International Journal of Soft Computing 1 (1) : 41-51, 2006

An Open XML-Based Article Composing System and Its Transformation Model

[1]Shang-Juh Kao and [2]Tseng-Chang Yen
[1]Department of Computer, [2]Department of Applied Mathematics,
National Chung-Hsing University, 250 Kuo-Kuang Road Taichung, Taiwan 402

Abstract: To compose an XML-based article, two source files, the bared article itself and the bibliography database file, have to be efficiently integrated. This integration involves citation from the article file, search on the bibliography database and finally association with the cited references to form the result article. All above operating processes are XML-based. In order to investigate the feasibility of integrated operations, we propose an open XML-based Article Composing System (XACS) and present a mathematical model of the transformation processing. With XACS, an author can freely compose an article with his favorite editors, efficiently utilize the citation service and automatically generate the desired document format, such as PDF, T_EX, or other XML format. Four phases are developed to compose an article in XACS: editing phase, validation phase, association phase and presentation phase. To explore the validity of transformation processing, a formal mathematical model is presented . Although every XML document can be represented by a tree, to compose an article requires more than two XML documents. The corresponding transformation has to be operated over forests. We thus give a forest grammar to model XML documents and a Macro Forest Transducer with equality test (MFT^{ET}) to completely model the transformation processing. Finally, concatenation and composition properties are verified to assert the validity of MFT^{ET}.

Key words: Composing system, transformation, XML

INTRODUCTION

XML[1] uses a set of markup conventions for describing texts and explicitly expresses semantic contents of a document. With these descriptive markups, XML documents are suitable for electronic processing and data reuse. Unlike HTML, XML is extensible: it does not contain a fixed set of markups. Nevertheless, XML documents may have a pre-defined set of syntaxes and could be formally validated. As a consequence, XML is gaining popularity for data representation and data exchange over the Internet in the area of academy, enterprise and various organizations and more and more markups are defined accordingly. It is expected that most information on the web will soon include some XML tags. For instance, most database management systems have already operated data in XML for a variety of applications. One of the most important facts that leads to the overall success of XML is that a series of XML-related standards have been quickly developed, such as Xpath[2], XSL[3], DOM[4], ...etc. This study reports an XML application, an open XML-based article composing system and its formal model of data transformation.

Having a well-managed bibliographic collection scheme and an efficient citation system are valuable in composing an article for research communities. Especially, the continuous growth of information sources and the subsequent increase in the size of bibliographic entries have made bibliography management and citation-referencing the frustrated tasks in composing an article. Despite the importance of bibliography in scientific/technical documents, a total solution for composing an XML-based article is still under development. Current solutions are limited in the sense that they are either application-oriented or word-processor oriented, such as WinBibDb[5] for $LA^{T}_{E}X$ and EndNote[6] for MicroSoft Word. To make citation and referencing easier in producing an article, a number of utilities have been developed in the last decades (e.g., EndNote, ProCite[7] and Biblioscape[8]). Several web-based bibliography management tools, such as RefWorks[9] and zNote[10], were also invented due to the wide accessibility of Internet. Nonetheless, author usually edits his article and bibliography database using his familiar software, no matter which bibliography management system is adopted. Author may also request some extra information to be included in the bibliography database, such as a critical formula or context associated with the related entry. Furthermore, it would be a heavy load for an author to prepare and transfer his article to meet the designated

Corresponding Author: Shang-Juh Kao, Department of Computer Science, National Chung-Hsing University, 250 Kuo-Kuang Road. Taichung, Taiwan 402

