Design and Evaluation of an Example-based Graphical Manipulation Framework for XML

Atsuyuki Morishima
Dept. of Info. Sci. and Eng., Shibaura Inst. of Tech.
Saitama, Saitama, Japan
amori@sic.shibaura-it.ac.jp

Hiroyuki Kitagawa
Inst. of Info. Sci. and Elec., Univ. of Tsukuba
Tsukuba, Ibaraki, Japan
kitagawa@is.tsukuba.ac.jp

Abstract

This paper explains a novel manipulation framework for XML. The key idea is to choose some example XML elements existing in the database and show the system how to manipulate them. The system then infers how to manipulate the whole collection of XML documents. The framework is unique in that while other approaches require users to write (or draw) explicit query specifications in their own query languages, ours needs implicit specifications through example operations. The problem is challenging, because XML documents can be semistructured data and inferring the intended operation is not trivial. Our algorithm is based on tree-style object modeling and path-expressions with wildcards to infer the user's intention. The paper presents the object modeling, its inference mechanism, and some results of our preliminary experiments.

1. Introduction

XML has become the standard format for data exchange and, consequently, development of query languages for XML has become one of the hottest issues in research and industrial communities. Well-known languages include XML-QL[6], XQuery[12], Quilt[5], XQL[10], XML-GL[4] and XSLT[11]. They are textual or graphical languages in which we can specify queries for XML manipulation. This paper explains a novel graphical manipulation framework for XML. The key idea is to choose some example XML elements existing in the database, and show the system how to manipulate them. The system then infers how to manipulate the whole collection of XML documents. The framework is unique in that while other approaches require users to write (or draw) explicit query specifications in their own query languages, ours needs implicit specifications through example data operations. The problem is challenging, because XML documents can be semistructured data [1][3] and inferring the intended operation is not trivial. For example, because the data structure is often irregular and implicit in semistructured data, the domain of objects an example represents cannot be fixed in advance. In contrast, the domains are fixed in advance in QBE [13] and other QBE-style query languages for relational databases. In our framework, the domain is defined dynamically according to the user's interaction with the system. By specifying 'another' example, the user allows the system to extend the intended domain. This feature is essential for operation of semistructured data. Our algorithm is based on tree-style object modeling and path-expressions with wildcards. The contributions of this paper are as follows: (1) We explain the XML manipulation through operations of example data in the database, and how the system generalizes the operations. (2) We show the results of our preliminary experiments to compare our framework to XML-QL and XML-GL, using a prototype system developed in our project. We proposed a preliminary version of the example-based semistructured data manipulation framework in the context of Web view construction in [8]. The framework presented in this paper is an extended and refined version and directly applied to XML document manipulation. In particular, it allows artificial examples and its semantics is simpler and more general. We explain the details later.

The remainder of this paper is organized as follows: Section 2 gives an example scenario. Section 3 explains how to manipulate XML documents in our framework. Section 4 illustrates how the user interacts with the system in the example scenario. Section 5 gives the formal semantics. Section 6 shows the results of our preliminary experiments. Section 7 is the conclusion.

2. Example Scenario of XML Manipulation

Suppose that we have two sets of XML documents in an XML repository (Figure 1). The documents in the first set contain information on academic publications (Figure 1(a)). Each document has a list of publications for one particular year. We assume those documents are *semistructured*. For example, some documents have their publications grouped by project (Figure 2(b)) where a `pub` element is a child of a *project* element, while others have a flat structure (Figure 2(a)) where a `pub` element is a direct child of a `pubs`

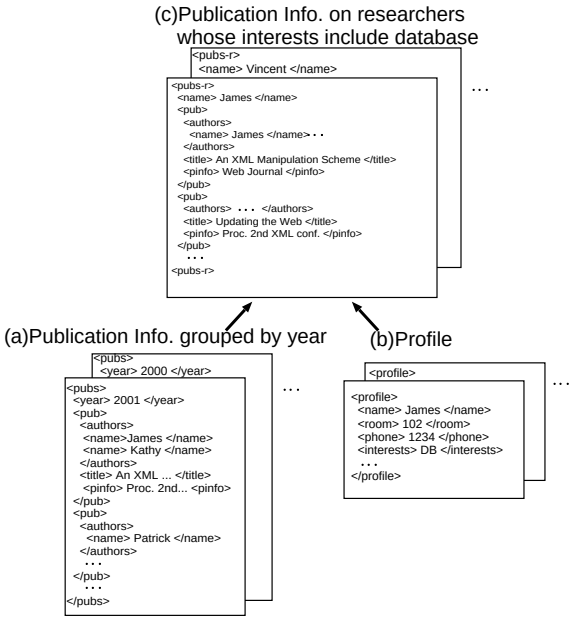

Figure 1. Example scenario

element.

The documents in the second set contain profile information on researchers of some research group (Figure 1(b)). The profile information includes their names, room and phone numbers, and their academic interests.

The requirement here is to construct another set of XML documents, each of which corresponds to a researcher in the research group whose interests include DB (database) and contains the list of the researcher's publications.

3. XML Manipulation through Example Operations

Our framework provides users with the following two types of windows for interaction:

DataBox: A *DataBox* is used to display a set of XML documents in the database. Figure 3 shows example DataBoxes. The DataBox (a) is used to display XML documents of publication information. The DataBox (b) is used to display the profile XML documents. A DataBox shows one XML document at a time. The user can click the Next and Previous buttons to browse other documents. We call an XML document in a DataBox just *a page*.

Canvas: The *Canvas* is a blank window onto which the user can drop data objects from DataBoxes.

3.1. Drag-and-Drop, Examples and Target Sets

Figure 4 is a simple operation example[1]. *Drag-and-Drop* is denoted by the dotted-lines. Here, we refer to the item being dragged and dropped as *an object*. XML elements and contents within the elements are objects. Objects dropped onto the Canvas appear in the result.

