AN IMPLEMENTATION OF XML DATA INTEGRATION

Weidong Pan, Jixue Liu and Jiashen Tian
School of Computer and Information Science, University of South Australia
Mawson Lakes, Adelaide, SA 5095, Australia

Keywords: XML, data Integration, enterprise infiormation sytsem, DTD, document, data transformation.

Abstract: Data integration is essential for building modern enterprise information systems. This paper investigates the implementation of XML data integration through transforming XML data from different data sources into a common global schema. Following the work our research group has done earlier, the paper is focused on the implementation of XML data transformation. First, the proposed methodology to realize XML data integration is sketched. Then, the representation of a DTD and document and other relevant concepts for transforming XML data are presented. The transforming operations defined by a set of operators are outlined focusing on the required functionality for data integration. Building upon these, the implementation of the data transformation operations is investigated. The current implementation is reported with a simplified example illustrating how the methodology can be applied for practical enterprise information integration.

1 INTRODUCTION

Data integration is essential for building modern enterprise information systems since data is normally distributed on different platforms. In order to integrate these data together to provide a unique view for users, there is a need for technologies to implement the conversion of data from different sources to a common unified schema. Since XML has been increasingly used for data representation and exchange across the Internet, it is of specific importance to integrate XML data from multiple data sources into a global schema with a unique structure. This paper investigates the implementation of XML data integration, more specifically, it aims to realize the integration through converting XML data from different schemas into a unified global schema.

A number of earlier research has devoted to XML data integration. The techniques proposed can be summarized into two main procedures, *scheme mapping* and *data conversion*. The former aims to develop techniques to map XML data from different sources to a common global schema, through which XML data distributed at different sources can be represented in a unique view; the latter aims to, based on the mapping between an original and a destination schema, develop techniques to convert XML data from different sources into a destination platform that conforms to an integrated global schema. Data conversion can be done through query processing or executing a sequence of transformation operations. By query processing, users retrieve data from a document and then build another document using the retrieved data. W3C's XQuery (Boag et al., 2007) and XSLT (Kay, 2007) are two typical query languages. Data conversion can also be implemented by executing a sequence of data transformation operations defined by a set of operators. Data transformation operators have been proposed in a number of previous work, some are similar to XML algebra expressions derived from relational data model (Zamboulis, 2004). In general, they just provide operators for XML document update and have not provided a systematic set of transformation operators with a clear semantic. Although a few has considered DTD transformation when document is being transformed (Erwig, 2003), none has covered the full DTD syntax. In addition, to the best of our knowledge, the implementation issue of the operators has not been well addressed and accordingly their performance has not been deeply studied from the viewpoints of practical applications.

To address these problems, our research group has proposed a set of XML data transformation operators to realize XML data integration (Liu et al., 2006). Compared with other data transformation operators, our operators include two types of transformation: 1) DTD transformation, and 2) document transformation. This can ensure the output documents of the operators always conform to the output DTDs, which is critical to the semantics of output data. Our operators are defined with the consideration of the full syntax of DTDs and documents, especially the nested

Pan W., Liu J. and Tian J. (2008).
AN IMPLEMENTATION OF XML DATA INTEGRATION.
In *Proceedings of the Tenth International Conference on Enterprise Information Systems - DISI*, pages 111-116
DOI: 10.5220/0001677401110116

brackets in element type definitions, multiplicity constraints attached to those nested brackets, and disjunction. They are complete for the transformation of DTDs and documents. In addition, they are semantically traceable when being used to realize XML data integration; each has a particular intention and creates a particular semantic effect towards the overall goal. Up to this point, we have completed a preliminary implementation of these operators using Java and JSP techniques. This paper is to report our recent research in XML data integration, focusing on the implementation of the operators, as the performance analysis for the operators has been presented in another paper (Tian et al., 2008).

The paper is organized as follows. In Section 2, an overview of the proposed methodology is presented. Section 3 illustrates the representation of XML data, providing some essential concepts and terms. Section 4 describes the data transformation operations defined by the data transformation operators. Section 5 looks into the implementation of the operators. Section 6 reports the current implementation and presents a simplified example to illustrate how the approach is used in practical enterprise information integration applications. The final section summarizes the paper and indicates the further work.