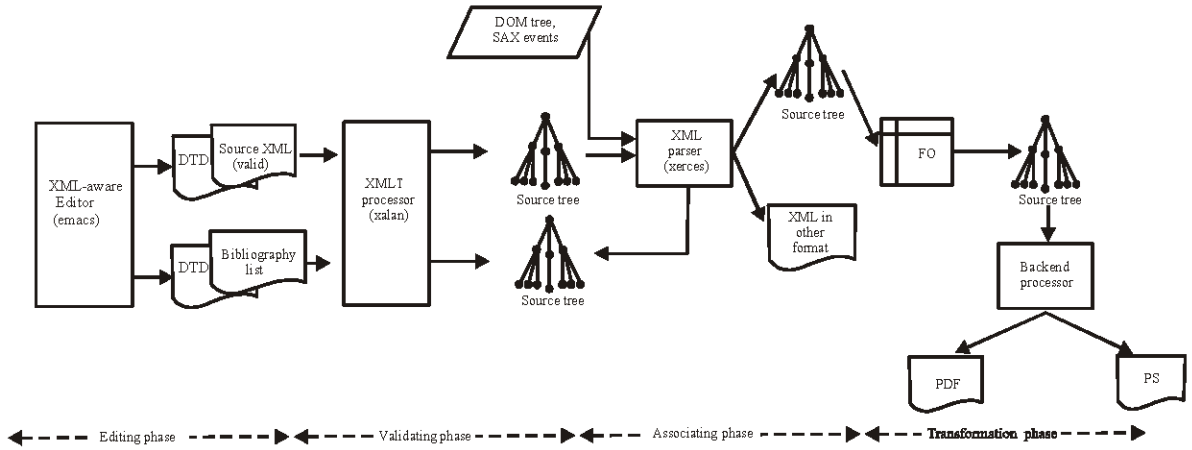

Fig. 1: The operational flow of XACS

format, such as PDF, T_EX, or other XML document format. As a result, an efficient XML-based article composing system should allow an author to freely organize his article and bibliography database as well as to flexibly accommodate the citation management and to meet the various document formats.

Since XML documents may be requested by either human beings or computer applications, transformation processing becomes very critical. The former request needs to format documents following the presentational syntax adapted for various devices, such as printers, phones, E-Book readers, personal digital assistants, etc. CSS[11] is a popular example to convert XML to HTML for Web browsers. The later request may require an intermediate XML transformation. XSLT[12] is such a example which is powerful in tree structure navigation to meet the intermediate processing. The second part of this paper will show how well these transformation operations work by developing a mathematical model for the transformation processing.

As the process of transformation requires searching the data in both source and translation tree, query operation becomes a necessary task in performing the transformation. Other opeartions such as duplication or concatenation are also required in compsoing an article. In an attempt to better describe the operations of transformation, a mathematical model with formal syntax is necessary. Though every XML document can be represented by a tree, XML transformation process is operated over forests. We thus provide a forest grammar to model XML documents and then define a Macro forest Transducer with equality test (MFT^{ET}) to completely model the transformation process in XACS. Two basic properties, namely the concatenation property and composition property are taken into account for the validity test of MFT^{ET}. Through these verification steps, we are sure that MFT^{ET} can effectively represent the thansformation operations of XACS and consequetly, XACS is properly functioning for composing an XML-based article with efficient bibliography reference.

XACS FRAMEWORK

A typical article composing systems usually allows an author to freely include citations in an article and to automatically generate a list of bibliography entries at the end of the article. However, most of them are specially designated for proprietary word processing software. For instances, EndNote, ProCite and Biblioscape were dedicated for Microsoft Word (with little support for other word processors), Refer is used only with UNIX troff and BiBTeX works for L^AT_EX typesetting system.

Coming along with the growing request of XML-formatted article publications, tens of free XML parsers have been developed for checking the well-formedness or validity of XML documents. In most cases, authors edit their articles with their own editing tools and subsequent XML processing is completed by an associated favorite software. In this study, XACS offers the flexibility for authors to edit, validate and automatically generate bibliography list via authors' familiar software. Consequently, the loading of authoring an article can be significantly reduced and the desired output can be flexibly generated. The workflow of XACS is illustrated in Fig. 1. As shown in the figure, composing an XML article in XACS may be carried out through the following sequential phases: editing phase, validation phase, association phase and presentation phase. Each phase is further described in the following:

```
<article>
<body>
<section>
<title>Introduction</title>
<para>Detailed information about XML
can be found on the web <cite id="W3C04"/>.
</para>
<para> Next, we focus on computation by
query automata<cite id="NeSc"/>.
</para>
</section>
</body>
<bib></bib>
</article>
```

Fig. 2: The art.xml file

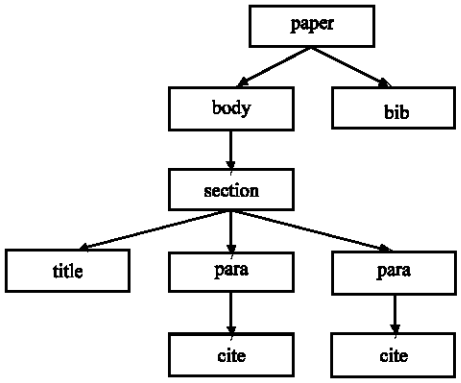

Fig. 3: The tree representation of the XML document in Fig. 2

Editing phase: The first step is to create and edit an article via authors' familiar XML-aware editor, such as emacs and xmlspy[13]. New bibliography entries are inserted into the bibliography database via the same editor in this phase. A well-formed XML document does not need an associated DTD(Document Type Definition) to describe its structure, but a valid XML document with its associated DTD will reveal more information to succeeding processor. Here, we assume that all XML documents are valid in XACS.

