International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:2, No:9, 2008

A proposal of an automatic formatting method for transforming XML data

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

Abstract—PPX(Pretty Printer for XML) is a query language that offers a concise description method of formatting the XML data into HTML. In this paper, we propose a simple specification of formatting method that is a combination description of automatic layout operators and variables in the layout expression of the GENERATE clause of PPX. This method can automatically format irregular XML data included in a part of XML with layout decision rule that is referred to DTD. In the experiment, a quick comparison shows that PPX requires far less description compared to XSLT or XQuery programs doing same tasks.

Keywords—PPX, Irregular XML data, Layout decision rule, HTML.

I. Introduction

AS well known, there exists a large amount of data described with XML, which is also used for the exchange of data, the description, the processing of information, etc.. Since XML is an expressive form of data or the document, which is necessary for technologies of storage, the retrieval, conversion, and publication, etc., the research on this area is quite active.

In this research, the method that converts XML data into HTML is discussed. A few existing languages, such as XQuery [1], XSLT 1.0 [2], XSLT 2.0 [3], JAVA and C++ etc. transform XML data into HTML by describing HTML tag directly in the program sentences, or convert the searched XML data into HTML by using XSL-FO [4] or CSS [5]. However, it is to necessary examine detail of data structure to layout XML data for using XQuery and XSLT etc. [6], [7], [8].

PPX is a query language for XML database, which has extensive formatting capability that produces HTML as the result of a query. This query language aims to focus on the design of the layout without considering XML data structure directly, so that the layout work of the XML data can be done easily.

For example, the following PPX 1 shows that how XML data in Figure 1 is layouted into HTML by complete specification of formatting method in the layout expression [1] of the GENERATE clause.

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

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

Fig. 1. Format results by PPX 1

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.

Moreover, the following PPX 2 changes only the variables location based on layout expression of PPX 1 and the results are shown in Figure 2, which is structure different with Figure 1.

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

The complete specification of formatting method of PPX shows that the extracted XML data structure is converted and layouted according to the variable location changes, variable addition, grouping of element node using the layout specification operators in the layout expression. In addition, the XML data can be decorated when HTML is generated by specifying the decoration operators.

However, the complete specification of formating method

University of Washington	
Birgit Aalderink	A Query Language for Semantic Web
	Path Selectivity Estimation for XML Data
Oxford University	
Ann Aalto	A Query Language for Semantic Web
Harvard University	
Francois Bancilhon	A Query Language for Semantic Web
Stanford University	
Samuli Aaltonen	A Query Language for Semantic Web

Fig. 2. Format results by PPX 2

of PPX that uses complete path expression [2] to specify the element nodes included in a part of XML [3] is not easy. A part of XML is under irregular element nodes. The irregular element node is an element node wherever appears by the "|" specified in DTD of table I. For example, under the univ element nodes, it has the text node and also some how many element nodes contain other element nodes in XML shown in Figure 4. The following PPX 3 specifies the element nodes that is under univ element nodes based on layout expression of PPX 2.

```
PPX 3:
GENERATE html
 [ $j/univ !
  [ $j/name ,
   [ $i/title ]!
 ]! ]!
 !
 [ $j/univ/name , $j/univ/add !
  [ $j/name ,
   [ $i/title ]!
  ]! ]!
FOR $i in db('paper.xml')/papers/paper,
FOR $j in $i/authors/author
```

In this query, the layout of output table becomes complex, also the description amount increases. However, if there is an important part of XML is under the univ element nodes, the complete specification of formatting method also becomes redundant.

In this paper, we propose a simple specification of formatting method of PPX that is a combination description of automatic layout operators and variables in the layout expression of the GENERATE clause. This method can automatically format XML data included in a part of XML with layout decision rule that is referred to DTD. The XML data included in a part of XML which is treated as irregular XML data in this paper. This rule automatically convert table structure of HTML and naturally express it based on the structure of source XML data.

The following PPX 4 specifies the simple specification of formatting method by combining the $j/univ variable and the

[2]discussed at II.A section with incomplete path expression.