2 OVERALL DESCRIPTION OF OUR METHODOLOGY TO REALIZE XML DATA INTEGRATION

As stated in the previous section, what we aim to achieve is XML data integration by converting XML data to a unified schema from different sources without losing valuable semantic information. It is complicated by the flexible XML syntax that allows different ways to represent the data of same semantics. We realize XML data conversion using a set of transformation operators that supply enough transformation power and preserve the data semantics to a certain degree.

The proposed methodology to achieve XML data integration is through data transformation. All the DTDs and the conforming documents from different data sources are converted to a unified XML format that conforms to the integrated global schema. As illustrated in Figure 1, in the conversion of a DTD from a particular source, a four-tuple expression is built based on the given DTD file. Then, the tuple expression is transformed to a new one through executing a sequence of data transformation operations, defined by a set of operators. After the transformation, the new tuple expression is restored into a new DTD file, achieving a conversion from the previous DTD to a new one. The transformation of an XML document from a particular source to a targeted schema has a similar process except that it requires the support from the transformation of its conformed DTD to ensure the output document has a compatible semantic structure.

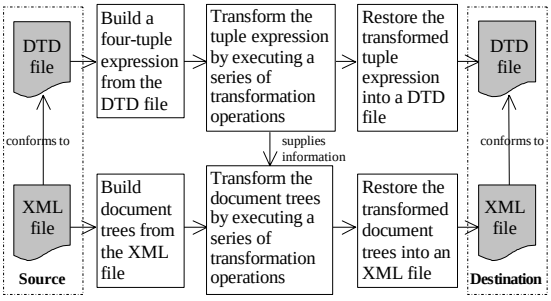

Figure 1: Framework for XML data transformation.

In the sections that follow, we will elaborate the methodology outlined here.

3 XML DATA REPRESENTATION

3.1 DTD Representation

A DTD defines the structure of its corresponding documents by a list of element type declarations. In the proposed approach, this information is represented by a four-tuple $D = (EN, G, \beta, \rho)$, where EN is the set of the element names in the DTD; G is the set of type constructors which define the elements; β is the set of the functions connecting an element with its type constructor; and ρ is the root of the DTD. Figure 2 shows a simplified DTD example and its corresponding tuple expression. It is extracted from a DTD used in a publishing company for publication information.

Note, to save space, the headers and the string type #PCDATA in the DTD are intentionally left out. An element in G can be a single element, or a component including multiple elements in conjunctive or disjunctive sequences. For example, if $g \in G$, then g can be in the following forms: $g = e$ $(e \in EN)$, $g = Str$ (Str is a symbol denoting #PCDATA), or $g = g_1, g_2$ or $g_1|g_2$ or $[g]^c$, where g_1 and g_2 are recursively defined, and $c \in \{\text{'?'}, \text{'1'}, \text{'+'}, \text{'}*\text{'}\}$. The multiplicity c defines the multiplicity constraints of g. Transforming a DTD also involves the transformation of the multiplicities. For handling multiplicities, '?', '1', '+' and '$*$' are represented by the intervals $[0,1]$, $[1,1]$, $[1,n]$ and $[0,n]$,

```
<!ELEMENT root (name,publ)*>
<!ELEMENT publ (year,(book|article)+)*>
<!ELEMENT book (title,ISBN, price)>
<!ELEMENT article (title,journal,issue?,page)>
```

(a) *A Simplified DTD example*

$EN = \{root, name, publ, year, book, article, title, ISBN, price, journal, issue, page\}$
$G = \{Str, [name, publ]^*, [year, [book|article]^+]^*, [title, ISBN, price], [title, journal, issue^?, page]\}$
$\beta(root) = [name, publ]^*, \beta(publ) = [year, [book|article]^+]^*,$
$\beta(book) = [title, ISBN, price],$
$\beta(article) = [title, journal, issue^?, page],$
$\beta(year) = \beta(name) = \beta(title) = \beta(ISBN) = \beta(price) = \beta(journal) = \beta(issue) = \beta(page) = Str.$

(b) *The tuple expression of the DTD example*

Figure 2: A DTD example and its tuple expression.

respectively. The operations of two multiplicities c_1 and c_2 are conducted relying on the operations of their intervals. Thus, $c_1 \oplus c_2 (= c_1 c_2)$ has the semantics of the multiplicity whose interval encloses the intervals of c_1 and c_2, e.g., $+? = *$ and $1? = ?$. Similarly, $c_1 \ominus c_2$ is the multiplicity whose interval equals to the interval of c_1 taking that of c_2 and adding that of 1, e.g., $? \ominus ? = 1$ and $* \ominus + = ?$.