Validation phase: Parsing is the most basic operation for XML document processing. After complete the editing, both the XML-based article and bibliography database are submitted to an XML parser and the parser will pass the root of the generated translation tree to the succeeding XSLT processor, such as Xalan[14] for validation purpose.

Association phase: Cited bibliography entries are selected form bibliography database and appended to the article in this phase. Many XML related standards, such as XSLT and XQuery, can carry out this association function. Almost all XML related standards are structured in XML format. DTD is an exception. In fact. both XSLT stylesheet and XQuery file are also written in XML. These XSLT stylesheets or XQuery files are usually static for specific publishers. For instance, authors prepare multiple articles with the same XSLT stylesheet or XQuery file for the same publisher. Concerning about the efficiency, we can compile the XSLT stylesheet or XQuery file as a DOM tree, called translation tree, to avoid repeatedly compilation. After receiving the both roots of source trees and translation tree, the XSLT processor can process the transformation and produce the result XML document in other XML format or the result tree for further processing.

Presentation phase: If we need a dedicated presentation, such as producing the PDF or TeX output, then we need translate the result tree into FO(floating object)[15] tree and invoke a backend program to produce the desired output format. All these tasks are accomplished in the presentation phase.

TREES, FORESTS AND FOREST GRAMMARS

Every XML document can be represented by a tree[16]. A part of formal language theory is concerned with the generation and translation of trees. In particular, tree transducer theory was originally motivated by syntax-directed translation in compilers and has recently been introduced for the XML transformation modeling[17].

The whole transformation process of XACS can be regarded as a typical application of XML document transformation[18]. The -pebble tree transducer and macro tree transducer were recently proposed for XML transformation model[19]. Because XML transformation processing is realized on forests, rather than on trees, we are more interested in forests manipulation. We adopt the macro forest transducer in[20] and extend it to Macro Forest Transducer with equality test to completely model the transformation process of XACS.

DATA MODEL FOR XML DOCUMENTS

XML documents can be represented by tree structure, inner nodes correspond to elements which determine the structure of the document, while the leaf nodes and the attributes provide the content. Sometimes, as in the case of typechecking, we are only interested in the structure of a document, rather than in the actual values of the attributes or the leaf nodes. In such case, we can adequately represent XML documents as trees over

a finite alphabet. Figure 2 shows a sample XML document and Fig. 3 shows its tree representation.

Let $\sum$ be an alphabet. The sets T_Σ of trees t and F_Σ of forests over $\sum$ are defined inductively in the following:

$$t ::= a\langle f\rangle, a \quad \Sigma \qquad f ::= t_1 \ldots t_n, n \geq 0$$

where are trees. Note that there is no priori bound on the number of children of a node in a tree; such trees are therefore *unranked*[21].

FOREST GRAMMARS

A valid XML document is a well-formed XML document, which conforms to the rules of a Document Type Definition(DTD). For each DTD, we can obtain a structurally equivalent extended context-free grammar[22]. Since XML transformation processing is conceptualized on forests, rather than on trees, in the following, a forest grammar over $\sum$ is defined. We use R_Σ to denote the set of regular expressions over $\sum$.

Definition 3.1: A forest grammar over $\sum$ is a tuple $G = (X, r_0, R)$ where X is a set of *variables*, $r_0 \in R_\Sigma$ is the *start expression* and R is a finite set of rules, also called productions, of the form $x \rightarrow a\langle r\rangle$ with $x \in X$, $a \in \sum$ and $r \in R_\Sigma$.

The following example is the forest grammar which generates the XML document in Fig. 2.

Example 3.2: Let G1 = ($\{x_a, x_b, x_s, x_t, x_p, x_c\}$, x_a, R) be a forest grammar with $\sum$ = {a, b, s, t, p, c} and the following rules:

$$x_a \quad a\langle x_b\rangle \qquad x_b \quad b\langle x_s^+\rangle \qquad x_s \quad s\langle x_t x_p^+\rangle$$

$$x_t \quad t\langle\varepsilon\rangle \qquad x_p \quad p\langle x_c\rangle \qquad x_c \quad c\langle\varepsilon\rangle$$

When querying an XML document, the alphabet is chosen as the set of element types occurring in the document. But this does not cover all XML document contents. In addition to elements and child elements, an attribute could be used with an associated character string and an element may also contain character data, i.e., text. In order to deal with character data, we could extend the alphabet set to include the characters and treat each single character as a leaf. Fully representation of all characters is space-consuming and is hard to be implemented efficiently. Besides, when searching for text in a document, one is usually not only interested in a single character but in a whole word, sentence, or even paragraph.

