International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:2, No:3, 2008

A formatting method for transforming XML data into HTML

Zhe JIN, and Motomichi TOYAMA, *Member, IEEE*

Abstract—In this paper, we propose a fixed formatting method of PPX(Pretty Printer for XML). PPX is a query language for XML database which has extensive formatting capability that produces HTML as the result of a query. The fixed formatting method is to completely specify the combination of variables and layout specification operators within the layout expression of the GENERATE clause of PPX. In the experiment, a quick comparison shows that PPX requires far less description compared to XSLT or XQuery programs doing the same tasks.

Keywords—PPX, XML, HTML, XSLT, XQuery, Fixed formatting method.

I. Introduction

A Few existing languages transforming XML data into HTML includes XQuery [1], XSLT 1.0 [2], XSLT 2.0 [3], JAVA and C++, etc. However, it is not easy for ordinary users to do programming [4], [5], [6].

In this paper, we propose a query language, called PPX, which uses formatting methods to transform XML data into HTML. This research aims to focus on the design of the layout without considering the data structure of XML directly, so that the layout work of the XML data can be done easily.

The PPX query language, which has fixed formatting method and automatic formatting method for XML database, has extensive formatting capability that produces HTML as the result of a query. In this paper, we only discuss a fixed formatting method. The following PPX 1 shows that XML instance in figure 2 is layouted into HTML by fixed formatting method in the layout expression [1] of the GENERATE clause of PPX.

```
PPX 1:
GENERATE html
 [ $i/title ! [ $j/univ , [ $j/name ]!
 ]! ]!
FOR $i in db('paper.xml')/papers/paper,
FOR $j in $i/authors/author
```

This query converts a flat list structure of searched XML data into the nest structure of XML data by combining the variables and the layout specification operators in the layout expression, and generates HTML. The results are shown in figure 1.

Zhe JIN is with the Department of Information and Computer Science, Keio University, Hiyoshi 3–14–1, Kohoku-ku Yokohama, 223-8522, Japan (e-mail: tetsu@db.ics.keio.ac.jp).

Motomichi TOYAMA is with the Department of Information and Computer Science, Keio University, Hiyoshi 3–14–1, Kohoku-ku Yokohama, 223-8522, Japan (e-mail: toyama@ics.keio.ac.jp).

[1]discussed at III.A section.

Indexing XML Data for Efficient XML Query Pattern Matching	
York University	James Clifford
	Laurent Aalto
Path Selectivity Estimation for XML Data	
York University	Antonio Badia
	Georgia Koutrika
	David Maier
Boston University	Anand Rajaraman
	Laurent Vieille
The University of Hannover	Francois Bancilhon
University of Washington	Birgit Aalderink
Query Processing in XML Databases	
York University	David Maier
Boston University	Laurent Vieille
Efficient Processing of XML Path Queries	
York University	James Clifford
Boston University	Laurent Vieille

Fig. 1. Format results by PPX 1

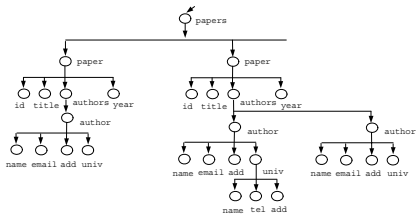

Fig. 2. An XML instance

The rest of this paper is organized as follows. In Section II, we discuss basic concepts. In Section III and Section IV, we present PPX query language and query processing. In Section V, we describe the implementation of the fixed formatting method of the PPX. In Section VI, the related work is mention. Lastly, Section VII summarizes the contributions of this paper.

II. Basic Concepts

This section introduces the path expressions [7], path expression sets and Format object of the PPX query language.

A. Path Expressions

The path expressions include absolute path expression and relative path expression. If they need not be distinguished, they will be abbreviated as the path expression. In this research, the path expression is divided into a complete path expression and an incomplete path expression.

(1) The complete path expression specifies the path expression from the root node to the target text node. For example, the following path expression shows a complete one.

Example 1: /papers/paper/title/text()

The complete path expression is connected with the path expression used in the first FOR clause of PPX 1 and the relative path expression used in the variable. The complete path expression searches for the XML data, which are the value of the title element node. The text node is omitted when it is specified in the layout expression of the GENERATE clause of PPX. The XML ata searched by the complete path expression sets is the object of a fixed format.