An example is denoted by the oval. The user can designate an object as an example by selecting the "Example" menu item that appears when the right mouse button is clicked over the object (Figure 5) before the drag-and-drop operation.

An example has its *target set*, which is the set of objects the example represents. Manipulation of an example is interpreted as manipulation of the objects in its target set. Objects in a target set are highlighted in each DataBox.

As a default, the target set is defined as a set of objects which appear 'at the same position' on their pages as the example object. For example, in Figure 5, the target set would be the set of researchers' names in the profile XML

[1] In the figure, multiple pages are shown simultaneously in a DataBox for explanation purposes. In reality, the Next and Previous buttons are used to view them.

```
<pubs>
  <year>2001</year>
  <pub>
    <authors><name>James</name>
             <name>Kathy</name></authors>
    <title>An XML manipulation..</title>
    <pinfo>Proc. 2nd XML Conf.,..</pinfo>
  </pub>
  <pub>
    <authors><name>Patrick</name></authors>
     ...
  </pub>
</pubs>
```

(a) Publication list with a flat structure

```
<pubs>
  <year>2000</year>
  <project>
     <pname>Web Integration</pname>
     <pub>
        <authors><name>Vincent</name></authors>
        <title>A Language for the Web</title>
        <pinfo>Web Journal ..</pinfo>
     </pub>
     <pub> ... </pub>
  </project>
     ...
</pubs>
```

(b)Publication list with a nested structure

Figure 2. Publication Info. variations

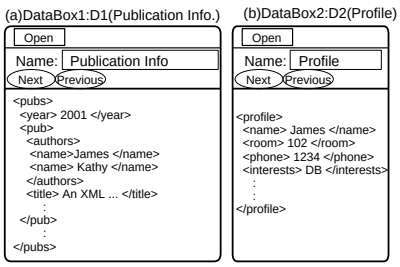

Figure 3. DataBoxes

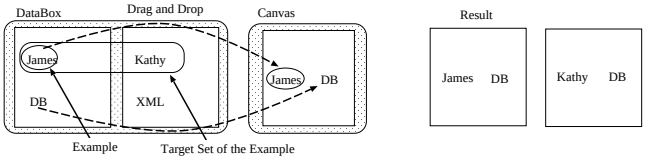

Figure 4. Manipulation of an example

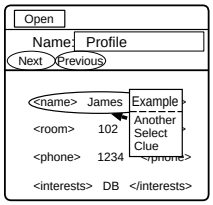

Figure 5. Menu to specify examples

documents. “Another” and “Select” menu items are used to directly modify the target set. “Clue” is used to construct another target set, which serves as ‘clues’ to select objects in the original target set.

The following regular expression shows the operation procedure in our framework.

(‘Example’ (‘Another’ | ‘Select’ | ‘Clue’)* | ‘D&D’)*

Here, ‘Example,’ ‘Another,’ ‘Select’ and ‘Clue’ mean selections of respective menu items on an object. The ‘D&D’ is a Drag-and-Drop operation. We call them ‘Example,’ ‘Another,’ ‘Select,’ ‘Clue’ and ‘D&D’ operations. Intuitively, our framework allows any combinations of the following operation patterns: (a) To designate an object as an example, and accept the default target set. (b) To designate an object as an example, and change the default target set by successive ‘Another,’ ‘Select’ and ‘Clue’ operations. (c) To drag-and-drop an non-example object (an object which the user did not designate as an example) onto the Canvas. In this case, only the object appears in the result. (d) To drag-and-drop an example object onto the Canvas.

If ‘Another’ operation is performed after ‘Example’ operation, the target set of the example is extended to include the ‘Another’ object and other objects that the system infers should be included into the target set. Intuitively, the system tries to generalize the relationship between the position of the example and that of the ‘Another’ object, and includes all the objects having the generalized relationship with the example into the target set.

If ‘Select’ operation is performed after Example operation, the system makes the target set smaller by selecting objects that meet a given condition in the target set. A window appears on the display so that the user can enter selection conditions. For example, he can enter the condition “= ‘J*.’ ” As a result, only the authors whose names start with ‘J’ become the members of the target set.

After ‘Example’ operation, the user can choose another object and perform ‘Clue’ operation on it. The system then constructs another target set whose example object is the designated object. (We call it the ‘Clue’ example.) The new target set is associated to the original target set[2], and mainly used to make the size of the original target set smaller, by serving as a selection condition. This is done with the following ‘Select’ operation on the Clue example. For example, if he specifies ‘Clue’ operation on James’s `<interests>DB</interests>` element, the system constructs a new target set whose example is the `<interests>` element. Members of the new target set are all the `<interests>` elements of all the researchers. Then, the user can perform ‘Select’ operation on the James’s `<interests>DB</interests>` element (i.e., its ‘Clue’ example object), and enter the condition “contains ‘DB’.” As a result, the original target set is reduced to contain only the names of researchers whose interests include “DB.”

3.2. Associations

When the user specifies multiple examples (and their target sets), there are usually *associations* among them. If an association occurs among target sets, only particular combinations of objects are qualified to be manipulated. Therefore, an association serves as a kind of join condition. Our framework supports two types of associations.

The first is *structural association* (S-Association). Two examples have an S-Association when their positions have some special relationship. One of the simplest cases is that two different examples on the same page imply an S-Association that two objects taken from their two target sets are related with each other only if they are on the same page. Suppose that the user wants to restructure pages in the DataBox in Figure 6 so that the name and his interests are arranged side by side. If the user takes example objects as in Figure 6, the system considers that there is no S-Association between the objects of the two target sets. Therefore, all combinations (Cartesian product) of two objects, one from target set A and the other from B, appear in the result. In contrast, if he takes examples as in Figure 7, an S-Association is implied and he gets the intended result.