In XACS, keywords of citations are represented as values of id attributes of cite elements, therefore attributes and texts should be explicitly represented in our model as well. It therefore makes sense to treat a segment of characters ,i.e. a string, as a single node. Since XML documents are written in UNICODE [23], the alphabet set must contain the full range of UNICODE characters , which are theoretically more than a million. Thus, the set of trees is now given by

$$t ::= a\langle f\rangle, a \in \sum | "s", s \in U^*$$

In order to match such a text node, we include structural conditions on text by a special mechanism, namely regular expressions over the UNICODE. In order to distinguish them from regular expressions over tree variables, we call these regular expressions as text *patterns*. Consequently, text patterns could be appeared in the right-hand side of a production rule as:

$$X \rightarrow \tau, \tau \in R_u$$

The following example shows the extension of G_1 using text patterns.

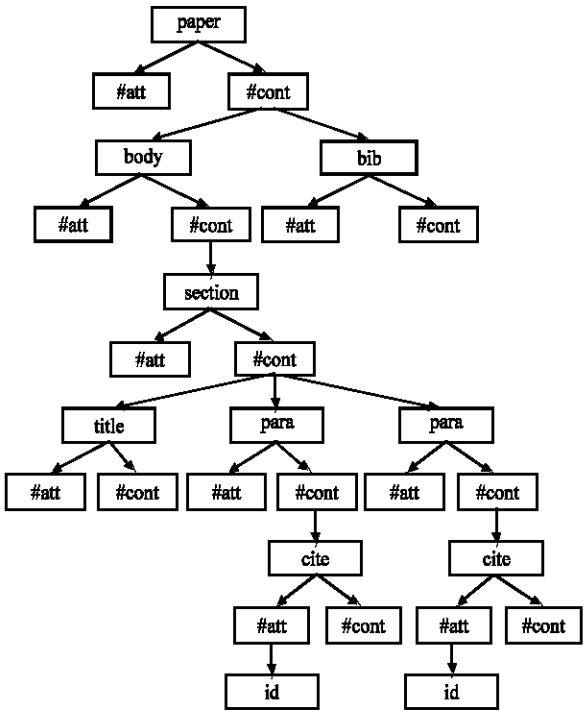

Fig. 4: The extended tree representation of an XML document in Fig. 2

Example 3.3: The forest grammar G_2 is an extension of G_1 defined as follows. $G_2 = (\{x_a, x_b, x_s, x_t, x_p, x_c, x_1\}, x_a, R)$ with $\sum = \{a,b,s,t,p,c\}$ U and the following rules:

$x_a \rightarrow a\langle x_b\rangle$ $\quad x_b \rightarrow b\langle x_x^+\rangle$ $\quad x_s \rightarrow s\langle x_t\, x_p^+\rangle$
$x_t \rightarrow t\langle x_1\rangle$ $\quad x_p \rightarrow p\langle x_c\rangle$ $\quad x_c \rightarrow c\langle x_1\rangle$
$x_1 \rightarrow \tau$ where $\tau \in R_u$

In XML, however, a node of the document tree is not only labeled with its element type, but also with a set of attribute assignments. For example, the citation element <cite id="W3C04"/> is uniquely identified by the id attribute with value "W3C04" (note: attribute value is quoted). One feasible way to deal with attributes is to adopt external predicates. For uniformity, however, we follow a different approach similar to the presentation as in DSSSL[24] and XPATH. That is, the attributes of an element are represented in the document tree as additional child nodes of the element.

In such representation, each element has exactly two children, labeled with auxiliary symbols #atts and #cont. Each #atts subtree contains an attribute assignment. Note that attribute assignments are unordered; therefore they need not appear in the same order as the XML opening tag. The children of the #cont node are the content of the element. This representation requires extension to the alphabet set with the two auxiliary symbols, #atts and #cont and the attributes names. The new representation of the art.xml is illustrated in Fig. 4.

MACRO FOREST TRANSDUCERS WITH EQUALITY TEST

Equality test is necessary for querying/transforming XML documents. We extend the concept of macro forest transducers (mft's) in[20] by the addition of equality test. The syntax and semantics of macro forest transducer with equality test are introduced.

