Update XML Data by Using Graphical Languages

Wei Ni and Tok Wang Ling
Department of Computer Science, School of Computing,
National University of Singapore, Singapore
Computing 1, Law Link, Singapore, S117590
{niwei, lingtw}@comp.nus.edu.sg

Abstract

To be a full-featured data exchange format, XML should support not only queries but also updates on its contents. The new W3C XML update facility has proposed a set of operators (insert, delete, replace and rename) and expressions to modify XML data. However, the new update standard is an extension of XPath/XQuery. As a result, it requires the full knowledge about complex XPath/XQuery writing techniques, which is too difficult for common users to use. In this paper, we intuitively represent XML update expressions as graphs. Based on our previous works of GLASS (Graphical query LAnguages for Semi-Structured data), we make an extension, named as GLASS^U, to support the new XML update facility. To our best knowledge, it is the world first graphical language that supports XML updates.

Keywords: XML, Update, Graphical Language

1 Introduction

XML [14] is now fast becoming the standard format of web data storing, exchanging and publishing. Today, all leading database management system (DBMS) vendors have released their new products supporting XML data storing, publishing and querying. However, to become a full-featured data exchanging standard, XML has to support not only queries but also updates on data contents.

Brief history of XML updates: XML update is not a new problem. Various works have been proposed for XML updates in different database systems since late 1990s along with the birth of XML. *Lorel* [1] system and its language support XML update in an OQL-style language and it has been implemented on the base of an object-oriented DBMS. The prototype of the W3C's XML update facilities can be traced back to the working draft [6] in year 2000. This working draft, also known as *XUpdate*, has a set of update operators including insert (with before or after), append (i.e. to insert as the last child element), remove, and update. After that, the research works in [13] has discussed the co-operation between XQuery and update operators including insert, delete, update, rename and replace, which was focused on the method of implementing update on the XML data that were mapped and stored in relational DBMS. Besides, the *extended XML-RL* [8] has proposed a declarative XML update language based on XML-RL which is originally a declarative XML query language. *XPathLog* [9] is a rule-based data manipulation language that only supports element creation and modification.

Based on the existing research works, W3C has released the new standard called *XML update facility* [17] in July 2006. The new standard has formally defined 4 update operators (insert, delete, replace and rename) and a new operator called transform. The transform operator will create a modified copy of the original data. The term "copy" here is especially a view and the transform expression can be regarded as a "create view" clause in XML[1]. In this paper, we use the term "view" instead of "copy" to avoid ambiguity.

Our motivation: Update expressions do not work alone. They always appear along with XPath[15]/XQuery[18] expressions combined with conditions. Therefore, a user should have the full knowledge of XPath/XQuery writing techniques before hi/she can use the new update facilities. Unfortunately, the complexity of writing XPath/XQuery expressions makes it difficult for common users to pose their XML updates.

Yet fortunately, XML data has a tree-like structure that can be naturally represented as graphs. In fact, graphs have been widely used in almost all fields of XML data applications such as modeling [7, 10], editing [12], and querying [3, 4, 11] in the past few years. Inspired by the above research works and successful applications, we propose to represent XML updates, i.e. update expressions and their conditions, in graphs to help common users manipulate their XML data more easily than using XPath/XQuery expressions.

Like textual languages, where the XML update is an extension of existing XML query languages, our graphical XML update language is also an extension of existing graphical XML query languages. In [11], we have proposed a graphical XML query language, which is named as *GLASS* (Graphical query LAnguage for Semi-Structured data). The GLASS was designed on the base of *ORA-SS* (Object-Relationship-Attribute model for Semi-Structured data) [7], a rich semantic data model for XML. We select GLASS to do the XML update extension because of two reasons. The first reason is that GLASS has the most powerful expressive capability in comparison current existing graphical XML query languages. It supports not only the traditional queries

[1] Notice that there is no "create view" clause in current XML query standard. Therefore, the transform expression just plays role of "create view" in XML and the "copy" here may not be a materialized one.

such as selection, projection, join and aggregation but also advanced features including swapping, quantification, negation and conditional output construction (if-then clause). Also, with the semantics in ORA-SS, GLASS is concise and clear for query representation. The second reason is the rich semantic information captured in ORA-SS. Traditional XML schema languages, e.g. DTD[14]/XSD [16] and their equivalents, do not represent the semantics in XML data such as *object ID*[2], *relationship types* and *relationship type attributes*. Such semantic information is extremely important for machines (software) to precess XML query correctly. In the scenario of XML updates, the semantic information is still important to achieve valid update result with respect to both structural and semantic constraints.

