language has the following features. First, it is the only XML data manipulation language based on a higher data model. Therefore, update requests can be expressed in a more intuitive and natural way. Second, our language is designed to deal with ordered and unordered data. Third, the update operation can be applied to more than one XML documents in the same time. This provides a natural way to exchange data between XML documents, even these documents are reside over the network. Fourth, our language can express complex update requests at multiple level in a hierarchy in a simple and flat way. Fifth, our language directly supports the functionality of updating complex objects. Lastly, Our language modifies tag names, values, and objects in a unified way by using logical binding variables. We are currently working on the implementation of the XML-RL update language.

References

1. Abiteboul, S., Quass, D., Mchug, J., Widom, J., Wiener, J.: The Lorel Query Language for semistructured Data. Int'l Journal on Digital Libraries. 1 (1997) 5-19
2. S. Ceri, S. Comai, E. Damiani, P. Fraternali et al. XML-GL: A Graphical Language for Querying and Restructuring XML Documents. Proceedings of the 8th International World Wide Web Conference, Toronto, Canada, 1999.
3. Y. Chen and P. Revesz. CXQuery: A Nodel XML Query Language. This is available at `http://citeseer.nj.nec.com/539624.html`.
4. I. Tatarinov, Z.G. Ives, A.Y. Halevy, D.S. Weld. Updating XML. In Proceedings of 2001 SIGMOD International Conference, Santa Barbara, CA, USA, 2001.
5. A. Laux, L. Martin. XUpdate - XML Update Language. W3c Working Draft, 2000. It is available at `http://www.xmldb.org/xupdate/xupdate-wd.html`.
6. M. Liu. A logical foundation for XML. In Proceedings of the 14th Int'l Conf. on Advanced Information Systems Engineering, Toronto, Canada, 2002, 568-583.
7. M. Liu and T.W. Ling. Towards declarative XML querying. In Proceedings of The 3rd Int'l Conference on Web Information Systems Engineering, Singapore, 2002.