SYNTAX

Let x_1 and x_2 be two input variables and Y be the set of formal parameters y_i, $i \geq 1$

Definition 4.1: A Macro Forest Transducer with equality test (MFT^{ET}) is a tuple
$M = (Q,\sum,\Delta,q_0,R)$ where

- Q is the set of states (or procedures);
- $\sum$ and Δ are finite alphabets with $Q \cap (\sum \cup \Delta) = \phi$, called the input and the output alphabet, respectively;
- $q_0 \in Q$ is the initial state;
- *R* is a finite set of rules of the form:

$p(e,y_1,...,y_n) \quad ^{c} \quad f,$ or

$p(a, x_1\ x_2, y_1 ..., y_n) \quad ^{c} \quad f,$

With $a \in \sum$, $p \in Q$ and c is a Boolean combination of equality constraints. The right-hand sides are of the following form f:

$t :: = b\langle f1\rangle$
$F :: = q(x_i, f_1, ..., f_m) \ |\varepsilon|y_j|tf_1|f_1f_2$

where $q \in Q$, $b \in \Delta$and $I \in \{1,2\}$, $j = 1, ..., n$,$n,m \in N$, $n, m \geq 0$. Moreover, the right-hand sides for empty input forests must not contain occurrences of the variables x_1 or x_2.

Note that a state (or procedure) may differ in its rank, i.e. the number of its accumulating parameters. We use superscript k in $Q^{(k)}$ to denote the number of parameters.

In the association phase of XACS, the input source tree (i.e., the tree representation of art.xml) is first duplicated and then each id attribute value of cite elements is selected as a keyword. In the end, items with the matched keywords are extracted from the bib.xml and are used to generate the output. The following three examples show their corresponding MFT^{ET} s.

Example 4.2: The macro forest transducer md = $(Q,\sum,\sum,q_0,R)$ will duplicate a input forest f over $\sum$, as defined in the following:

- $Q = \{q_0^{(1)}\}$;
- $\sum$ is a finite alphabet;
- R consists of the rules:

$q_0(\varepsilon,y_1) \quad y_1$;

$q_1(b\langle x_1\rangle x_2,y_1) \quad q_2(x_1,y_1,b\langle x_1\rangle)q_1(x_2,\varepsilon)$

$q_0(a\ x_1\ x_2,y_1) \quad a\ q_0(x_1,y_1)\ q_0(x_2,y_1)$ for all $a \in \sum$.

Example 4.3: The macro forest transducer $M_s = (Q, \sum, \sum, q_0, R)$ will select the text child nodes of the *c* nodes from the input forest f over $\sum$, as defined in the following:

- $Q = \{q_0^{(1)}\}$;
- $\sum$ is a finite alphabet; and U the output alphabet;
- R consists of the rules:

$q_0(\varepsilon, y_1)$ y_1;

$q_0(c\ x_1\ x_2, y_1)$ $q_0\ x_2(y_1, x_1)$,

$q_0(a\langle x_1\rangle x_2, y_1)$ $q_0(x_1, y_1)q_0(x_2, y_1)\ \forall a$ Σ, a c

Example 4.4: The macro forest transducer $M_t = (Q, \sum, \sum, q_0, R)$ will select the qualified subtrees (rooted at *b*) from the input forest f over, as defined in the following:

- $Q = \{q_0^{(1)}, q_1^{(1)}, q_2^{(2)}\}$;
- $\sum$ is a finite alphabet,
- R consists of the rules:

$q_0(a\langle x_1\rangle x_2, y_1)$ $q_1(y_1, a\langle x_1\rangle x_2)$ where is initialized as the tree representation of the bibliography file;

$q_1(a\langle x_1\rangle x_2, y_1)$ $q_1(x_1, y_1)q_1(x_2, y_1)\ \forall a$ Σ, a b ;

$q_2(b\langle x_1\rangle x_2, y_1)$ $q_2(x_1, y_1, b\langle x_1\rangle)q_1(x_2, \ )$;

$q_2(a\langle x_1\rangle x_2, y_1, y_2)$ $q_2(x_1, y_1, y_2)\ \forall a$ Σ, a i ;

$q_2(i\langle x_1\rangle x_2, y_1, y_2)$ $x_1 = y_1$ $q_2(\varepsilon, y_1, y_2)$;

$q_2(\varepsilon, y_1, y_2)$ y_2

Semantics of MFT^{ET}: A *macro forest transducer* with *equality test* (MFT^{ET}) is a tuple $M = (Q, \sum, \Delta, q_0, R)$. Each state in can be considered as a procedure which recurs over the first argument while the additional arguments serve as accumulating parameters. A production rule can be applied only if the associated constraint is satisfied. Instead of defining the translation by means of a derivation relation, we adopt the meaning function defined in[20] to describe the transitions of the MFT^{ET}.