In this paper, we mainly focus on the extension work for XML updates, which is named as "**GLASS**$^{\mathbf{U}}$". We propose this extension to make XML update expressions more intuitive and easier to use.

Organization: The rest of this paper is organized as follows. In Section 2, we give some preliminary knowledge about the new W3C XML update facility, the ORA-SS and GLASS. In Section 3, we introduce our extension work for graphical XML update expressions. With examples, we demonstrate how to express the new XML update facility in our graphical expressions. In Section 4 we make a brief summary of this paper and highlight future works on this direction.

2 Preliminary knowledge

In this section, we introduce the formal semantics of the 5 operators in the new W3C update facilities. After that, we also give a brief review of our previous works in XML data modeling (ORA-SS) and query representation (GLASS).

2.1 W3C XML update facility

There are four XML update operators: *insert*, *delete*, *replace*, *rename*; and one auxiliary operator - *transform* to make copies (views) for modification instead of the source data.

Terms in explanation: All preserved keywords are written in capital letters. The *src_exp* and *tgt_exp* refer to source expression and target expression in XML updating respectively. The tgt_exp is the navigation to the place(s) where the update operation should be applied. It is a compulsory component in every update expression. In contrast, the src_exp refers to those newcomers in the result after the update operation, which is often used in the insertion and replacement.

The general format and the semantics of the five operators are explained as follows.

(1) Insert

The general format of the insert expression is

DO INSERT *src_exp* ***prep*** *tgt_exp*

which means to insert the node(s) with its (their) substructures described by the *src_exp* into a document at a certain position about the node(s) specified by the *tgt_exp*. The ***prep*** is the preposition that specifies the position of the insertion, which can be INTO (by default), BEFORE, AFTER, AS FIRST and AS LAST.

(2) Delete

The general format of the delete expression is

DO DELETE *tgt_exp*

which means to delete the sub-trees rooted at the node(s) specified by the *tgt_exp*.

(3) Replace

The replace has two different formats for either semantics of replacing nodes with sub-structures or modifying the values of certain nodes.

The *first* format, corresponding to the first semantic meaning, is

DO REPLACE *tgt_exp* WITH *src_exp*

which means to replace the node(s) with sub-structures specified by the *tgt_exp* with the node(s) with sub-structures described by the *src_exp*. The *src_exp* nodes will take the hierarchical position of the nodes of *tgt_exp*.

The *second* format, corresponding to the second semantic meaning, is

DO REPLACE **VALUE OF** *tgt_exp* WITH *src_exp*

which means to modify the value (i.e. PCDATA or CDATA content) of the node (i.e. instance of XML element or attribute respectively) specified by the *tgt_exp* with the value of the *src_exp* without changing the original node's name, position and its sub-structures in the document tree.

(4) Rename

The general format of the delete expression is

DO RENAME *tgt_exp* AS *new_name*

where the *new_name* can be either a string or a variable. The rename operator only changes the tag name of an XML element or the name of an XML attribute.

(5) Transform

The transform operator is NOT an update operator. It is used to create a view to which the above 4 XML update operators will be applied. The format of the transform operator is

TRANSFORM *$var* := *src_exp* MODIFY *update_exp*

where *$var* is the user-defined name (in form of variable) of the created nodes in the view with sub-structures specified by *src_exp*; *update_exp* is the expression(s) of the above 4 XML update operators we have listed.

With the transform expression, the updates are used to create a view with the nodes of *$var*. Therefore, the *update_exp* in the transform expression will not be applied to the original XML document.

2.2 ORA-SS and GLASS

The ***ORA-SS*** is a rich semantic data model for XML data. In comparison with other XML data models such as DOM [2], OEM [10], Dataguide [5], XML Graph [4], etc., ORA-SS can capture and express much richer semantics in XML data such that:

(1) It models the XML data into object classes, relationship types and attributes.
    (a) Object classes are for those internal nodes (complex

[2] In ORA-SS, the object concept refers to real world objects and the object ID identifies different real world object instances, not just for element instances.