3.2 Document Representation

A well-formed XML document is a textual representation of data and is composed of elements with hierarchically nested structures as defined in its corresponding DTD. In our methodology, a document is represented by a series of trees $T = (e : val\ T_1\ T_2\ \cdots\ T_m)$, where $T_1\ T_2\ \cdots\ T_m$ are recursively defined child trees of T, $e : val$ is the root of T and the parent of $T_1\ T_2\ \cdots\ T_m$, and $e \in EN$, val is a text string if the root node contains a value, otherwise, it is omitted. Figure 3 is an example showing an XML document represented by such trees.

3.3 Hedge and Hedge Conformation

A hedge H is a sequence of trees under one node in a document. For instance, in the document shown in Figure 3, T, T_1T_2, $T_3T_4T_5$, and (title:ABC)(ISBN:-345)(Price:50) are four hedges. A hedge may contain smaller hedges. Here our interest is in which child trees of a node belong to a hedge conforming to a specific type constructor. For example, let $g^+ = [A, [B, C^?]^*, D^?]^+$, $\beta(e) = g^+$ and T=$(e((A)(B)(B)(C)(A)(B)(C)(D)))$, then hedge (A) conforms to $[A]$, hedge $(B)(B)(C)$ conforms to $[B, C^?]^*$, hedge $(A)(B)(B)(C)$ conforms to g, and hedge $(A)(B)(B)(C)(A)(B)(C)(D)$ conforms to g^+. A hedge H conforms to g is denoted by H^g.

By using the hedge notation, the child trees of a node can be logically split and thus, the cardinality constraints of a structure can be checked.

```
<root>
 <name>M. Fox</name>
 <publ>
  <year>2006</year>
   <book><title>ABC</title><ISBN>-345</ISBN> <price>50</price></book>
   <book><title>DEF</title><ISBN>-302 </ISBN><price>120</price></book>
  <year>2005</year>
   <book><title>XYZ</title><ISBN>-145</ISBN> <price>180</price></book>
   <article><title>FGH</title><journal>J1</journal><issue>2</issue><page>55-58
     </page></article>
  <year>2004</year>
   <article><title>XXX</title><journal>J2</journal><page>20-24</page></article>
   <article><title>T8</title><journal>J1</journal><issue>2</issue><page>8-15
     </page></article>
 </publ>
 <name>K. Page</name>
 <publ>
  <year>2006</year>
   <book><title>YYY</title><ISBN>-452 </ISBN> <price>200</price></book>
  <year>2004</year>
   <book><title>ZZZ</title><ISBN>-223</ISBN> <price>220</price></book>
  <year>2003</year>
   <article><title>GG</title><journal>J2</journal> <page>75-80</page></article>
   <book><title>TTTT</title><ISBN>-243</ISBN><price>180</price></book>
 </publ>
</root>
```

(a) *A simplified document example*

T = (*root*: $T_1\ T_2$)
T_1 = ((*name*: "M. Fox") (*publ:* ($T_3\ T_4\ T_5$))), T_2 = ((*name*: "K. Page") ···)
T_3 = ((*year*: "2006") (*book*: ···))
T_4 = ((*year*: "2005") ···), T_5 = ((*year*: "2004") ···)
·····

(b) *The document trees for the example*

Figure 3: A document example and its tree expression.

4 XML DATA TRANSFORMATION OPERATIONS

In the proposed methodology, the transformation of XML data is implemented through executing a series of data transformation operations against the tuple expression and the document trees. The data transformation operations are defined by a set of operators. Because of the syntax differences between DTD and document, each operator has defined two parts: one for transforming the DTD and the other for transforming its conforming documents. The formal definition of each operator has been presented in (Liu et al., 2006). This section will provide an overall description of the data transformation operations defined by those operators.