(2) The incomplete path expression specifies the path expression from the root node to the target element node. For example, the following path expression shows an incomplete one.

Example 2: /papers/paper

The incomplete path expression used in the first FOR clause of PPX 1 searches for the XML data, which are the value of the element nodes included in a part of XML [2] that is under the paper element nodes. The XML data searched by the incomplete path expression sets is the object of an automatic format.

B. Path expression sets

The path expression sets are comprised by the query path expression set P(Q) and the XML path expression set P(X). Each of set includes the complete path expression and the incomplete path expression.

P(Q): This is a group of the path expressions existing in the PPX query.

P(X): This is a group of the path expressions existing in the layout object part of XML.

C. Format object

The following explaination is about the format object, which is different from the relationship between two kinds of path expression sets. The format objet includes the object of fixed format and the object of automatic format.

The object of a fixed format is defined as follows.

Definition 1 (The object of fixed format) The group of path expression set in the intersection of XML path expression set P(X) and complete query path expression set P(Qc)

$$P(Xc) = P(X) \cap P(Qc)$$

is a complete XML path expression set P(Xc), and the value of these text nodes becomes the object of a fixed format.

The object of an automatic format is defined as follows.

Definition 2 (The object of automatic format) The group of the path expression set in the part where complete query path expression set P(Qc) is excluded from XML path expression set P(X)

$$P(Xi) = P(X) - P(Qc)$$

is an incomplete XML path expression set P(Xi), and the value of the text nodes included in a part of XML becomes the object of an automatic format.

[2]A part of XML between start tag and end tag of all the element nodes in XML

III. PPX

The basic structure of PPX query language consists of GENERATE, FOR, and WHERE clauses etc.

GENERATE $< media >< LayoutExpression >$

FOR $<$ "$" $+ VariableName \ in \ PathExpression >$

WHERE $< condition >$

In the GENERATE clause, the output media (HTML, XML, etc.) and the layout expression are specified. By the layout expression, the output of the media with all kinds of structures can be realized. In this paper, we only discuss the HTML output medium. Due to the usage similarity of FOR, WHERE clauses etc. in both PPX and XQuery, the explaination of them is omitted, only the GENERATE clause is explained here.

A. Layout Expressions

In the layout expressions can be specified a fixed formatting method that is combination of variables and layout specification operators.

1) Variables: The variables represent the searched XML data obtained by path expressions. They consist of the variable name and the relative path expression. It is shown as follows.

Variable ::= "$"+VariableName/Relative PathExpression

In case of a completely specified format, we use a complete path expression in which the relative path expression specified in the variable is connected with the path expression specified in the FOR clause.

2) Layout Specification Operators: The layout specification operators are extension of operators of SuperSQL [8]. They comprise omissible operators and existing connect operators, repeat operators and decorative operators.

(1) Connect Operators

There are horizontal (,), vertical (!) and depth (%) operators which connect the objects generated as their operands horizontally, vertically and in the depth direction, respectively. In case of generating HTML, the depth connect operator specifies the hyper link in a hypertext (figure 3).

(2) Repeat Operators

There are horizontal([ ],), vertical([ ]!), and depth([ ]%) operators. In a pair of brackets, the layout expression is specified. The multiple instances generated by the inner layout expression are connected repeatedly in each direction. When a subexpression of repeat operator is connected to one or more primary items, the latter are used to group repeating items. In this way, redundant display of grouping items can be suppressed (figure 4).

(3) Decorative Operators

Decorative operators are supported to designate decorative features of outputs, such as font size, table border, width, image directory, in the form of @ follows a layout expression, and decorative expressions e_j are in a pair of braces({ }), which are separated by comma. Eache_j is $item_j = value_j$, $j > 0$. A decorative operator d_i is described as below.