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

Fig. 3. Format results by PPX 4

& automatic layout operator embed in the layout expression, and generates HTML as Figure 3.

```
PPX 4:
GENERATE html
 [  &( $j/univ ) !
  [ $j/name ,
   [ $i/title ]!
 ]! ]!
FOR $i in db('paper.xml')/papers/paper,
FOR $j in $i/authors/author
```

In this query, the $j/univ variable shows searched XML data included in a part of XML, which is under univ element nodes using the incomplete path expression. The & automatic layout operator shows automatically formatting XML data with the layout decision rule.

The rest of this paper is organized as following. In Section II, the basic concepts are discussed. In Section III and Section IV, the PPX query language and layout decision rule is review. In Section V, the simple specification of formatting method of PPX is evaluate. In Section VI, the related work is mention. Finally, the conclusion is given in section VII.

II. Basic Concepts

This section introduces the path expressions, the path expression sets and the irregular XML data of the PPX query language.

A. Path Expressions

The path expressions [9] include absolute path expression and relative path expression. If they need not be distinguished, in the following sections, they are both referred as the path expression. Here, the path expression is further 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 path expression.

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

In the PPX 4, the complete path expression is connected with the path expression used in the first FOR clause and the relative path expression used in the $i/title variable. The complete path expression searches for the XML data, which are the value of the title element nodes. The text node is omitted when it is specified in the layout expression of the GENERATE clause of PPX.

(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 path expression.

Example 2: /papers/paper/authors/author/univ

In the PPX 4, the incomplete path expression is connected with the path expression used in the first FOR clause, the second FOR clause and the relative path expression used in the $i/univ variable. The incomplete path expression searches for the XML data, which are the value of the element nodes included in a part of XML is under the univ element nodes. The XML data searched by the incomplete path expression is the object of an automatic format.

B. Path expression sets

The path expression sets are composed with three kinds of path expression sets, which are query path expression set P(Q), XML path expression set P(X), and DTD path expression set P(D). Each of path expression sets includes the complete path expression and the incomplete path expression.

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

(2) P(X): This is a group of the path expressions existing in the layout object part of XML in Figure 4.

(3) P(D): This is a group of the path expressions existing in the part of DTD (table I) of P(X).

The following explaination is about the irregular XML data from the relationship among three kinds of path expression sets.

TABLE I
A FRAGMENT OF DTD

DTD
$<!DOCTYPE\ papers[$
$<!element\ papers(paper+)>$
$<!element\ paper(id, title, authors)>$
$<!element\ id(\#pcdata)>$
$<!element\ title(\#pcdata)>$
$<!element\ authors(author+)>$
$<!element\ author(name, email, univ)>$
$<!element\ name(\#pcdata)>$
$<!element\ email(\#pcdata)>$
$<!element\ univ(\#pcdata \mid name, add, tel \mid name, papers)?>$
$<!element\ name(\#pcdata)>$
$<!element\ add(\#pcdata)>$
$<!element\ tel(\#pcdata)>$
$]>$

C. Irregular XML data

Irregular XML data is the XML data included in a part of XML that exists under irregular element nodes, which is the object of an automatic format. The irregular XML data is further divided into valid XML data and invalid XML data.

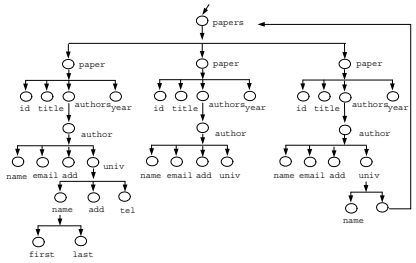

Fig. 4. An XML instance

Definition 1 (The valid XML data) The path expression P that in the intersection of P(X) - P(Q) and P(D), when t, which exists in incomplete path expression sets P(Qi) of P(Q) is the prefix of path expression P. The group of path expression P is shown as following:

$\{P|P \in ((P(X) - P(Q)) \cap P(D)) \wedge (\exists t \in P(Qi))\} : t$ is prefix of P.

The valid XML data is the value of path expression sets P. For example, the value of the XML path expression sets included in a part of XML that exists under univ element nodes by the incomplete path expression, which specifies the univ element node shown in example 2 of section II.A, is irregular XML data in the XML data of Figure 4. Here, the value of the XML path expression sets that is compatible to DTD in Table 1 is valid XML data, and these not to DTD is invalid XML data.