Definition 4.5: Let $M = (Q, \sum, \Delta, q_0, R)$ be an arbitrary, but fixed, macro forest transducer with equality test. The meaning of a state $q \in Q$ having $k \geq O$ accumulating parameters is a function:

$$[q]: F_\Sigma \quad (2^{F_\Delta})^k \qquad 2^{F_\Delta}$$

These functions are inductively defined by:

$$[q](a\langle s_1\rangle s_2, S_1, \ldots, S_k): [f_1]\sigma\rho \cup \ldots \cup [f_m]\sigma\rho$$

for $a \in \sum$ where $\sigma(x_i) = s_i$, I=1,2, p $(y_j = S_j)$ for j = 1, ..., k, $a\langle s_1\rangle S_2$, S_1, ... S_k satisfy c and

$q(a\langle x_1\rangle x_2, y_1, \ldots, y_n)$ ^{c} $f_1 | \ldots | f_m$ R and

$[q](\varepsilon, S_1, \ldots, S_k): [f_1]\phi\rho \cup \ldots \cup [f_m]\phi\rho$,

where ϕ is the empty assignment, $p(y_j) = S_j$ for j=1,...,k, ε, S_1, ... S_k satisfy c and

$q(\varepsilon, y_1, \ldots, y_n)$ ^{c} $f_1 | \cdots | f_m$ R .

Here, $[.]\sigma\rho$ denotes the evaluation of a right-hand side expression with respect to the bindings σ, ρ of the formal parameters x_i and y_j, respectively.

$[\varepsilon]\sigma\rho = \{\varepsilon\}$,
$[y_j]\sigma\rho = \rho(y_j)$,
$[b\langle f_1\rangle f_2]\sigma\rho = \{b\langle s_1\rangle s_2 \mid s_i \quad [f_i]\sigma\rho\}$
$[f_1 f_2]\sigma\rho = \{s_1 s_2 \mid s_i \quad [f_i]\sigma\rho\}$,
$[q(x_i, f_1)]\sigma\rho = [q](\sigma(x_i), [f_1]\rho\rho, \ldots, [f_k]\sigma\rho)$

The transformation induced by the transducer $M = (Q, \sum, \Delta, q_0, R)$ is the function $\mu_M(f): F_\Sigma \qquad 2^{F_\Delta}$ induced by the start procedure q_0 with the input forest, f and empty forests in the accumulating parameters:

$$\mu_M(f) = [q_0](f, \{\varepsilon\}, \ldots, \{\varepsilon\}).$$

To explicitly illustrate how meaning functions work, the following three examples show the derivation steps of MFT^{ET}'s which constitute the simplified transformation in the association phase.

Example 4.6: Let M_d be a MFT^{ET} defined in example 4.2 and trees $t_1 = a\langle c\langle\varepsilon\rangle d\langle\varepsilon\rangle\rangle$ and $t2 = b\langle c\langle\varepsilon\rangle d\langle\varepsilon\rangle\rangle$ By taking $f = t_1 t_2$ as the input forest, the derivation steps can be explicitly stated as in the following.

$$\begin{aligned}
\mu_{Md}(f) &= [q_0](a\langle x_1\rangle x_2, y_1) \\
&= [a\langle q_0(x_1, y_1)\rangle q_0(x_2, y_1)]\sigma\rho \\
&= a\langle [q_0](c\langle\varepsilon\rangle d\langle\varepsilon\rangle, \varepsilon)\rangle [q_0](b\langle c\langle\varepsilon\rangle d\langle\varepsilon\rangle\rangle, \varepsilon) \\
&= a\langle c\langle [q_0](\varepsilon, \varepsilon)\rangle [q_0](d\langle\varepsilon\rangle, \varepsilon)\rangle b\langle [q_0](c\langle\varepsilon\rangle d\langle\varepsilon\rangle, \varepsilon)\rangle \\
&= a\langle c\langle\varepsilon\rangle d\langle [q_0](\varepsilon, \varepsilon)\rangle\rangle b\langle c\langle [q_0](\varepsilon, \varepsilon)\rangle [q_0](d\langle\varepsilon\rangle, \varepsilon)\rangle \\
&= a\langle c\langle\varepsilon\rangle d\langle\varepsilon\rangle\rangle b\langle c\langle\varepsilon\rangle d\langle [q_0](\varepsilon, \varepsilon)\rangle\rangle \\
&= a\langle c\langle\varepsilon\rangle d\langle\varepsilon\rangle\rangle b\langle c\langle\varepsilon\rangle d\langle\varepsilon\rangle\rangle
\end{aligned}$$