$d_i =$<$LayoutExpression$>@$\{e_1, e_2, ..., e_n\}$

(4) Omissible Operators

Omissible operators include the &- operator and the &+ operator. The &- operator expresses XList form without tags and the &+ operator expresses XList form with tags, for

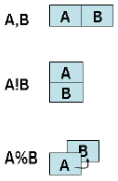

Fig. 3. Connct operators

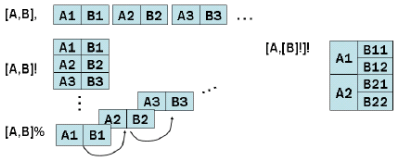

Fig. 4. Repeat operators and grouping

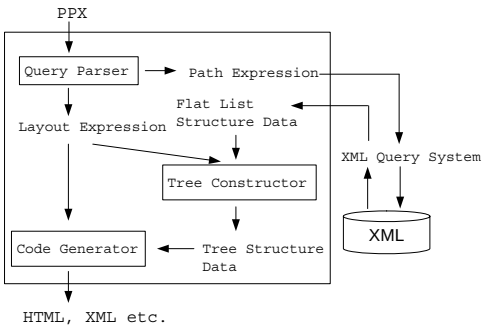

Fig. 5. System architecture

the searched XML data included in a part of XML by an incomplete path expression.

B. System architecture

This system consists of the query parser, the list constructor and the code generator as shown in Figure 5. The PPX query is divided into the layout expression and the path expression in the query parser. The path expression searches for XML data. The searched XML data of flat list structure is reconstructed by layout expression in the tree constructor. Finally, the reconstructed XML data is transformed into HTML with a variety of table structures in the code generator.

C. Overview of fixed formatting method

The XML data format which transform XML data into HTML with the formatting method is divided into two steps as shown in Figure 6.

1) Step 1: Reconstructing XML Data: Here, the XML data with the searched flat list structure is reconstructed. That is, the structure is converted into a tree structure different from the structure of original XML according to specifications of variable location change, variable addition, or grouping of element node by layout specification operators etc. in the layout expression as shown in Figure 6(2).

2) Step 2: Tagging: Here, the reconstructed XML data is tagged and transformed into HTML. In addition, the HTML is decorated according to the decorative operators when it is produced, as shown in Figure 6(3).

IV. Query Processing

In this section, we will have a brief discussion on how the formatting method of PPX queries layout XML data into HTML.

University of Washington		
Birgit Aalderink	2006	A Query Language for Semantic Web
		Path Selectivity Estimation for XML Data
Oxford University		
Ann Aalto	2006	A Query Language for Semantic Web
Harvard University		
Francois Bancilhon	2006	A Query Language for Semantic Web
Stanford University		
Samuli Aaltonen	2006	A Query Language for Semantic Web
Boston University		
Anand Rajaraman	2006	Path Selectivity Estimation for XML Data
Laurent Vieille	2006	Path Selectivity Estimation for XML Data
		Query Processing in XML Databases
	2005	Efficient Processing of XML Path Queries
		Dynamically Updating XML Documents
The University of Hannover		
Francois Bancilhon	2006	Path Selectivity Estimation for XML Data

Fig. 7. Format results by PPX 2

A. Structure convert and layout

For example, the following PPX 2 specifies the formatting method that layout XML data into HTML based on the layout expression of PPX 1.

```
PPX 2:
GENERATE html
 [ $j/univ ! [ $j/name , [ $i/year ,
  [ $i/title ]!
 ]! ]! ]!
FOR $i in db('paper.xml')/papers/paper,
FOR $j in $i/authors/author
```

In this query, the name is grouped with the univ, the year is grouped with the name and the title is grouped with the year. The results are shown in figure 7, in which the structure is different from the structure in figure 1.

These PPX examples show that the extracted XML data structure is converted and layouted according to the variable location changes, variable addition, grouping of element node using layout specification operators in the layout expression.