XML elements that contain child elements and/or XML attributes) in an XML document tree.

(b) Relationship types are the relationship types among different object classes on one path.

(c) Attributes are usually those leaf nodes (XML attributes and simple XML elements with PCDATA only) in an XML document tree.

(2) It differentiates the attributes of object classes from those of relationship types.

(3) It uses object ID attributes instead of the key, which is more convenient in use.

(4) It can express *n*-ary ($n \geq 2$) relationship types among different object classes on one path.

(5) It expresses the participation constraints of object class instances in n-ary relationship types.

(6) It expresses the functional dependency, object class inheritance and other semantics.

All these above semantics are crucial for XML storage, query and management in DBMS. However, the DTD/XSD and previously mentioned XML data models do not express these important semantics. Only ORA-SS catches them all by using its schema diagram, functional-dependency diagram and inheritance diagram.

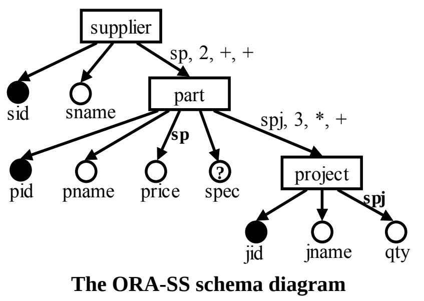
In this paper, we mainly use the schema diagram in ORA-SS, which is also the core component of the data model.

The Example 1 in Fig. 1 gives a comparison between DTD and ORA-SS schema diagram. In Fig. 1, the complex XML element *supplier*, *part* and *project* are modeled as object classes; all other simple XML elements and XML attributes are modeled as attributes. The filled circle means this ORA-SS attribute is the object ID attribute of the corresponding ORA-SS object class.

Example 1. (ORA-SS schema diagram & DTD of "spj.xml")

```
<!ELEMENT supplier (part+)>
  <!ATTLIST  supplier  sid  ID #REQUIRED>
  <!ELEMENT sname PCDATA>
  <!ELEMENT part (price, spec?, project*)>
    <!ATTLIST part  pid  CDATA #IMPLIED
                    pname  CDATA>
    <!ELEMENT price PCDATA>
    <!ELEMENT spec PCDATA>
    <!ELEMENT project (jname, qty)>
      <!ATTLIST project  jid  CDATA #IMPLIED >
      <!ELEMENT jname PCDATA>
      <!ELEMENT qty PCDATA>
```

The DTD schema



The ORA-SS schema diagram

Fig. 1. An ORA-SS schema diagram showing the binary and ternary relationship types.

There are two relationship types defined in the schema diagram.

The label "sp, 2, +, +" means: "sp" is a binary relationship type between supplier and part where each supplier supplies one or many different parts and one part can be supplied by one or many different suppliers.

The label "spj, 3, *, +" means: "spj" is a ternary relationship type among supplier, part and project, where each combination of supplier and part corresponds to zero or many different projects and one project is associated with one or many different combination of supplier and part instances.

The last but the most important semantic information in Fig. 1 is the difference between object class attributes and relationship attributes. By default, an attribute is associated with its parent object class. However, some attributes are not only determined by their parent object classes. For example, the price value is determined by the combination of supplier and part instances. Thus, the price is an attribute of the binary relationship type sp. Similarly, the qty (quantity) is an attribute of the ternary relationship type spj, which means the qty value is determined by the combination of supplier, part and project instances along the same path in the XML document tree.

The ***GLASS*** is a graphical XML query language proposed on the base of ORA-SS. A typical GLASS query consists of 4 parts:

(1) The left hand side graph (LHS graph) indicates the query conditions.

(2) The right hand side graph (RHS graph) specifies the result construction.

(3) The links explicitly declare the mapping between the components in LHS and RHS graphs.

(4) The condition logic window is an optional field for user to write complex query conditions such as quantifiers, negations, logic/mathematic expressions and conditional output construction (e.g. IF-THEN clause).