The DTD transformation operation that each operator performs is listed in Table 1. Because the document transformation operation of each operator is to convert a given document into one with a structure that conforms to the transformed DTD, the table also reveals the information for what transformation operation will be carried out on the conforming document by each operator. For instance, *unnest* operator converts the DTD $\beta(e) = [g_1, g_2^+]$ into a new DTD $\beta_1(e) = [g_1, g_2]^+$. The operator also transforms the document by converting the hedge

Table 1: The DTD transformation operation of each operator.

Operator	DTD transformation operations
opin	$\boldsymbol{opin}([g_1^{c_1},\cdots,g_n^{c_n}]^c) \longrightarrow [g_1^{c_1?},\cdots,g_n^{c_n?}]^{c\ominus?}$, where $c \supseteq ?$
opout	$\boldsymbol{opout}([g_1^{c_1},\cdots,g_n^{c_n}]^c) \longrightarrow [g_1^{c_1\ominus?},\cdots,g_n^{c_n\ominus?}]^{c?}$, where $i=1,\cdots,n(c_i \supseteq ?)$
min	$\boldsymbol{min}(c_c,i,[g_1^{c_1}\|\cdots\|g_i^{c_i}\|\cdots\|g_n^{c_n}]^c) \longrightarrow g=[g_1^{c'_1}\|\cdots\|g_i^{c'_i}\|\cdots\|g_n^{c'_n}]^{c\ominus c_c}$, where $c \supseteq c_c$ and if $i=0$: $\forall\, j=1,\cdots,n(c'_j=c_jc_c)$; else $i\in[1,\cdots,n]$: $c'_i=c_ic_c \wedge \forall\, j\neq i(c'_j=c_j)$
mout	$\boldsymbol{mout}(c_c,i,[g_1^{c_1}\|\cdots\|g_i^{c_i}\|\cdots\|g_n^{c_n}]^c) \longrightarrow g=[g_1^{c'_1}\|\cdots\|g_i^{c'_i}\|\cdots\|g_n^{c'_n}]^{cc_c}$, where if $i=0: j=1,\cdots,n$ $(c_j \supseteq c_c \wedge c'_j=c_j\ominus c_c)$; else $i\in[1,\cdots,n]$: $c_i \supseteq c_c \wedge c'_i=c_i\ominus c_c$ and $\forall\, j\neq i(c'_j=c_j)$
rename	$\boldsymbol{rename}(e,e_1) \longrightarrow e_1$, where $e_1 \notin parent(e)$
shift	Let $g=g_1,g_2$, $\boldsymbol{shift}(g_1,g_2) \longrightarrow g=(g_2,g_1)$
group	$\boldsymbol{group}(g) \longrightarrow [g]^1$
ungroup	$\boldsymbol{ungroup}([g]^1) \longrightarrow g$
expand	Let $g_e \in g \wedge e \notin parent(g_e)$, then $\boldsymbol{expand}(g_e,e) \longrightarrow g=e \wedge \beta(e)=g_e$
collapse	Let $g_p=e^c \wedge \beta(e)=[g_e]^{c_1}$ and $g_e \cap parent(e)=\phi$ then $\boldsymbol{collapse}(e) \longrightarrow g_p=[g_e]^{cc_1}$
nest	Let $g=[g_1,g_2^{c_2}]^c \wedge c \supseteq +$, $\boldsymbol{nest}(g_2) \longrightarrow g=[g_1,g_2^{c_2+}]^c$
unnest	Let $g=[g_1,g_2^{c_2}]^c \wedge c_2=+\|*$, $\boldsymbol{unnest}(g_2) \longrightarrow g=[g_1,g_2^{c_2\ominus+}]^{c+}$
fact	Let $g=[[g_0,g_1]^1\|\cdots\|[g_0,g_h]^1]^c$, then $\boldsymbol{fact}(g_0) \longrightarrow g=[g_0,[[g_1]^1\|\cdots\|[g_h]^1]^1]^c$
defact	Let $g=[g_0,[[g_1]^1\|\cdots\|[g_h]^1]^1]^c$, then $\boldsymbol{defact}(g_0) \longrightarrow g=[[g_0,g_1]^1\|\cdots\|[g_0,g_h]^1]^c$
merg	Let $g_1=e_1^{c_1},\cdots,e_m^{c_m}$ and $g_2=\bar{e}_1^{\bar{c}_1},\cdots,\bar{e}_m^{\bar{c}_m}$, where $\forall\, i=1,\cdots,m(\beta(e_i)=\beta(\bar{e}_i))$, then $\boldsymbol{merg}(g_1,g_2) \longrightarrow [e_1^{c_1\bar{c}_1}\times\cdots\times e_m^{c_m\bar{c}_m}]^+$
split	Let $g^c=[e_1^{c_1},\cdots,e_m^{c_m}]^c$, $c=*\|+$, then $\boldsymbol{split}(g) \longrightarrow g_1^{\bar{c}_1},g_2^{\bar{c}_2},\cdots,g_h^{\bar{c}_h}$, where $h\geq 2$ and $\bar{c}_1=c\ominus+$, $\bar{c}_2=\cdots=\bar{c}_{h-1}=?, \bar{c}_h=*$ and $g_1=[e_{11}^{c_1},\cdots,e_{m1}^{c_m}]$, $g_2=[e_{12}^{c_1},\cdots,e_{m2}^{c_m}],\cdots,g_h=[e_{1h}^{c_1},\cdots,e_{mh}^{c_m}]$ and $\forall\, i=1,\cdots,m(e_{i1}=e_i \wedge \forall\, j=2,\cdots,h(\beta(e_{ij})=\beta(e_i)))$ and all e_{ij} $(i=1,\cdots,m; j=2,\cdots h)$ are distinct and are not in $parent(g)$
proj	Let $g=[e_1^{c_1},\cdots,e_m^{c_m}]^{c_g}$, $f=[\bar{e}_1^{\bar{c}_1},\cdots,\bar{e}_w^{\bar{c}_w}]^{c_f}$, $\forall\, e_i \in g(\beta(e_i)=Str)$, $\forall\, \bar{e}_j \in f(\beta(\bar{e}_j)=Str)$, $c_1,\cdots,c_m,\bar{c}_1,\cdots,\bar{c}_w \in [1,?]$, $c_f \supseteq c_g$ and $\forall\, \bar{e}_j^{\bar{c}_j} \in f(j=1,\cdots,w)$(if $\exists\, e_i^{c_i} \in g$ so that $e_i=\bar{e}_j$ then $\bar{c}_j \supseteq c_i$, else $\bar{c}_j=?$), then $\boldsymbol{proj}(g,f) \longrightarrow f$