Definition 2 (The invalid XML data) The path expression P that in the part of from P(X) - P(Q) - P(D), when t, which exists in incomplete path expression sets P(Qi) of P(Q) is the prefix of path expression P. The group of path expression P is shown as following:

$\{P|P \in ((P(X) - P(Q)) - P(D)) \wedge (\exists t \in P(Qi))\} : t$ is prefix of P.

The invalid XML data is the value of path expression sets P.

III. PPX

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

```
GENERATE <media> <Layout Expression>
FOR <"$"+Variable Name in Path Expression>
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 study, only HTML output medium in discussed. 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, the complete specification of formatting method and the simple specification of formatting method can be specified. The complete specification of formatting method that combines of variables and layout specification operators can be specified. The simple specification of formatting method that combines of the variables and the omission operators or the automatic layout operators can be specified.

1) Operators: The operators are the extensions of layout specification operators of SuperSQL [10]. They comprise omissible operators, automatic layout operators and the existed layout specification operators that include connect operators, repeat operators, and decorative operators etc.

(1) Connect Operators

There are horizontal (,), vertical (!) and depth (%) connect operators which connect the objects generated as their operands horizontally, vertically and in the depth direction, respectively. In the case of generating HTML, the depth connect operator specifies the hyper link in a hypertext (figure 5).

(2) Repeat Operators

There are horizontal([ ],), vertical([ ]!), and depth([ ]%) repeat operators. In a pair of brackets, a layout expression is specified. The multiple instances generated by the inner layout expression are connected repeatedly into 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 6).

(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. Each e_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 that without tags, and the &+ operator expresses XList form with tags, for the searched XML data included in a part of XML by an incomplete path expression.

(5) Automatic Layout Operators

Automatic layout operators include the & operator. The & operator automatically format irregular XML data included in a part of XML with layout decision rule that is referred to DTD.

2) Precedence: Usually, the leftmost connector takes priority over the other connectors. When it is necessary to change the precedence, pair of curly braces should be used.

3) 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 following.

Variable ::= "$"+VariableName/RelativePathExpression

In case of the complete specification of formatting method, 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. For other case of the simple specification of formatting method, we use a incomplete path expression in which the relative path expression specified in the variable is connected with the path expression specified in the FOR clause.

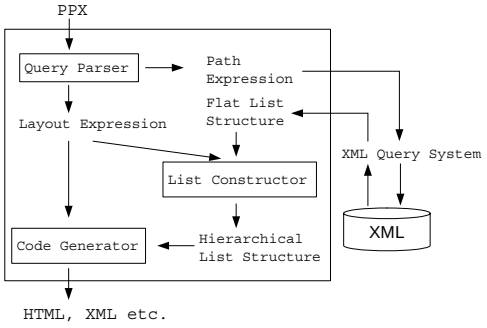

Fig. 7. System architecture

B. System architecture

This system consists of the query parser, the list constructor, and the code generator as shown in Figure 7. The PPX 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 list constructor. Therefore, the irregular XML data included in a part of XML do not reconstructe and it remaines its former structure and passes to the list constructor as one tupple. Finally, the reconstructed XML data is transformed into HTML with a variety of table structures in the code generator. The irregular XML data automatically being transformed into HTML with layout decision rule that discussed at section IV.

IV. Layout Decision Rule

The layout decision rule is divided into following four steps. The flow is shown in Figure 8.

(1) Step 1. (Generation of Path Expression Sets)