Query 0: (*An GLASS query example*)

Find all suppliers who supply some parts to the project "J001"; display the supplier with sid, sname, all parts they supply (not only those for J001) with pid, pname and price.

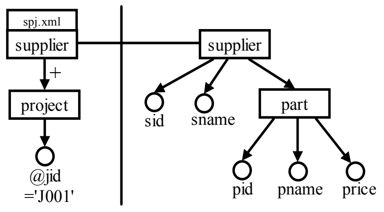


Fig. 2. The GLASS query graph for Query 0.

3 GLASS^U

A user is supposed to know the data schema and semantics before he/she proposes an XML update. In this paper, the XML schema is assumed to be represented by ORA-SS schema diagram.

3.1 The notations for XML updates

In comparison with our previous works of GLASS, the graphical representation of XML updates should be more precise in specifying paths and positions (especially for insertion) in an XML document tree. It also requires a one-to-one mapping from XML components to visual notations. To meet the requirement of XML updates, we add some new notations.

In Table 1, we list all the notations of graphical XML updates. It should be emphasized that,

(1). The attribute in ORA-SS (the circle) can be either a simple element or an attribute in an XML document tree. However, when drawing a graph for updating or querying, we should specify the original data type exactly. Thus, we use the circle with "@" to distinguish XML attributes from simple XML elements.

(2). The *Value* node (the filled triangle) is important for XML updates with respect to the semantics of the keyword "VALUE OF". If it is below an element, it means the PCDATA content of the element; if it is below an attribute, it means the CDATA value of the attribute.

(3). The label of the *Action* node (curved rectangle) can be the 4 basic update operators: *insert, delete, replace* and *rename*.

XML and XML update component s	Notations for XML updates in GLASS^U	Notation name	Notation meaning
Complex XML element	$\boxed{\text{Supplier}}$	Rectangle	Supplier is a complex element in XML
Simple XML element	○ Sname	Circle	Sname is a simple element in XML
XML attribute	○ @Sid	Circle with "@"	Sid is an attribute in XML
Action, the XML update operator	target ← (INSERT) — source	Curved rectangle	An INSERT action
Value of XML element or XML attribute	▲	Filled triangle	-
Functions, mathematic formulas	⬡ +1	Hexagon	To increase the value by 1
Parent-child relationship ("/")	$\longrightarrow$	Arrow	-
Ancestor-descendant relationship ("//")	$\xrightarrow{+}$	Arrow (with "+")	-
IDREF/IDREF S	$\dashrightarrow$	Dashed Arrow	

Table 1. The visual notations and their meanings for XML updates.

3.2 The graphical XML update expressions

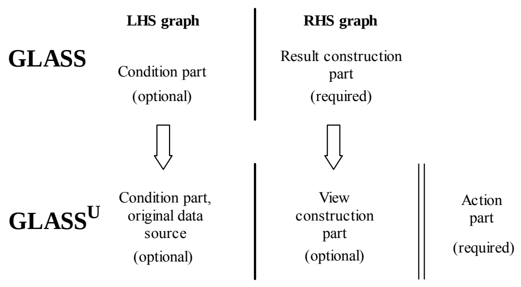
An XML update graph consists of 3 parts, from left to right, they are:

(1). **Condition part** - indicates where the update operation shall be applied, which is an optional part for updates;

(2). **View part** - generates a view from the condition part for updating, the specification of output, which is an *optional* part when the update is applied to the source data;

(3). **Action part** - specifies what kind of update operation is applied and the corresponding sources for modification, which is a *required* part for any graphical update expressions.

The condition part and view part are separated by a single vertical line; and the view part and action part are distinguished by a double vertical line. Fig. 3 indicates the relation between the extension GLASS^U and the original GLASS.



LHS graph
RHS graph
GLASS
Condition part (optional)
Result construction part (required)
GLASS^U
Condition part, original data source (optional)
View construction part (optional)
Action part (required)

Fig. 3. The comparison between the structures of GLASS and GLASS^U.