The second type of association is *value association* (V-Association). Two examples have a V-Association when their values are the same. Suppose that we have two DataBoxes (Figure 8), and that the user wants a set of pages, each of which contains a researcher’s name, his room number, and his interests. If the user takes examples like those in Figure 8, a V-Association occurs and he gets the intended result. (Note that target sets A and B

[2] Its default target set is defined according to the original target set. Detailed discussion is given in Subsection 5.2.

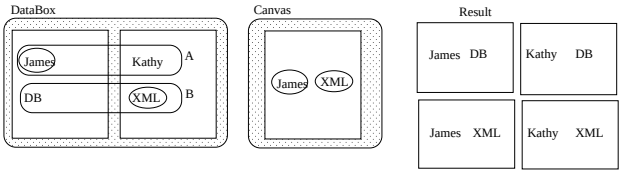

Figure 6. No S-Association

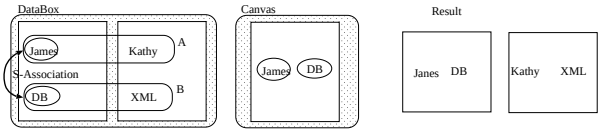

Figure 7. S-Association between A and B

have S-Association, and so do target sets C and D.) However, if he specifies 'Patrick' as an example for target set C, a V-Association does not occur and the system computes Cartesian product. The result would be { (James, Rm102, CG), (James, Rm102, DB), (Patrick, Rm205, CG), (Patrick, Rm205, DB)}.

3.3. Examples Given by Users

The scenario in Section 2 is an example of *restructuring* XML documents. There are other scenarios where the main purpose is *querying*. In querying scenarios, finding 'actual' examples in the DataBox is often difficult, because the number of the data objects the user wants may be very small. Our system provides a mechanism to support those kinds of scenarios, which allows the user to show the system 'artificial' (imaginary) examples, as in QBE. Figure 9 gives an example. When the user chooses a menu item (omitted in the figure), the DataBox presents a blank area. The user can then describe an artificial example and drag and drop it onto the Canvas to show what the system is to do. The user can manipulate the artificial example as if it were an 'existing' example. For example, he can perform "Clue" and "Select" operations on the value "215" in Figure 9 and drag and drop the value "Peggy" into the Canvas. The system will select the person whose room no. is 215. Note that the name does not have to be "Peggy," because the user did not declare any selection condition on that value.

4. Interaction with the System in the Example Scenario

Figure 10 illustrates operations to get the required result in the example scenario given in Section 2. We assume here that DataBoxes D1 and D2 contain all the publication information pages and the researcher profile pages, respectively. The operation sequence is as follows:

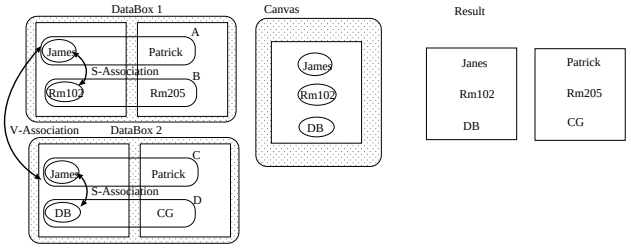

Figure 8. V-Association between A and C

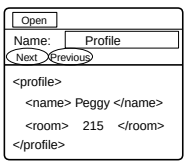

Figure 9. 'Artificial' example given by the user

(1) Designate '<name>James</name>' in D1 as an example. The default target set includes author names which appear first in publication list pages having the structure shown in Figure 2(a). Next, specify that each of '<name>Kathy</name>' and '<name>Patrick</name>' in D1 is an 'Another' object. Then, press the Next button of the DataBox D1 to find the publication page of year 2000, and specify that '<name>Vincent</name>' on the page is an 'Another' object. (Alternatively, you can use another page which has the structure shown Figure 2(b).) The system uses rules to generalize the relationship between positions of '<name>James</name>,' '<name>Kathy</name>,' '<name>Patrick</name>' and '<name> Vincent </name>,' so that the target set of this example is extended to include all author names.

(2) D&D the "<name>James</name>" from D1 onto the Canvas.

(3) Designate the '<name>James</name>' object in D2 as an example. Note that the two target sets of '<name>James</name>' objects in D1 and D2 have V-Association. Therefore, this specifies an equi-join between their target sets.

(4) Then designate the <interests> element in D2 as the 'Clue' example for the example '<name>James</name>' in D2.

(5) Perform 'Select' operation on the 'Clue' example and enter the condition "contains 'DB' " so that the target set of the example '<name>James</name>' in D2 includes only the names of researchers with their interests including "DB".

(6) Designate the '<pub>' element in D1 as an example. D&D it onto the Canvas.

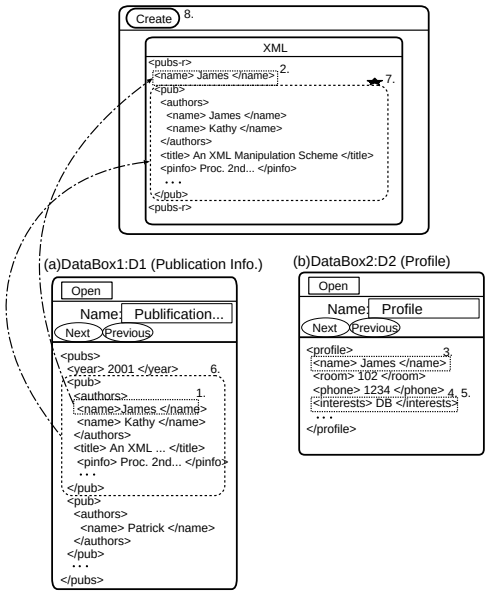

Figure 10. Manipulation for the scenario

(7) Put the repetition mark (*) on the dropped <pub> object. As a result, all of his publications are listed in this page. Otherwise, a new page is produced for each publication. Essentially, the repetition mark works as Nest operator of the nested relational algebra [7]. Subsection 5.4 contains details.