Example 4.7: Let M_s be a MFT^{ET} defined in example 4.3. By taking $f = a\langle c\langle\tau_1\rangle c\langle\tau_2\rangle\rangle$ as the input forest, the derivation steps can be explicitly stated as in the following.

$$\begin{aligned}\mu_{M_s}(f) &= [q_0](a\langle x_1\rangle x_2, y_1)\\ &= [q_0(x_1, y_1)q_0(x_2, y_1)]\sigma\rho\\ &= [q_0](c\langle\tau_1\rangle c\langle\tau_2\rangle, \varepsilon)\\ &= [q_0](c\langle\tau_2\rangle, \tau_1)\\ &= [q_0](\varepsilon, \tau_1\tau_2)\\ &= \tau_1\tau_2\end{aligned}$$

Example 4.8: Let M_t be a MFT^{ET} defined in example 4.4. By taking $f = \tau$ and $y_1 = a\langle b\langle i\langle\tau\rangle\rangle b\langle i\langle\tau_1\rangle\rangle\rangle$ as the input, the derivation steps can be explicitly stated as in the following.

$$\begin{aligned}\mu_{M_s}(f) &= [q_0](a\langle x_1\rangle x_2, y_1)\\ &= [q_1(y_1, a\langle x_1\rangle x_2)]\sigma\rho\\ &= [q_1](a\langle b\langle i\langle\tau\rangle\rangle b\langle i\langle\tau_1\rangle\rangle\rangle, \tau)\\ &= [q_1](b\langle i\langle\tau\rangle\rangle b\langle i\langle\tau_1\rangle\rangle, \tau)\\ &= [q_2](i\langle\tau\rangle, \tau, b\langle i\langle\tau\rangle\rangle)[q_1](b\langle i\langle\tau_1\rangle\rangle, \tau)\\ &= [q_2](\varepsilon, \tau, b\langle i\langle\tau\rangle\rangle)[q_2](i\langle\tau_1\rangle, \tau, b\langle i\langle\tau_1\rangle\rangle)\\ &= b\langle i\langle\tau\rangle\rangle\end{aligned}$$

Concatenation and Composition Properties: In general, a forest must be traversed several times for transformation in writing an article. In the association phase of XACS, for instance, a traversal is required to duplicate the input forest and another traversal is to extract each id attribute value of cite elements. The MFT^{ET} M_d in example 4.2 and MFT^{ET} M_s in example 4.3 accomplish the previous two traversals respectively. In addition to the duplication and extraction processes, to completely compose an article requires the binding and association between the bared article and the bibliography database file. As a result, MFT^{ET}s must possess the concatenation and composition properties to completely derive a result article.

Macro tree transducers(mtt's)[25] which is originated from macro grammars[26] and are top-down finite state tree transducers equipped with parameters to accumulate auxiliary results. At each computation step, the inclusion of parameters is helpful in building up the output. The advantage of adopting macro tree transducers is that they are well-understood and have a rich set of theories available[26]. The major deficiency, though, is that, due to tree encoding, concatenation of intermediate trees may not be a tree again. Macro forest transducers(mft's), however, are closed under the concatenation. The satisfaction of the concatenation property of mft's has been proved in[20]. Enabling with the concatenation property, the computing power of mft's is significantly improved.

Macro forest transducers support concatenation of forests as basic operation and thus are appropriate for XML data model. In order to model the predicates (such as @id=$id of <xsl:copy-of select="$bib/bib/bibitem [@id=$id]/child::*" />) in the XSL expression, in this paper, we extend mft's by adding the equality test in production rules. By such addition, all the related properties, such as termination, totality and the emptiness, of MFT^{ET}s are still valid as inherited from mft's, which were reported in[20], with the enhancement in the computing power. Since the satisfaction of an equality test $\pi = \pi'$ can be completed in $O(|\pi|+|\pi'|)$, therefore the time complexity of the translations of MFT^{ET}s meets the corresponding complexity bounds for mft's.

As of the composition property, we have to be sure that the model is still valid after the search, extraction and composition between both the article and bibliography files. Suppose that the class of all transformations which can be realized by MFT^{ET}s is denoted by FMAC. Let $\circ$ denote the composition operation. The following theorem shows that MFT^{ET}s have the composition property, which can be derived straightly from Corollary 4.10 of[25] and Theorem 8 of[23].

Theorem 4.9: $FMAC \circ FMAC \subseteq FMAC$.