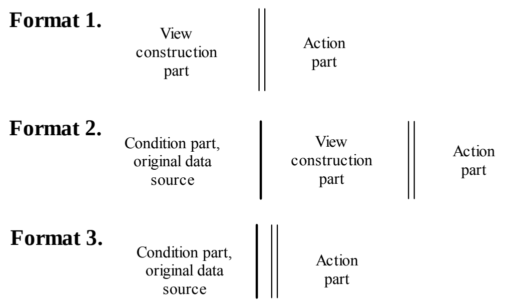


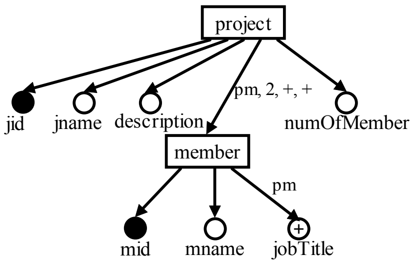
Fig. 4. The 3 different formats of GLASS^U expressions

Although both the condition part and view part are optional in the extended graphical representation for XML update, they cannot be omitted at the same time. Moreover, when the view part is omitted, the single vertical line between the condition part and view part should not be omitted. We illustrate the three different general formats of GLASS^U expressions in Fig. 4, where the update action can be either applied to the source data or used to construct a view with respect to transformation.

Example 2. (The DTD and ORA-SS schema diagram of "project.xml")

```
<!ELEMENT project (jname, description, member+,
                    numOfMember)>
  <!ATTLIST  project jid ID #REQUIRED>
  <!ELEMENT jname PCDATA>
  <!ELEMENT description PCDATA>
  <!ELEMENT member (mname, jobTitle+)>
     <!ATTLIST member mid CDATA #IMPLIED >
     <!ELEMENT mname PCDATA>
     <!ELEMENT jobTitle PCDATA>
  <!ELEMENT numOfMember PCDATA>
```

The DTD schema



The ORA-SS schema diagram

Fig. 5. The ORA-SS schema and DTD of the XML document "project.xml"

Format 1 is used when the update action is applied to construct a view.

Format 2 is the general format that contains both the condition part and the view part. In this format, a view is generated according to the specification in the condition part. If the target arrow of the action part points to the source (condition) part, it means the update should change the source data. Otherwise, if the target arrow points to the view part, the update is applied to construct a view only.

Format 3 is used when the update action is applied to the source data, probably the most common format of an update. With this format, all contents in the source data, specified by the condition part, will be changed by the update action. Notice that, after such an update, the changes in the source data should be propagated to all its derived views in a DBMS. The update propagation is important but beyond the discussion in this paper.

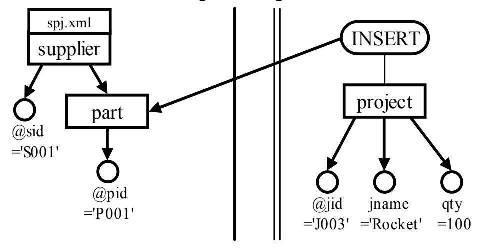
In our following discussion, all query examples are proposed on the XML data "spj.xml" and "project.xml" with their ORA-SS schemas in Example 1 and Example 2.

(1) Insertion, target arrow and source link

Query 1. In the "spj.xml", insert a new project instance that the supplier "S001" supplies the part "P001" to the new project; and the quantity of the part is 100 unit.

```
FOR $p IN doc("spj.xml")/supplier[@sid= "S001"]
                    /part[@pid = "P001"]
RETURN
    DO INSERT
        <project @jid = "J003">
            <jname>Rocket</jname>
            <qty>100</qty>
        </project >
        AFTER
        $s/sname
```

XML update expression



GLASS^U update expression

Fig. 6. The XML update expression and our graphical representation of Query 1.

In Fig. 6, the graphical update expression is in Format 3 (See Fig. 4.), which consists of condition part and action part (the view part is omitted).

In the action part, there is an INSERT action with two out-stretching edges. The first edge is a line that links the action with the new project instance. This line, which we call "*source-line*", is used to link the action and the graphical expressions corresponding to the *src_exp*. The second edge is an arrow which points to the *part*. The arrow, which we call "*target-arrow*", points to the target node and/or target position of the action corresponding to the *tgt_exp* in Section 2.1. By default, the target arrow points to the parent node with the default preposition "INTO". We can add the key word label "AS FIRST" or "AS LAST" on the target arrow, which indicates that the inserted instance will the first/last child element of the target node. Only when the preposition is "BEFORE" or "AFTER", the target arrow should point to the node for reference which is in fact the sibling node of the inserted instance.