The first step, from a part of XML that is object of layout generate XML path expression sets, and from a part of DTD that is corresponding to a part of XML, generates tree patterns with DTD path expression sets (In section IV.A).

(2) Step 2. (Connection Method of Tree Patterns)

The connection method is given respectively to the DTD path expression sets of each tree patterns by referring to the information of DTD (In section IV.B).

(3) Step 3. (Matching of Path Expression Sets)

The DTD path expression sets with the connection method is matched to the XML path expression sets (In section IV.C).

(4) Step 4. (Output Results)

The irregular XML data is converted into HTML by tagging(In section IV.D).

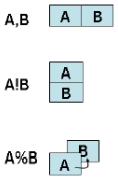

Fig. 5. Connct operators

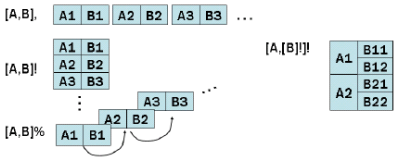

Fig. 6. Repeat operators and grouping

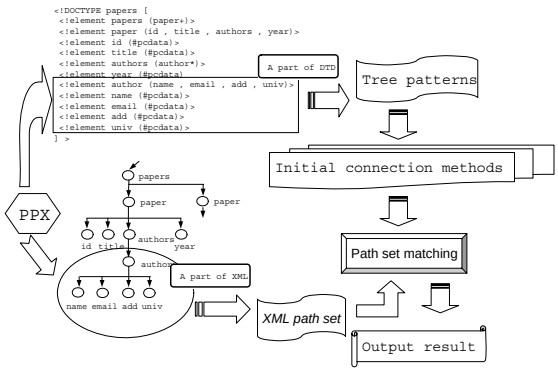

Fig. 8. Overview of layout decision rule

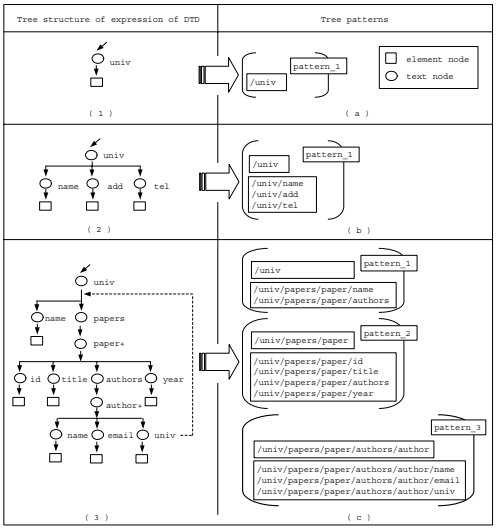

Fig. 9. Tree patterns with DTD path expression sets

A. Generation of Path Expression Sets

The XML path expression sets generate from each level of a part of XML that is object of layout by the depth division. Moreover, the DTD path expression sets generate in a part of DTD that is corresponding to a part of XML by the depth division also. However, the DTD path expression sets do not generate at recurrent part. Afterwards, it resolves based on the level with the text node, and these DTD path expression sets compose tree patterns.

For example, at the left of Figure 9, it shows three kinds of a part of DTD generated by the $j/univ variable that specifies in the layout expression of PPX 4 in chapter 1, and at the right of Figure 9, it shows the tree patterns with the DTD path expression sets respectively. However, at the recurrent part that under the univ element nodes in Figure 9(3), neither the path expression sets nor the tree patterns is generated.

B. Connection Method of tree patterns

The tree patterns include frequent tree pattern and non-frequent tree pattern. Here, the frequent tree pattern is repetition appearance, and the non-frequent tree pattern is not. For example, a tree pattern 1 is non-frequent tree pattern, and the tree pattern 2 and 3 are the frequent tree patterns shown at the right of Figure 9(3).

The frequent tree pattern has the frequent DTD path expression sets and the non-frequent DTD path expression sets. The non-frequent tree pattern only has non-frequent DTD path expression sets. Those tree patterns have different connection methods by different DTD path expression.

The algorithm that gives the connection method to the tree patterns is shown in Figure 10 [4].

First of all, the DTD path expression sets of the tree patterns judge the repetition or not. Then, if it is a repeated DTD path expression set, the vertical repetition is given. On the contrary, the previous connection method is referred, if it is a not repeated DTD path expression set. That is, if the connection method of previous DTD path expression set is horizontal connection, the vertical connection is given to present DTD path expression set, and if the connection method of previous DTD path expression set is vertical connection, the horizontal connection is given to present DTD path expression set.

It will refer to the repeated operators exists on the outside in layout expression when can not refer to the previous connection method. That is, if it is a horizontal repeat operator, the vertical connection method is given to present DTD path expression set, and if it is a vertical repeat operator, the horizontal connection method is given to the present DTD path expression set. It repeats to the last tree pattern like this.

C. Matching of Path Expression Sets

The tree patterns with the connection methods is matching to a part of XML. That is, the DTD path expression sets of the

[4]C1 is horizontal connection. C2 is vertical connection. G1 is horizontal repeat connection. G2 is vertical repeat connection.