(8) Press the 'Create' button on the Canvas.

5. Semantics

This section defines the formal semantics of our framework in terms of the predicate logic and the nested relational algebra [7]. The major refinement compared to the preliminary one given in [8] is twofold:

(a) It treats example objects and 'Clue' examples in a unified way.

(b) It distinguishes between element contents and XML elements.

Refinement (a) gives a simpler and more general scheme. For example, operation of selecting objects in a target set is orthogonal to specifying a 'Clue' example: A 'Clue' example is now a kind of example object and has no different formalization. Also, the size of a target set can be reduced directly through explicit selection conditions on its example object. In addition to the refinements, artificial examples are permitted.

We define the formal semantics in the following three steps: (1) We express the source XML data as an object tree. (2) We derive the *target relation* from the user's example operations. The target relation represents target sets

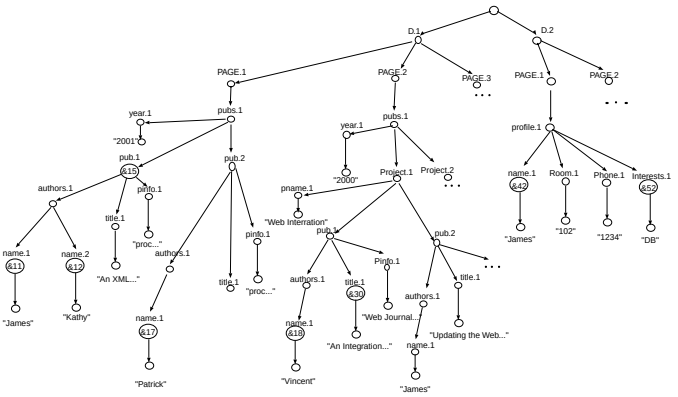

Figure 11. Tree object model of the XML data

and their associations. (3) We restructure the relation into a nested structure that reflects the grouping specified by repetition marks on the Canvas.

5.1. Data Modeling

We represent the source XML data as an object tree. The tree in Figure 11 represents a part of the source data in the example scenario. Every node (object) is annotated with a *label*, which consists of a *label name* and a *label number*. The second level nodes (children of the root of the whole tree) correspond to DataBoxes. The third level nodes, labeled with 'PAGE.i,' correspond to pages. Label numbers are sequentially assigned to sibling nodes with the same label name. Labels for leaves (T.1) is omitted in the figure. Every object has an OID. Several OIDs are explicitly presented in the form of $\&n$ for explanatory purposes. Note that this is *semistructured data*. There are two kinds of publication pages whose structure are different from each other: A '<pub>' element may be a direct child of a '<pubs>' element or placed under a '<project>' element.

Artificial examples given by users are attached as subtrees to the tree structure. A "PAGE.i" object is added to each subtree's root and the subtree (whose root is now the "PAGE.i" object) is attached to a DataBox (i.e. "D.i" object) where the user wrote the artificial example.

In the following discussion, <n> and <i> are abbreviations for <name> and <interests>, respectively. Also, $path(o)$ and $value(o)$ denote the path from the root to the object o, and the value of o, respectively. For example, $path(\&11)$ =D.1→PAGE.1→pubs.1 →pub.1 →authors.1→name.1, and $value(\&11)$ = "<n>James </n>."

5.2. Target Sets

Each example object has a corresponding target set. Given an example e, its *target set* (denoted by TS_e) is defined as follows:

$$TS_e = \{o | o \in O \wedge C\text{-}Pred_e(o)\},$$

Wildcard	What to match with
Name.?	A node with label name *Name*
?	A node with any label
?*	Any sequence of any nodes (the length can be 0)

Figure 12. Wildcards

where O is the set of all the objects in the object tree, and $C\text{-}Pred_e(o)$ is a *candidate predicate* incorporating a path expression. A path expression is similar to a path but may contain wildcards. $C\text{-}Pred_e(o)$ holds if and only if $path(o)$ conforms to the path expression. $C\text{-}Pred_e(o)$ is determined by the 'Example' and 'Another' operations as shown below.

Example

The following $TS_{\&11}$ gives the target set specified by Operation (1) in Section 4.

$$TS_{\&11} = \{o| \quad o \in O \wedge \text{D.1}\rightarrow\text{PAGE.?}\rightarrow\text{pubs.1}\rightarrow\text{?*}\rightarrow\text{pub.?} \rightarrow\text{authors.1}\rightarrow\text{name.?}[o]\}$$

Wildcards considered in the paper are listed in Figure 12. Given the source data shown in Figure 11, $TS_{\&11}$ = {"<n>James</n>", "<n>Kathy</n>", "<n>Patrick</n>", ..., "<n>Vincent</n>", ...} (all authors in the publication pages).

Derivation of the Candidate Predicate for an Example's Target Set

In the derivation process, we add *annotations* to predicates to give information on $path(e)$ and $value(e)$. Annotations are surrounded by "⟨" and "⟩". We represent the null sequence as ε.

For example, specification of $TS_{\&11}$ with annotations is as follows:

$$TS_{\&11} = \{o| \quad o \in O \wedge \text{D.1}\rightarrow\text{PAGE.?}\langle\text{PAGE.1}\rangle \rightarrow\text{pubs.1} \rightarrow\text{?*}\langle\varepsilon\rangle \rightarrow\text{pub.?}\langle\text{pub.1}\rangle \rightarrow\text{authors.1} \rightarrow\text{name.?}\langle\text{name.1}\rangle[o\langle\text{<n>James</n>}\rangle]\}$$

Note that the annotations give information on $path(\&11)$ and $value(\&11)$.