Moreover, the following PPX 3 specifies the formatting method that using the % operators based on the layout expression of PPX 2.

```
PPX 3:
GENERATE html
 [ $j/univ % [ $j/name , [ $i/year ,
```

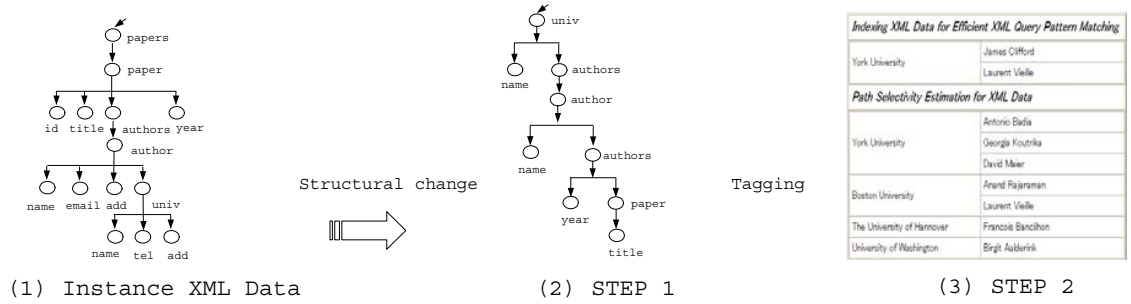

Fig. 6. Overview of fixed formatting method

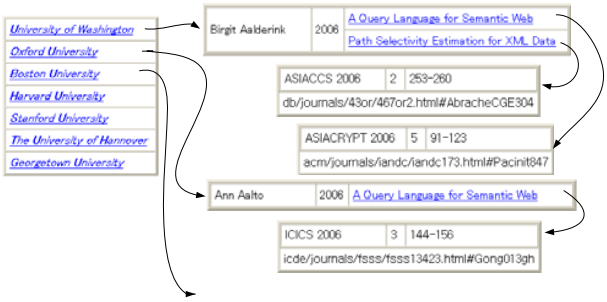

Fig. 8. Format results by PPX 3

```
    [ $i/title %
     [ $n/conf , $n/volume , $n/pages !
       $n/pdf ],
 ]! ]! ]! ]!
FOR $i in db('paper.xml')/papers/paper,
FOR $j in $i/authors/author,
FOR $n in db('paper.xml')/papers/paper
WHERE $n/title = $i/title
```

PPX 3 shows that the variable $j/univ and the variable $i/title which are on the left of depth connect operator(%) become two anchors, and unite with the subpage groups generated on the right side, by the hyperlink. The results are shown in figure 8.

In addition, the XML data can be decorated when HTML is generated by specifying the decoration operators. For example, the following PPX 4 generates decorated HTML as shown in figure 9.

```
PPX 4:
GENERATE html
 [ $j/univ@{font-style=oblique, color=red} !
  [ $j/name@{color=blue} ,
   [ $i/year@{color=blue} !
    [ $i/title@{color=green} ]!
 ]! ]! ]!
FOR $i in db('paper.xml')/papers/paper,
FOR $j in $i/authors/author
```

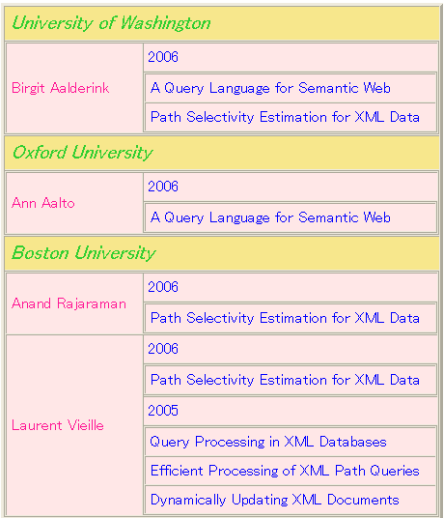

Fig. 9. Format result by PPX 4

B. Express by XList form

Besides, the following PPX 5 specifies the formatting method using the &- operator based on the layout expression of PPX 1.

```
PPX 5:
GENERATE html
 [ $i/title !
  [ $i/year , &-( $i/authors ) ]!
 ]!
FOR $i in db('paper.xml')/papers/paper
```

The XML data included in a part of XML is searched by an incomplete path expression, whose relative path expression specified in the variable was connected with the path expression specified in the FOR clause, and express XList form that without tags in the HTML as shown in figure 10. Moreover, the searched XML data can express XList form with tags by changing the &- operator to the &+ operator.