(2) Replacement and function

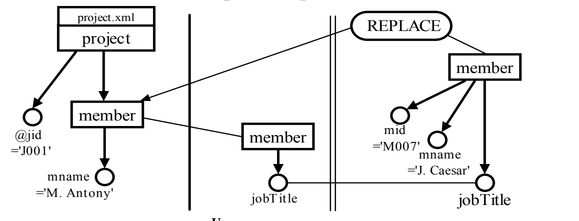
Query 2. In "project.xml", project "J001", the member "M. Antony" is replaced by "J. Caesar" and "J. Caesar" should take all job titles of "M. Antony". (*without* **VALUE OF**)

Query 3. In the "spj.xml", increase the price value of part "P001" under each supplier by 10%. (*with* **VALUE OF**)

The example of Query 2 demonstrates the semantics of replacing nodes without using the keyword "VALUE OF". Because the original member "M. Antony" will be replaced, we should remember his job titles in project "J001" before it is replaced. To achieve this, we have to use the transform expression to create a temporary view. After that, we construct the new member instance "J. Caesar" and put the job titles in the temporary view under "J. Caesar". Finally, we replace the new instance to the old one in the original data. The GLASS^U expression in Fig. 7 is of Format 2.

```
FOR $j IN doc("project.xml")
        /project[@jid = "J001"]
LET $m := $j/member[mname = "M. Antony"]
RETURN
    TRANSFORM
      COPY $c := $m
    DO REPLACE
      $m
    WITH
      <member @mid = 'M007'>
          <mname>J. Caesar</mname>
          $c/jobTitle
      </member>
```

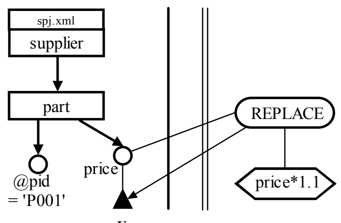
XML update expression



GLASS^U update expression

Fig. 7. The XML update expression and our graphical representation of Query 2.

```
FOR $p IN
doc("spj.xml")/supplier
        /part[@pid = "P001"]
RETURN
  DO REPLACE VALUE OF
      $p/price
    WITH
      $p/price*1.1
```



XML update expression **GLASS^U update expression**

Fig. 8. The XML update expression and our graphical representation of Query 3.

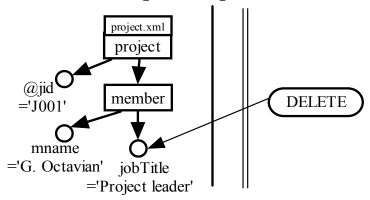
In comparison, Query 3 is a replacement on the value of an XML node. To express the semantics of "VALUE OF", we use the value node (the filled triangles). In the condition part, the triangle below the price means "the value of the price element"; and in the action part, the target-arrow points to the triangle under the price in the condition part. The source-line that links to the price attribute indicates which nodes are involved in the function or mathematic formula in the hexagon. This is important when there are several different attributes are involved or the source node appears under different object class nodes.

(3) Deletion

Query 4. **(Deletion)** In the "project.xml", below the project "J001", the member "G. Octavian" will no longer be the project leader. Thus, we will delete the corresponding job title.

```
FOR $oct IN doc("project.xml")/project[@jid= 'J001']
                              /member[mname='G. Octavian']
$jt IN $oct/jobTitle
WHERE $jt = "Project leader"
RETURN
     DO DELETE
          $jt
```

XML update expression



GLASS^U update expression

Fig. 9. The XML update expression and our graphical representation of Query 2.

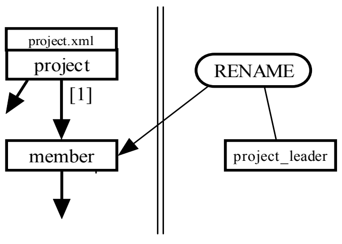
In comparison with the other update operators, the delete operator always has only one operand – the target node(s) to delete.