In general, $C\text{-}Pred_e(o)$ is derived as follows:

(1) First, when the user specifies that the object e is an example, the *default candidate predicate* $p[o\langle v\rangle]$ is derived. Here, p is the same as $path(e)$ except that its PAGE.i is replaced by PAGE.?⟨PAGE.i⟩, and v is $value(e)$. For example, consider Operation (1) in Section 4. When the user specifies the object &11 (with its value "<n>James</n>") as an example, the default candidate predicate derived is D.1→PAGE.?⟨PAGE.1⟩→ pubs.1→ pub.1 → anthors.1→name.1[$o\langle$<n>James</n>$\rangle$]. The predicate defines the set of objects appearing at the same position on different pages ($i.e.$ the default target set).

(2) An 'Another' operation modifies the candidate predicate to accept the 'Another' object. Modification rules shown in Figure 13 are used. Each rule prescribes how to modify the original predicate according to $path(e)$ and $path(a)$, where e and a are the example and an 'Another' object, respectively. In Figure 13, B and C denote label names, q_i denotes a partial path, and p_i denotes the partial path expression of the original predicate to which q_i conforms. q_i' is a partial path that p_i can accept. The basic idea behind the rules is to place a wildcard at the position where $path(e)$ and $path(a)$ conflict with each other.

For example, in Operation (1), the user specifies the object &12 (with its value "<n>Kathy</n>") as the first 'Another' object. Then, $path(a)$ = D.1→PAGE.1→pubs.1→pub.1→authors.1→name.2.
We can obtain $path(e)$ = D.1→PAGE.1→ pubs.1 → pub.1 → authors.1→name.1 from the annotated default candidate predicate. The system finds that the default candidate predicate cannot accept $path(a)$ because name.1 in the path expression conflicts with name.2. In this case, they conflict at their label numbers. Therefore, we can apply Rule 1, and get D.1 → PAGE.? ⟨PAGE.1⟩ → pubs.1 → pub.1 → authors.1 → name.?⟨name.1⟩ [$o\langle$<n>James</n>$\rangle$]. Next, the user specifies the objects &17 (with its value "<n>Patrick</n>") and &18 (with its value "<n>Vincent</n>") as the second and third 'Another' objects. In a similar way, we can apply Rule 1 and Rule 3 to the modified predicate. This results in the above $TS_{\&11}$.

Derivation of the Candidate Predicate for an Clue's Target Set

Operations (3) and (4) in Section 4 construct the following new target set for the 'Clue' example.

$$TS_{\&52} = \{o| \quad o \in O \wedge \text{D.2}\rightarrow\text{PAGE.?}\langle\text{PAGE.1}\rangle\rightarrow\text{profile.1} \rightarrow\text{interests.1}[c\langle\text{<i>DB</i>}\rangle]\}$$

Let $C\text{-}Pred_{cl}(c)$ be the candidate predicate of the 'Clue' example *cl* for an example object e. (We call *e the target example* here. In this case, &42 with its value '<n>James</n>' is the target example for the 'Clue' example &52 with its value 'DB'.) The system determines $C\text{-}Pred_{cl}(c)$ using $C\text{-}Pred_e(o)$, $path(e)$ and $path(cl)$ as follows:

(1) Let P be the predicate '$p[c\langle v\rangle]$' where p is $path(cl)$. For example, consider Operation (4) in Section 4. Then, P =D.2→PAGE.1→ profile.1→ interests.1[$c\langle$DB$\rangle$]. Note that P holds if and only if c is the 'Clue' example cl itself. Also, let P_e be the candidate predicate for the target example e. (That is, $C\text{-}Pred_e(o)$.) In Operation (4), P_e = D.2→PAGE.?→ profile.1→ name.1[$o\langle$<n>James</n>$\rangle$].

(2) Find the longest prefix partial path expression p_e of P_e such that $path(e)$ and $path(cl)$ share the same par-

	Original Pred.	$Path(e)$	$Path(a)$	Modified Pred.
Rule 1	$p_1\ B.i\ p_2[e\langle v\rangle]$	$q_1\ B.i\ q_2$	$q_1'\ B.k(\neq i)\ q_2'$	$p_1\ B.?\langle B.i\rangle\ p_2[e\langle v\rangle]$
Rule 2	$p_1\ B.i\ p_2[e\langle v\rangle]$	$q_1\ B.i\ q_2$	$q_1'\ C(\neq B).k\ q_2'$	$p_1\ ?\langle B.i\rangle\ p_2[e\langle v\rangle]$
Rule 3	$p_1\ q_3\ p_2[e\langle v\rangle]$	$q_1\ q_3\ q_2$	$q_1'\ q_4(\neq q_3)\ q_2'$	$p_1? * \langle q_3\rangle\ p_2[e\langle v\rangle]$

Figure 13. Rules to modify candidate predicates. (p_i, q_i and q_i' can be ε.)

tial path in the range of p_e. In Operation (4), p_e = D.2→PAGE.?→profile.1. Then, the $C\text{-}Pred_{cl}(c)$ is derived from P by replacing the corresponding prefix part of P with p_e. Therefore, $C\text{-}Pred_{cl}(c)$ =D.2→PAGE.?→ profile.1→ interests.1[$c\langle$DB$\rangle$]. In the above example, the predicate holds if and only if c is an (<interests>) element of some researcher on profile pages.