$H^{g_1}H_1^{g_2}\cdots H_m^{g_2}$ which conforms to $\beta(e)$ into a hedge $H^{g_1}H_1^{g_2}\cdots H^{g_1}H_m^{g_2}$ that conforms to $\beta_1(e)$.

The operations that transform a DTD and its conforming documents into a target schema from its original schema progress in two recognized phases. In the first phase, the XML data is converted into a flat form; and in the second phase, the flat form is converted into the target schema. A flat form is defined by $\beta(e)=[g_1^{c_1}|\cdots|g_n^{c_n}]^c$, where $n\geq 1$ and $\forall i=1,\cdots,n(c_i=1|?\wedge g_i=[e_{i1}^{c_{i1}},\cdots,e_{im_i}^{c_im_i}]), \forall j=1,\cdots,m_i(e_{ij}\in EN \wedge c_{ij}=1|?\wedge\beta(e_{ij})=Str)$. Obviously such a flat form has the minimum layers of brackets to keep the semantics of disjunctions and conjunctions. It can help to simplify the comparison of two data schemas. Here the idea is to respectively convert the original schema $\beta_s(e)$ and the target schema $\beta_t(e)$ into the flat form $\mathcal{B}_s(e)$ and $\mathcal{B}_t(e)$, and then convert $\mathcal{B}_s(e)$ into $\mathcal{B}_t(e)$ through a series of projection operations. Suppose the operation sequence from $\beta_t(e)$ to $\mathcal{B}_t(e)$ is Φ, then carry out the revered sequence Φ^{-1} against $\mathcal{B}_t(e)$ to convert it into an equivalent schema $\beta'_s(e)$. By this process, $\beta_s(e)$ is converted to $\beta'_s(e)$, which agrees with the structure defined in $\beta_t(e)$. Note all the operations in Φ^{-1} are respectively the inverse operations in Φ, e.g., if *min* is executed in Φ, then the corresponding operation is *mout* in Φ^{-1}.