V. Experimental Evaluation

In this section, we have implemented the proposed fixed formatting method and comparison of the proposed PPX

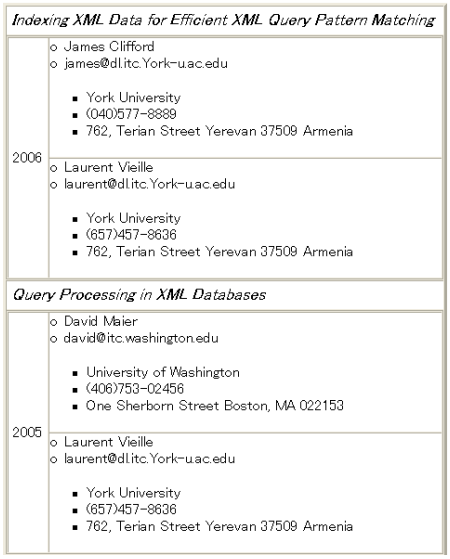

Fig. 10. Format results by PPX 6

and existing XQuery, XSLT 1.0, XSLT 2.0, etc. is shown, concerning the description amount and the effectiveness of transformation abilities. Moreover, discuss the problem of this fixed formatting method encountered when experimenting which should be resolved.

A. Experimental Environment

We implemented the fixed formatting method using Java and used XML data (table I) of UW XML repository [9] to generate HTML. The DB2 Version 9 is used for PPX and XQuery. The XMLSpy [10] is used for XSLT 1.0 and XSLT 2.0.

TABLE I
THE TEST DATA DETAILS

file name	element	max-depth	avg-depth	size
psd7003.xml	21305818	7	5.15147	683MB
dblp.xml	3332130	6	2.90228	127MB
sigmod.xml	11526	6	5.14107	467MB
treebank_e.xml	2437666	36	7.87279	82MB

B. Comparison of description amount

PPX transforms XML data into HTML with a small description amount as shown in the previous section. On the other hand, XQuer, XSLT 1.0, XSLT 2.0 do the same transformation as PPX but need large description amount.

1) XQuery: For example, the following XQuery does the same transformation as PPX 1. This XQuery describes HTML tags directly in the query sentence to transforming XML data into HTML with the same table structure in figure 1 in section I.

```
XQuery:
FOR $i in
db2-fn:xmlcolumn('paper.xml')//paper
RETURN
<table border="1">
<tr>
<td>{$i/title/text()}</td>
</tr>
<tr>{
FOR $j in $i//authors
RETURN
<td>{
FOR $l in distinct-values ($j//univ)
RETURN
<table border="1"><tr>
<td>{$l}</td>
<td>
<table border="1">{
FOR $k in $j/author[univ = $l]
RETURN
<tr>
<td>{$k/name/text()}</td>
</tr>
}</table>
</td>
</tr></table>
}</td>
}</tr>
</table>;
```

In this query, larger amount of description is required than with PPX and the initial query can not be recycled, it needs to be rewritten.

2) XSLT: For example, the following XSLT 2.0 does the same transformation as PPX 1. This XSLT describes HTML tags directly in the style sheet sentence to transforming XML data into HTML with the same table structure in figure 1 in section I.

```
XSLT 2.0:
<xsl:stylesheet version="2.0">
<xsl:output method="html"
encoding="Shift_JIS"/>
<xsl:template match="/">
<html><table border="1">
<xsl:apply-templates select="*"/>
</table></html>
</xsl:template>
<xsl:template match="papers">
<xsl:for-each-group
select="paper" group-by="title">
<xsl:sort select="title"/>
<tr><td>
<table border="1">
<tr><td>
<xsl:value-of
select="current-group()/title"/>
</td></tr>
</table>
</td></tr>
<tr><td>
<xsl:apply-templates select="authors"/>
</td></tr>
```

```
</xsl:for-each-group>
</xsl:template>
<xsl:template match="authors">
<xsl:for-each-group
select="author" group-by="univ">
<xsl:sort select="univ"/>
<table border="1">
<tr><td>
<xsl:for-each select="univ">
<xsl:value-of select="."/>
</xsl:for-each>
</td><td>
<table border="1">
<xsl:for-each select="current-group()">
<tr><td>
<xsl:value-of select="name"/>
</td></tr>
</xsl:for-each>
</table>
</td></tr>
</table>
</xsl:for-each-group>
</xsl:template>
</xsl:stylesheet>
```

However, the style sheet requires also larger amount of description than with XQuery and it is not possible to recycle the initial style sheet, it is necessary to rewrite it.