Select Operation on a Target Set

If 'Select' operation is performed on an example (or 'Clue' example) object, a selection condition predicate is inserted into the target set definition. For example, Operation (5) modifies the target set of the 'Clue' example (DB) as follows:

$TS_{\&52} = \{o|\quad o \in O \wedge$ D.2→PAGE.?$\langle$PAGE.1$\rangle$→profile.1 →interests.1[$c\langle$DB$\rangle$]$\wedge c$ contains 'DB'$\}$

5.3. Target Relation

A target relation represents the target sets of examples and the associations among them.

Definition

Assume that there are n target sets (specified by examples $e_1, \ldots e_n$, including Clue examples), and l associations among them. Then, the target relation is defined as follows.

$$\begin{aligned} TR = \{\quad & (value(o_1), \ldots, value(o_n)) \ \mid \\ & o_1 \in O \wedge C\text{-}Pred_{e_1}(o_1) \wedge S\text{-}Pred_{e_1}(o_1) \\ & \vdots \\ & \wedge o_n \in O \wedge C\text{-}Pred_{e_n}(o_n) \wedge S\text{-}Pred_{e_n}(o_n) \\ & \wedge A\text{-}Pred_1(o_{a_1}, o_{b_1}) \wedge \ldots \wedge A\text{-}Pred_l(o_{a_l}, o_{b_l})\}, \end{aligned}$$

where $A\text{-}Pred_i$ is an S-Association predicate or V-Association predicate, and $S\text{-}Pred_{e_i}$ corresponds to a selection condition specified in 'Select' operation (it is '*True*' when no 'Select' operation is done for the target set). S-Association predicate has the form $Share_p(o, o')$. It holds if and only if $path(o)$ and $path(o')$ share the same partial path which starts from the root and conforms to the path expression p. Figure 14 illustrates the meaning of $Share_{D.2\rightarrow PAGE.?\rightarrow profile.1}(o_i, o_j)$. V-Association predicate has the form $o \stackrel{v}{=} o'$, and holds if and only if $value(o) = value(o')$.

The scheme to derive V-Association and S-Association predicates is the same as the preliminary version in [8]. We explain the basic idea here. V-Association predicates are derived in a straightforward way: They are determined by the values of example objects. S-Association predicates are determined by paths of example objects and their candidate predicates: If two example objects e and e' share the same prefix path, and the corresponding path expressions (including wild cards) are the same, then the system infers that the user wants to associate an object o in one target set with o' in the other target set when they share the partial path in the same way as e and e'.

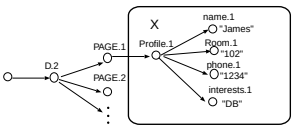

Figure 14. An object set X such that $(\forall o_i \forall o_j \in X) Share_{D.2\rightarrow PAGE.?\rightarrow profile.1}(o_i, o_j)$.

Example

The target relation for the example scenario is shown in Expression A. A superscript number indicates an operation number in Section 4 to which the predicate corresponds.

$\{(value(o_1), value(o_2), value(o_3), value(c_1))|$
$o_1 \in O \wedge p_1$→authors.1→name.?$\langle$name.1$\rangle$[$o_1\langle$<n>James</n>$\rangle$]$^{(1)}$
$\wedge o_2 \in O \wedge p_1$[$o_2\langle$<pub>...An XML...</pub>$\rangle$]$^{(6)}$
$\wedge o_3 \in O \wedge p_2$→name.1[$o_5\langle$<n>James</n>$\rangle$]$^{(3)}$
$\wedge c_1 \in O \wedge p_2$→interests.1[$c_1$]$^{(4)} \wedge c_1$ contains "DB"$^{(5)}$
$\wedge Share_{p_2}(o_3, c_1)^{(4)} \wedge Share_{p_1}(o_1, o_2)^{(6)} \wedge o_1 \stackrel{v}{=} o_3^{(3)}\}$

where

p_1 = D.1→PAGE.?$\langle$PAGE.1$\rangle$→pubs.1→?*$\langle\varepsilon\rangle$→pub.?$\langle$pub.1$\rangle$, and
p_2 = D.2→PAGE.?$\langle$PAGE.1$\rangle$→profile.1.

Expression A. Specification of the target relation

Figure 15 shows all the target sets and associations involved in the example scenario. Figure 16 shows the target relation based on the target sets and associations.

5.4. Creating the Nested Structure

Creation of the nested structure reflecting the specified grouping is straightforward. It depends on the position of examples and repetition marks (*) on the Canvas. Figure 17

Ex.<n>James</n> in D1	Ex. <pub>..An XML..</pub> in D1	Ex.<n>James</n> in D2	Ex. "DB" in D2
<n>James</n>	<pub>..An XML..</pub>	<n>James</n>	DB
<n>James</n>	<pub>..Updating..</pub>	<n>James</n>	DB
<n>Vincent</n>	<pub>..An Integration..</pub>	<n>Vincent</n>	Automata, DB

Figure 16. Target relation

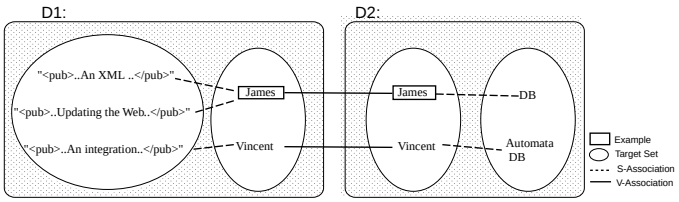

Figure 15. Target sets and associations

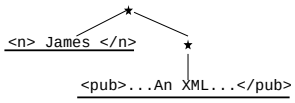

Figure 17. Grouping specification

shows the grouping structure specified on the Canvas shown in Figure 10. The nested relation is constructed by applying Projection and Nest operators [7]. In this example, the following expression produces the nested relation shown in Figure 18.

$$\nu_{V=(\underline{\texttt{<pub>..AnXML..</pub>}})}(\pi_{\underline{\texttt{<n>James</n>}},\underline{\texttt{<pub>..AnXML..</pub>}}}(TR))$$

The system outputs the result in the form of XML documents (Figure 1(c)).