(4) Rename and Transformation

Query 5. Create a view from "project.xml", rename the first member element of each project as "project_leader".

```
FOR $p IN
doc("project.xml")/project
RETURN
     TRANSFORM
       COPY $tp := $p
     DO RENAME
       $tp/member[1]
     AS
       "project_leader"
     RETURN $tp
```

XML update expression



GLASS^U update expression

Fig. 10. The XML update expression and our graphical representation of Query 5.

The graphical XML update expression in Fig. 10 is in Format 1 (in Fig. 4) that only contains the view part and action part. Therefore, we create a view of the original data; and, within the view, we rename the first member element under each project element as project leader, which will not change the source data.

4 Summary

In this paper, we focus on the problem of XML update representation. Our extension work in this paper brings contributions in two folds. On one hand, we inherit the fruit of an existing graphical query language: GLASS and re-use its expressive power to represent complex path navigation and/or query conditions. One the other hand, the extension enhances the graphical query language into a full-featured one that now support both data query and data update.

Related Works: There has been several research works proposed on the subject of graphical XML query representation such as the graphical XML query languages [3, 4, 11] and graphical user interfaces (GUIs) [12]. However, none of the current existing graphical XML query languages includes the XML update. To our best knowledge, our extension of GLASS, namely GLASS^U, is the first extension of a graphical XML query language to represent XML update expressions.

Future Works: The XML update brings the problem of validation. As we have mentioned that we choose ORA-SS model because the rich semantics in ORA-SS is the base of the validation of semantic constraints. Therefore, one future research direction is the semantic validation of XML updates, especially for data centric XML data.

References:

[1] Abiteboul, S., Quass, D., McHugh, J., Widom, J., and Wiener, J. (1996): The Lorel query language for semistructured data. *Journal on Digital Libraries*, 1(1).

[2] Apparao, V. and Byrne, S. (1998): Document object model (DOM) level 1 specification. W3C Recommendation.

[3] Braga, D., Campi, A., and Ceri, S. (2005): XQBE: A Visual Interface to the Standard XML Query Language. *ACM Transactions on Database Systems*, Vol. 30, No. 2, p398-443.

[4] Ceri, S., Comai, S., Damiani, E., Fraternali, P., Paraboschi, S., and Tanca, L. (1999): XML-GL: a graphical language of querying and restructuring XML documents. *Proc. WWW8.*

[5] Goldman, R. and Widom, J. (1997): Dataguides: Enabling query formulation and optimization in semi-structured databases. *Proc. VLDB1997.*

[6] Laux, A., Martin, L. (2000): XUpdate - XML Update Language. W3C Working Draft.

[7] Ling, T. W., Lee, M. L., Dobbie, G. (2005): *Semistructured Database Design.* Springer Science + Business media, Inc.

[8] Liu, M., Lu, L., Wang, G. (2003): A Declarative XML-RL Update Language. *Proc. ER 2003.*

[9] May, W. (2001): XPathLog: A Declarative, Native XML Data Manipulation Language. *Proc. IDEAS'01.*

[10] McHugh, J., Abiteboul, S., Goldman, R., Quass, D., and Widom, J. (1997): Lore: A database management system for semi-structured data. *SIGMOD Record*, 26(3):54-66.

[11] Ni, W. and Ling, T. W. (2003): GLASS: A Graphical Query Language for Semi-Structured Data. *Proc. DASFAA2003.*

[12] Papakonstantinou, Y., Petropoulos, M., and Vassalos, V. (2002): QURSED: Querying and Reporting Semistructured Data. *Proc. ACM SIGMOD.*

[13] Tatarinov, I., Ives, Z.G., Halevy, A.Y., and Weld, D.S. (2001): Updating XML. *Proc. ACM SIGMOD.*

[14] Extensible Markup Language (XML) http://www.w3.org/XML/ Accessed Aug 2007.

[15] XML Path Language (XPath) http://www.w3.org/TR/xpath Accessed Aug 2007.

[16] XML Schema http://www.w3.org/XML/Schema Accessed Aug 2007.

[17] XQuery Update Facility W3C Working Draft 11 July 2006 http://www.w3.org/TR/2006/WD-xqupdate-20060711/ Accessed Aug 2007.

[18] W3C XML Query (XQuery) http://www.w3.org/XML/Query/ Accessed Aug 2007.