C. Comparison of transformation abilities

PPX can easily transform XML data into HTML by reconstructing the tree structure by combining the layout specification operators. Moreover, according to changes in the variables and the layout specification operators, the conversion of nest structure, generating more HTML tables which hyperlink to one HTML table, data decoration and expressing the searched XML data included in a part of XML by XList form, can be easily performed.

XQuery, XSLT etc. transform the XML data into HTML by describing HTML tags directly in the program sentences or the searched XML data can be converted into HTML by using XSL-FO(CSS).

In this case, XQuery for extracting XML data uses a lot of FOR clauses and LET clauses will be nested in the RETURN clause, and the condition which child element is in which element should be specified in detail.

In this case, XSLT includes XSLT 1.0 and XSLT 2.0. As for XSLT 1.0, one of the greatest problems is that it can not execute SELECT DISTINCT directly for node groups. For this kind of conversion, all nodes whose element names become group objects are selected and sorted according to their element names. In addition, it is necessary to distinguish whether the element name, after using the xsl:if block and processing, is the same as the element name of the nodes or not. Besides, it is complex to use the Muenchian method because it does not support making the group, and consumes more memory. The XSLT 2.0 uses xsl:for-each-group to group nodes based on some standards, and it processes for every group formed by selection processing.

Moreover, XSLT is intuitively difficult to describe and to edit, since the users doing these programmings need to understand the transformation process based on the rival cancellation between template rules to convert the conversion originally specified by the pattern matching is demanded.

In addition, XQuery and XSLT does not provide easy methods that can express XList form for the searched XML data included in a part of XML by an incomplete path expression.

D. Examination of formatting method

In the following section, we will discuss the problem of this fixed formatting method encountered when experimenting and that should be resolved.

1) Irregular element nodes: the fixed formatting method annnot easily process the irregular element nodes. For example, in the XML instance as figure 2, we know that the first author's univ element node has the text node, but the second author's first univ element node has several element nodes. To layout the XML data of the univ element nodes, it is required to specify the format in the layout expression as shows in PPX 6.

```
PPX 6:
GENERATE html
 [ $i/title ! [ $j/name ,
   [ $j/univ ]!
 ]! ]!
 !
 [ $k/title ! [ $l/name ,
   [ $l/univ/name ! $l/univ/add ]!
 ]! ]!
FOR $i in db('paper.xml')/papers/paper,
FOR $j in $i/authors/author,
FOR $k in db('paper.xml')/papers/paper,
FOR $l in $k/authors/author
```

In this query, as the layout of the output table becomes complex, the description amount increases as well. To solve this problem, we introduce the IF-THEN-ELSE sentence as described in the layout expression of PPX. Then, the layout of the XML data can be done with a minimum description amount. Also, the specification of irregular element node as the univ element node can be easily done as in following PPX 7.

```
PPX 7:
GENERATE html
 [ IF ( $j/univ/text() )
    THEN ( $j/univ )
     ELSE ( $j/univ/name ! $l/univ/add ) !
  [ $j/name ,
   [ $i/year ,
    [ $i/title ]!
 ]! ]! ]!
FOR $i in db('paper.xml')/papers/paper,
FOR $j in $i/authors/author
```

2) Empty element nodes: Having introduced the IF-THEN-ELSE sentence, processing empty element node has become easy. For example, the following PPX 8 test whether the univ element nodes is empty element and if it is the case uses another method to process it.

```
PPX 8:
GENERATE html
 [ IF ( $j/univ/text() = NULL )
    THEN ( "$j/univ" )
     ELSE ( $j/univ ) !
  [ $j/name , [ $i/year , [ $i/title ]!
 ]! ]! ]!
FOR $i in db('paper.xml')/papers/paper,
FOR $j in $i/authors/author
```

However, if the element node "$j/univ" is empety, it does not group the name element nodes together.