6. Experiments

We conducted preliminary experiments to examine the usability of our framework; we used our prototype system named AQUA [9] (Figure 19) for the experiments. The system is implemented in Java (Java2 SDK1.3) and utilizes its drag and drop API. The code is about 21,000 lines. XML-QL and XML-GL were used for comparison. Relationships are given in Figure 20.

6.1. Method

We recruited 18 people and divided them into three groups according to their skills and database-related knowledge. Members of Group A had no knowledge of concepts related to databases. People in Group B had knowledge of relational databases and SQL. People in Group C had knowledge of XML-QL in addition to database concepts.

We asked them to compose queries using XML-QL, XML-GL and AQUA, and observed their specification process and results. In the experiment, we used the AQUA prototype system and the XML-QL prototype implementation

Ex.<n>James</n> in D1	V
	Ex. <pub>..An XML..</pub> in D1

<n>James</n>	<pub>..An XML..</pub>
	<pub>..Updating..</pub>
<n>Vincent</n>	<pub>..An Integration..</pub>

Figure 18. Result nested relation

Figure 19. AQUA prototype system

obtained from the AT&T-Labs web-site[16]. We did not have an operational XML-GL system when we conducted the experiment. Therefore, we constructed an XML-GL environment by preparing XML-GL query components in a commercial drawing tool so that users could drag and drop them to specify queries. To make a fair comparison possible, we did not allow users to execute queries. For each framework, we handed them a manual of the framework and gave them a tutorial for 20 minutes. We then asked them to compose queries to manipulate XML documents in the framework.

The set of XML documents to be manipulated were real XML documents (selected from XML documents of publications information in DBLP[14] and SIGMOD Record[15]). The set of documents forms semistructured data, because they have the same tags in part but differ in their DTD structure. For each framework, we asked them to give queries to meet the following requirements:

Q1: Construct a set of XML documents, each of which has the title of a publication.

Q2: Construct a set of XML documents, each of which has a combination of the title of a SIGMOD Record paper and

	CUI	GUI
Explicit specification in query languages	XML-QL	XML-GL
Implicit specification through examples		Our Framework (AQUA)

Figure 20. Relationship among frameworks

	Group A	Group B	Group C	Average
AQUA	72s (6/6)	36s (5/6)	40s (6/6)	49s (17/18)
XML-GL	74s (5/6)	61s (6/6)	67s (6/6)	67s (17/18)
XML-QL	200s (5/6)	74s (6/6)	113s (6/6)	129s (17/18)

Table 1. Result of the experiment (Q1)

	Group A	Group B	Group C	Average
AQUA	87s (5/6)	55s (6/6)	55s (6/6)	66s (17/18)
XML-GL	197s (5/6)	141s (6/6)	183s (6/6)	173s (17/18)
XML-QL	343s (3/6)	175s (4/6)	176s (6/6)	231s (13/18)

Table 2. Result of the experiment (Q2)

one of its authors.
Q3: Construct an XML document that has a list of titles of publications grouped by author.

In the experiment, we changed the order of the three frameworks for each person so that the order would not affect the results as a whole. We measured the time required to compose queries and checked the percentage of correct queries. Participants were allowed to refer to the tutorial manuals during the test. After the test, we asked them to answer questionnaires to check usability of each framework.

6.2. Results and Discussions

(1) Time: Tables 1 to 3 list the average query composition time (in seconds) and the proportion of correct answers (the number of correct answers/people) in Q1, Q2 and Q3. For Q1 and Q2, the order in terms of time is AQUA<XML-GL<XML-QL (Figure 21). However, for Q3, the time with XML-QL is shorter than that for XML-GL. The reason is that some of the participants were familiar with XML-QL. (They are in Group C.)

We performed an analysis of variance and a Tukey's multiple comparison test using the type of specification framework as its factor to see if there is a significant difference among the three frameworks. As a result, we found that the difference between our framework and XML-GL (or XML-QL) was statistically significant at the level of 1%. This means that the time required to compose queries with our framework is clearly shorter compared to other frameworks.
(2) Percentage of correct answers: The tables show that, regarding the percentage of correct answers, the order is AQUA>XML-QL>XML-GL. But there is only a small difference and we did not find any statistical significance in the experiment. The reason for the high percentages in general is that the requirements in this experiment were not so difficult. We did not count mistakes related to misspelling in XML-QL and XML-GL.
(3) Usability: We asked participants to give usability scores between -5(the most difficult) and +5(the easiest) to XML-GL and our framework, assuming that the usability score of XML-QL is 0. The order of the frameworks based on the score is AQUA>XML-GL>XML-QL (Figure 22).

We performed an analysis similar to the one we did with the query composition time. The specification framework and the group were considered as factors in the analysis. As a result, we found that the difference between our framework and XML-GL (or XML-QL) was statistically significant at the level of 1%. This means that our framework was clearly easier than XML-GL and XML-QL in the experiment. On the other hand, there was no significant difference between different groups. This shows our framework was evaluated highly regardless of participants' knowledge on databases and XML-QL.