```
Input: A part of DTD
output: The connection methods of DTD path expression sets

01. while (It is not last level of DTD)
02.    If (Frequent DTD path expression)
03.       Then (Giving G2)
04.    Else if (Non-frequent DTD path expression)
05.       if (Previous connection method exist)
06.          if (Previous connection method is C1)
07.             Then (Giving C2)
08.          Else if (Previous connection method is C2)
09.             Then (Giving C1)
10.          Else if (Previous connection method is G1)
11.             Then (Giving C2)
12.          Else if (Previous connection method is G2)
13.             Then (Giving C1)
14.       Elst if (Previous connection method doesn't exist)
15.          if ([ ]! exist on the outside in layout expression)
16.             Then (Giving C1)
17.          Else if ([ ], exist on the outside in layout expression)
18.             Then (Giving C2)
19. endwhile
```

Fig. 10. Algorithm to generate connection methods

tree patterns included in a part of DTD matches to the XML path expression sets included in a part of XML, and connectes with the XML data by the connection methods of the tree patterns. The matching methods include complete matching and suffix matching.

The complete matching satisfies the following conditions.

(1) Absolute matching. It is necessary to agree from the root node to the element node of the DTD path expression sets by the XML path expression sets completely.

(2) Complete including. All the XML path expression sets should exist in the DTD path expression sets.

(3) All matching. All tree patterns should matche to a part of XML.

If it does not satisfy any one of those conditions, it is not complete matching. When a part of XML of which recurrence appears under the univ element nodes as shown in Figure 9(3), the suffix matching is used. The HTML generated by complete matching is connected with other HTML table generated by suffix matching by hyperlink.

The suffix matching satisfies the following conditions.

(1) Absolute matching. It is necessary to agree from the root node to the element node of the DTD path expression sets by the XML path expression sets completely.

(2) Complete including. All rear part of the XML path expression sets should exist in the DTD path expression sets.

(3) All matching. All tree patterns generated from a part of DTD should match to rear part of a part of XML.

If either of above conditions does not satisfy, the suffix matching is not done, and matching is ended.

D. Output Results

The XML path expression sets that match to the DTD path expression sets and have data are valid XML data, and can be converted to HTML form by giving HTML tags. Other XML path expression sets that does not match and have data are expressed with XList form by giving XList tags in HTML table.

V. Experimental Evaluation

In this section, we have implemented the proposed simple specification of formatting method with layout decision rule and comparison of the proposed PPX 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 simple specification of formatting method encountered when experimenting which should be resolved.

A. Experimental Environment

According to the algorithm of Figure 10, we implemented the layout decision rule using Java. We used XML data (table II) of UW XML repository [11] to generate HTML. The XML data included in a part of XML that searched by the incomplete path expression. Table III shows each incomplete path expression corresponding to different part of XML, which is the object of an automatic format. The DB2 Version 9 is used for PPX and XQuery. The XMLSpy [12] is used for XSLT 1.0 and XSLT 2.0.

TABLE II
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

TABLE III
Different part of XML

incomplete	a part of XML
/papers	under the PAPERS element nodes
/papers/paper[1]	under the first PAPER element node
/papers/paper/authors	under the AUTHORS element nodes
/papers/paper/authors/univ	under the UNIV element nodes

B. Description Amount

1) PPX: PPX transforms XML data into HTML with a small description amount as shown in the previous section and the initial query is possible to recycle. On the other hand, XQuer, XSLT 1.0, XSLT 2.0 do the same transformation as PPX but need large description amount.

2) 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>;
```

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

3) 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" />
<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 larger amount of description than with XQuery, and it is not possible to recycle the initial style sheet, is necessary to rewrite it.