5 IMPLEMENTATION OF THE XML DATA TRANSFORMATION OPERATIONS

5.1 Main Challenges

There are a number of challenges in the development of the XML data transformation operators when issues such as information preservation, nested brackets, and mutually nested conjunction and disjunction are considered. Obviously these issues must be adequately addressed for practical enterprise information integration applications.

It is essential to preserve the semantics of data when carrying out a data transformation. The relationships between data elements must be preserved. For example, before and after a data transformation, it is desired that the *title* and *ISBN* of a book are put under the same element. This derives a requirement that the data transformation operations must be able to be reversed. Although our operators have been defined by carefully considering this requirement, it is very complicated to realize the reverse due to the consideration of the full XML syntax. As an example, let $\beta_1(e)=[g_1^?|g_2]^*$, if we do $min(?,0,[g_1^?|g_2]^*)$, we'll get $\beta_2(e)=[g_1^{??}|g_2^?]^{*\ominus?}$, that is $[g_1^?|g_2^?]^+$because $??=?$ and $*\ominus?=+$. Clearly we cannot go back to the original $\beta_1(e)$ by doing $mout(?,0,[g_1^?|g_2^?]^+)$ because in that case, we attain $\beta_3(e)=[g_1|g_2]^*$. A solution to such problems is not to carry out the multiplicity operation immediately but just keep the information. That means we should store the operation ?? attached to g_1 in the $\beta_2(e)$ in a particular data structure, rather than getting their operation result immediately. This leads to the data structure and the algorithm implementing the operations more complicated.

5.2 Overall Implementation Framework

The overall framework for implementing a data transformation operator can be briefly described as follows. Based on the element e to be operated, the $\beta(e)$

is located from the tuple expression built from a given DTD, and then its output part is updated. This realizes the conversion from $\beta(e)$ to $\beta_1(e)$, the latter will determine the transformed DTD of the operator. The $\beta(e)$ is also provided for the document transformation. The node e in the document is found through parsing the document tree, then all the hedges conforming to $\beta(e)$ under the node are converted into the new ones conforming to $\beta_1(e)$. The conversion requires considerable complicated operations against the hedges, including comparison, group, modification, re-structure, etc.

5.3 The Implementation of Document Transformation

To transform a document from its original schema into a targeted one by using our operators, five tasks must be performed: 1) construct document trees from the given XML file; 2) locate the nodes to be operated in the document trees; 3) identify the hedges requiring conversion under each of the nodes; 4) transform the hedges into a new format according to the operator; and 5) store the modified document trees into an XML file.

Some of the tasks can be accomplished with the help of the XML DOM parser (Maruyama, 2002). It is invoked to traverse the document tree, and when the node e to be operated is found, it transmits all its child trees to a procedure. The latter identifies the hedges that conform to the $\beta(e)$ provided by the DTD transformation. If such a hedge is identified, then task 4 will start and the hedge will be converted according to the definition of the operator. At the last, an XML file is built by storing the new hedges to it, which can be accomplished by the parser.

An algorithm has been developed to identify the hedges that conform to a particular type definition from the child trees of a node. It splits the child trees into logic groups using the hedge notations and then checks their conformation to a given type definition. Its input arguments include: 1) a type definition expression gg to which the child trees conform; 2) a type constructor gg_1 to which the hedges to be identified must conform; 3) the child trees of a node; and 4) an index from which to start the search in the child trees. In the algorithm, gg is used as the rule to parse the child trees, gg_1 is used as the criterion to check if a hedge is the one to be identified.

The algorithm produces two indexes as its output. The elements between them in the child trees form the hedge identified by the algorithm. Due to the space limit, we are not able to provide the algorithm in this paper. The interested readers can contact us to get it.