Possible reasons for the AQUA's high usability include (a) Manipulation of actual data instances and the system's direct reaction for the manipulation enable users to see what they are doing, and (b) The users do not have to learn unfamiliar concepts, such as regular path expressions. Basically, AQUA's superiority over the others can be discussed from those two perspectives. In other words, AQUA has advantages in terms of (a) the interactive support given by the implemented environment, and (b) the conceptual simplicity. It is possible, for example, that XML-QL's usability score would be higher if we provide users with a tool to help write XML-QL queries, instead of allowing them to use only a text editor. However, we would not expect the other frameworks to get a higher score than AQUA, because AQUA holds the advantage of simplicity. Here, we must note that AQUA is designed for *naive end-users*, and that we assume in the discussion that they are main users. Therefore, the discussion would change if they were *experienced users*, who would prefer to use a small number of 'abstract concepts' rather than show a large number of examples.

7. Conclusion

In this paper, we have explained a graphical manipulation framework of XML documents. It is unique in that users implicitly specify XML manipulation through example operations of data existing in the database, while other approaches require them to code explicit specifications in textual and graphical languages. The results of our prelim-

	Group A	Group B	Group C	Average
AQUA	62s (5/6)	46s (6/6)	49s (6/6)	52s (17/18)
XML-GL	245s (5/6)	238s (5/6)	205s (5/6)	229s (15/18)
XML-QL	323s (6/6)	160s (5/6)	146s (6/6)	210s (17/18)

Table 3. Result of the experiment (Q3)

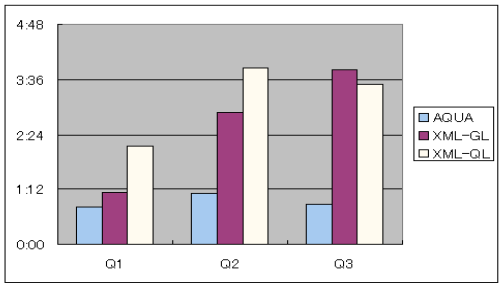

Figure 21. Average time

inary experiments show that our framework is promising and has high usability even for naive end-users having no knowledge of concepts related to databases and query languages.

Given the results showing it is a promising approach, our current interest is expressive power analysis. Particularly, we are interested in elaborating the algorithm so that we can show some completeness. One direction we are now taking is to take advantage of results in computational learning theory[2]. Another interesting task is to extend the framework to accept a DTD and produce a template on the Canvas. The template will help users to decide where objects are to be dropped. We also plan to implement translators that output queries in popular XML query languages instead of queries for our proprietary XML query engine.

Acknowledgements

The authors are grateful to many people who have contributed to the development of AQUA and its experimental evaluation. Special thanks are due to Mr. Seiichi Koizumi and Mr. Satoshi Takano. They also thank Nipppon Television Network Corporation for providing part of sample data. This work has been supported in part by the Grant-in-aid for Scientific Research from Japan Society for the Promotion of Science.

References

[1] S. Abiteboul. Querying semi-structured data. *Proc. 6th International Conference on Data Theory (ICDT'97)*, pp. 1-18, 1997.

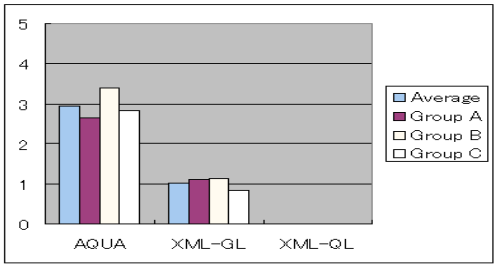

Figure 22. Usability scores

[2] D. Angluin. Computational Learning Theory: Survey and Selected Bibliography. *Proc. 24th Annual ACM Symposium on Theory of Computing*, pp. 351-369, 1992.

[3] Peter Buneman. Semistructured data. *Proc. 16th ACM Symposium on Principles of Database Systems (PODS'97)*, pp. 117-121, 1998.

[4] S. Ceri, S. Comai, E. Damiani, P. Fraternali, S. Paraboschi, and L. Tanca. XML-GL: a Graphical Query Language for Querying and Restructuring XML Documents. *8th International World Wide Web Conference (WWW8)* 1999.

[5] Don Chamberlin, Jonathan Robie, and Daniela Florescu. Quilt: an XML Query Language for Heterogeneous Data Sources. *Proc. WebDB'00*, 2000.

[6] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. XML-QL: A query language for XML. *Proc. QL'98 - The Query Languages Workshop*, http://www.w3.org/TR/1998/NOTE-xml-ql-19980819/.

[7] P. C. Fischer and S. J. Thomas. Operators for non-first-normal-form relations. *Proc. IEEE COMPSAC83*, pp. 464-475, 1983.

[8] A. Morishima, S. Koizumi and H. Kitagawa. Drag and Drop: Amalgamation of Authoring, Querying, and Restructuring for Multimedia View Construction. *Proc. 5th IFIP 2.6 Working Conference on Visual Database Systems(VDB5)*, pp. 257-276, 2000.

[9] A. Morishima, S. Koizumi, H. Kitagawa, and S. Takano. Enabling End-users to Construct Data-intensive Web-sites from XML Repositories: An Example-based Approach. *Proc. 27th VLDB Conf.*, pp. 703-704, 2001.

[10] J. Robie, J. Lapp, D. Schach. XML Query Language (XQL). http://www.w3.org/TandS/QL/QL98/.

[11] W3C. XML Transformations (XSLT). http://www.w3.org/TR/xslt.

[12] W3C. XQuery 1.0: An XML Query Language. X3C Working Draft, http://www.w3.org/TR/, 2001.

[13] M. M. Zloof. Query-by-Example: a Data Base Language. *IBM Systems Journal*, Vol. 16, No. 4, pp. 324-343, 1977.

[14] DBLP XML Records. http://www.informatik.uni-trier.de/ ley/db/about/dallas-usage.html

[15] ACM SIGMOD Record: XML Version. http://www.acm.org/sigmod/record/xml

[16] XML-QL: A Query Language for XML. User's Guide. http://www.research.att.com/ mff/xmlql/doc/