3) Automatic formatting method: If there is an important part of XML under the univ element node, the specification of formatting method becomes redundant. Therefore, we are developing an automatic layout operator to automatically format XML data included in a part of XML. For example, the following PPX 9 using combinations of the & operator and the variable $j/univ in the layout expression, can be automatically format XML data included in a part of XML is under the univ element nodes.

```
PPX 9:
GENERATE html
 [ & ( $j/univ ) !
  [ $j/name , [ $i/year , [ $i/title ]!
 ]! ]! ]!
FOR $i in db('paper.xml')/papers/paper,
FOR $j in $i/authors/author
```

VI. Related Work

Three methods to transform XML data into HTML are categorized in this section.

A. By using HTML tags

Generic programming languages, such as JAVA, PERL, PHP, and C++ are used to convert searched XML data tags into HTML tags with DOM or SAX and to display in Web browser. Besides, this is also possible with languages such as XQuery, XSLT 1.0, XSLT 2.0, and XDuce [11], etc. provided the HTML tags in the program sentence.

B. By using XSL-FO(CSS)

The query language, the converting language and the stylesheet language give formatting information such as the margin, the color, and the font size, etc. for the searched/extracted XML data in order to display it in Web browser by using XSL-FO [12] or CSS[13].

C. By using TFE

The proposed PPX, which can layout XML data into HTML by using TFE, offers an easy description method. SuperSQL also uses TFE [14] to structurize the output result of the relational database and treats the output to HTML, XML and PDF, but can not be treated as XML data.

VII. Conclusions and future work

In this paper, we propose a fixed formatting method of PPX, which is to completely specify the combination of variables and layout specification operators within the layout expression of the GENERATE clause to layout XML data into HTML. In the experiment, the results show that with fixed formatting method, the XML data can be formatted correctly.

We are currently working on developing automatically formatting XML data that does not only express XList form, but also layout into HTML without completely specifies the formatting method. Moreover, we are developing a method to convert PPX query into equivalent XSLT generated automatically.

References

[1] W3C: XML Query Language (XQuery). http://www.w3.org/TR/.
[2] J. Clark, editor, XSL Transformations (XSLT), Version 1.0, W3C Recommendation 16 November 1999. W3C, 1999.
[3] M. Kay, editor, XSL Transformations (XSLT), Version 2.0, W3C Recommendation 27 January 2007. W3C, 2007.
[4] Ramin Firoozye, XML and XSL from servers to cell-phones, a new Internet content model, Proceedings of XML Europe2000, Paris, france, 2000.
[5] Volker Turau, A Caching System for Web Content Generated from XML Sources Using XSLT, OOIS 2002 Workshops, LNCS 2426, 2002, pp.197-207.
[6] M. Rys, XQuery in Relational Database Systems, XML 2004 Conference, Washington DC, Nov 2004.
[7] W3C: XML Path Language (XPath). http://www.w3.org/TR/.
[8] M. Toyama, SuperSQL: An Extended SQL for Database Publishing and Presentation, Proc. ACM SIGMOD, 1998, pp.584-586.
[9] UW XML repository: http://www.cs.washington.edu/research/xmldatasets.
[10] www.Altova.com/XMLSpy.
[11] H. Hosoya and B. C. Pierce, XDuce: A Statically Typed XML Processing Language, ACM Transactions on Internet Technology, 2003, pp.117-148.
[12] Pawson, D, XSL-FO: Making XML Look Good in Print, O'Reilly, United States, 2002.
[13] Lie. H., Bos, B, Lilley, C. and Jacobs, I., Cascading Style Sheets, Level 2. W3C: www.w3.org/TR/.
[14] T. Seto, T. Nagafuji, M. Toyama, Generating HTML Sources with TFE Enhanced SQL, ACM Symposium on Applied Computing(SAC'97), ACM(1997), pp.96-105.

Zhe JIN received the M.E. degree in computer science from keio university in 2004. He is a member of IPSJ and IEICE. His research interests include XML and database.

Motomichi TOYAMA received the B.E., M.E. and Ph.D. degrees in computer science from keio university in 1979, 1981 and 1984, respectively. His research interests include database. He is a member of IEEE Computer Society, ACM, IPSJ and IEICE.