6 THE OPERATOR PACKAGE AND ITS APPLICATIONS FOR ENTERPRISE INFORMATION INTEGRATION

To apply the proposed methodology to realize enterprise information integration, we have developed *a Complete XML Data Transformation System*. Figure 4 shows its entry interface. From the interface, users can, across the Internet, perform various data transformation to realize enterprise information integration. They can carry out the data transformation in a *step-by-step* mode or ask the system to automatically accomplish a series of transformation operations for them.

Figure 4: The entry interface of the system.

In the following, we will briefly illustrate the applications of the method in practical enterprise information integration via an example. Recently, publications are no longer just the distribution of the printed works, e.g. book or journal. They now include various electronic media, e.g. CD, web, etc. In different publishing companies, the publication information is normally encoded in different XML formats. It thus requires techniques to integrate these information together to provide users with publication information in a unified format. Suppose an enterprise information system adopts the DTD shown in Figure 5 to provide users with publication information. Note the headers and the string type #PCDATA of the DTD are not included in the figure to save space. Clearly in order to provide users with publication information using such a structure, all the relevant publication information encoded in other formats, including the one shown in Figures 2 and 3, must be converted to match the structure.

Our methodology can be used to realize the publication information integration. The basic process is, by

using the data transformation operators, publication information encoded in other format is converted to a flat form, and then further converted into an equivalent structure which conforms to the integrated global schema.

```
<!ELEMENT root (course,ref*)*>
<!ELEMENT ref (book|article|VCD|videoTape)>
<!ELEMENT book (title,author,publisher,ISBN,ISSN?,year)>
<!ELEMENT article (title,author,source+)>
<!ELEMENT VCD (title,publisher,ISSN)>
<!ELEMENT videoTape (title,publisher,ISSN)>
<!ELEMENT source (URL*|conference|journal)>
<!ELEMENT conference (cname,organizer,venue,time,URL?)>
<!ELEMENT journal (jname,Vol,No?,PG?)>
      ......
```

Figure 5: A common global DTD structure.

7 SUMMARY AND FURTHER WORK

This paper has presented a framework for implementing XML data integration via converting XML data from different data sources to an integrated global schema. The representation of DTDs and documents, and the transforming operations of XML data, defined by a set of operators, have been illustrated. The implementation of the transformation operations has been investigated and a package of the operators has been reported. In the proposed method, the full XML syntax has been covered, including nested brackets in element type definitions, multiplicity constraints attached to those nested brackets, and disjunction, for which other work has not provided sufficient support. Using the methodology, the transformed documents always conform to the transformed DTDs, which is a property that is not possessed by any query languages and algebras where users require detailed programming of the transformation procedure. With our work, users are freed from using complex language syntaxes and they just need to combine operators for achieving their desired XML data integration.

Our next work includes the refinement of the operators and the automated XML data conversion using the operators according to a sequence of operations defined based on the mapping between the original and target schema. The latter is also the one we will continue to investigate.

REFERENCES

Boag, S., Chamberlin, D., Fernandez, M. F., Florescu, D., Robie, J., and Simeon, J. (2007). Xquery 1.0: An xml query language. *1st International Conference on Template Production.*

Erwig, M. (2003). Toward the automatic derivation of xml transformations. *LNCS 2814 - ER 2003 Workshops ECOMO, IWCMQ, AOIS, and XSDM Proceedings*, pages 342–354.

Kay, M. (2007). XSL Transformations (XSLT), Version 2.0. *http://www.w3.org/TR/xslt20/.*

Liu, J., Park, H., Vincent, M., and Liu, C. (2006). A formalism of XML Restructuring Operations. *LNCS - ASWC*, pages 342–342.

Maruyama, H. (2002). *XML and Java: developing Web applications*. Addison-Wesley, Boston, Massachusetts.

Tian, J., Liu, J., Pan, W., Vincent1, M., and Liu, C. (2008). Performance Analysis and Improvement for Transformation Operators in XML Data Integration. *APWeb 08*, pages 214–226.

Zamboulis, L. (2004). Xml data integration by graph restructuring. *BNCOD*, pages 57–71.