C. Transformation Abilities

1) Structural Conversion: The simple specification of formatting method of PPX for extracting XML data uses the FOR clause (WHERE clause). This method can automatically format XML data into HTML or express the XML data included in a part of XML by XList form without considering XML data structure. However, since the conversion of the XML data is based on source data structure, the degree of freedom of structural conversion is limited. As the result, it is necessary to develope a new automatic layout method of based on various rules with becoming of the degree of freedom of structural conversion flexibility.

In the case of XQuery, for extracting XML data, a lot of the FOR clause and the LET clause will be nested in the RETURN clause, and the condition which child element is in which element should be specified in detail. This query transforms the XML data into HTML by describing HTML tags directly in the program sentences or by using XSL-FO(CSS) with considering XML data structure.

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 element to group nodes based on some standards, and it processes for every group formed by selection processing.

Moreover, XSLT is intuitively difficult for ordinary user to describe and to edit, since the understanding of the transformation process based on the rival cancellation between template rules to convert the conversion originally specified by the pattern matching is demanded. Also, it transforms the XML data into HTML by describing HTML tags directly

in the program sentences or by using XSL-FO(CSS) with considering XML data structure.

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.

2) Data Representation: XQuery and XSLT can extract, sort and group spreferable data of users to layouts. PPX extracts all data included in a part of XML and automatic formats with the simple specification of formatting method. This method is more convenient than complete specification to process the irregular element nodes. However, the XML data that user prefers can not be layout. Moreover, because it not sort/group to the XML data, so same data are appearing many times and source data structure remains in HTML tables.

3) Data Matching: When the DTD path expression sets of tree patterns do not matches the XML path expression sets of a part of XML. The XML path expression sets, may have valid XML data and invaild XML data. Then it is expressed in XList form. So, whether it is valid XML data or invalid XML data, it can not be telled by the HTML tables.

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 [13], 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 formating 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 or CSS.

C. By using TFE

The proposed PPX, which can layout XML data into HTML by using TFE [14] offers an easy description method. SuperSQL also uses TFE 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 simple specification of formatting method of PPX, which is a combination of variables and automatic layout operators within the layout expression of the GENERATE clause to layout irregular XML data included in a part of XML into HTML with layout decision rule. This rule can be converted into the table structure of HTML that naturally expresses based in the structure of the source XML data automatically. In the experiment, the results show that by PPX the XML data can be formatted correctly without considering the data structure etc.

We are currently working on developing a new layout decision rule that takes statistics of concerning the structure and the content to the XML data included in a part of XML, which is searched and structurized based on the processing system of XQuery and XSLT for deciding the best layout method and automatically formatting XML data. Moreover, we are developing the method of converting PPX into equivalent XSLT automatically.

References

[1] D. Chambelin, J. Clark, D. Florescu, J. Robie, J. SimLeon, and M. Stefanescu. XQuery 1.0: An XML query language. W3C Working Draft, June 2001.
[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] Pawson, D: XSL-FO: Making XML Look Good in Print. O'Reilly, United States, 2002.
[5] Lie. H., Bos, B, Lilley, C., and Jacobs, I. Cascading Style Sheets, Level 2. W3C; see www.w3.org/TR/.
[6] Ramin Firoozye: XML and XSL from servers to cell-phones, a new Internet content model. Proceedings of XML Europe2000, Paris, france, 2000.
[7] Volker Turau: A Caching System for Web Content Generated from XML Sources Using XSLT. OOIS 2002 Workshops, LNCS 2426, pp.197-207, 2002.
[8] M. Rys: XQuery in Relational Database Systems. XML 2004 Conference, Washington DC, Nov 2004.
[9] W3C: XML Path Language (XPath). http://www.w3.org/TR/.
[10] M. Toyama: SuperSQL: An Extended SQL for Database Publishing and Presentation. Proc. ACM SIGMOD, 1998, pp.584-586.
[11] UW XML repository: http://www.cs.washington.edu/research/xmldatasets.
[12] www.Altova.com/XMLSpy.
[13] H. Hosoya and B. C. Pierce. XDuce: A statically typed XML processing language. ACM Transaction on Internet Technology (TOIT), pp.117-148, May 2003.
[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.