Remark: Consider $f \in F_\Sigma$ as the input forest and MFT^{ET}'s defined in examples 4.2, 4.3 and 4.4. The overall transformation of the association phase can be modelled as $\mu_{Md}(f)\mu_{Mt} \circ \mu_{Ms}(f)$.

IMPLEMENTATION AND AN APPLICATION SCENARIO

In the followings, we will present an XACS implementation by adopting well-known XML-related software for each phase in the framework. As an open framework, however, XACS can be implemented using different software tools.

Query and Transformation: As an example of XML applications, in XACS, an examination of the bibliography file and reconstruction of the article by appending the referenced bibliography entries are imperative. That is, query and concatenation processing of XML files are necessary. Even though XML query language has not been standardized yet, XSLT seems to be a candidate at the moment. In Xacs, we adopt XSLT for transformation purpose. XSLT is a part of XSL. XSL is a recommendation for the XML extensible stylesheet language and it contains three components: XSLT (XSL: Transformations) for transformation, Xpath (XML Path

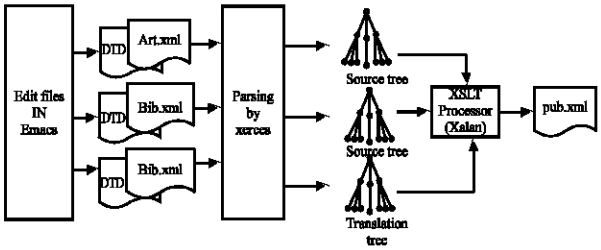

Fig. 5: An implementation for XACS

Language) for addressing and FO (XSL Formatting Objects) for presentation. XSLT is an XML-based language designed to transform one XML document into another XML document or another structured document. It uses the XML Path Language (XPath) to perform queries on an XML document in order to specify a particular part of the document.

Recently, various extensions were added to XSL. The most significant ones are the addition of *variables* and *parameter passing* between template rules. These additions together with the use of modes, which are named states when used in finite state machines and were already defined in earlier versions of XSL, make XSL a powerful query language[28]. For generating the result article, in XACS, these additional features are helpful in processing both the raw article and bibliography file. Hence, we adopt XSLT for transformation purpose, such as duplication, query, concatenation and so on.

Implementation Environment: To complete the procedures of composing an article in XACS, we need first parsing the target article XML file(art.xml in this paper), bibliography database file(bib.xml in this paper) and the XSL file (if necessary bib.xsl in this paper) and activate the translation processor to produce an result file(pub.xml in this paper). An XML parser, Xerces, first checks whether an XML document is well-formed, then builds a source tree according to its data model.

The source trees generated by Xerces are transferred to an XSLT processor. We adopt Xalan from *Apache project* as XSLT processor, since it is efficient and an open source software. XSLT processor plays a major role in whole transformation process and can accept various types of input sources, such as a URL, an XML stream, a DOM tree, SAX Events, or a proprietary data structure. No parsing is needed when the XSLT processor accepts a DOM tree as its input source, since a DOM tree is already a valid result of XML parsers.

For the demonstration purpose, we illustrate an XACS application using Fedora Core 2 system with Pentium 3.0 GHz CPU and 512 MB memory. Xerces-2.6.0[29] and Xalan-1.9.0[14] from the Apache project are chosen as XML parser and XSLT processor respectively. Figure 5 shows the operational flow of the implementation in an XACS application. All example files used in the demonstration including the raw article, the bibliography database file, bibliography stylesheet and result output are included in Fig. A1 through Fig. A4 in Appendix.

Programming Tips: Without join and skolem functions, XSLT lacks some basic property a query language should have. However, the addition of variables and parameter passing between template rules after XSLT version 1.0, together with the use of modes render XSLT into a powerful query language. XSLT variables are similar to variables in other programming languages with the major exception that they may only be assigned a value once at the time they are declared.

The <xsl:variable> element is used to declare a variable. The variable is global if it's declared as a top-level element and local if it's declared within a template. There are two attributes in <xsl:variable>, namely a required "*name*" attribute which specifies the name of the variable and an optional "*select*" attribute which defines the value of the variable. There are also two ways to define the value of a variable: by the *select* attribute or by embedding within the content of <xsl:variable>. If the *select* attribute is presented, the <xsl:variable> cannot contain any content. The<xsl:variable> can be used to store the result of a complex expression, expressions that are repeatedly referenced, the content of some node, or a tree fragment. In our implementation, we apply the <xsl:variable> element twice: first of all we apply <xsl:variable name="bib" select="document('bib.xml')"/> element to store the source tree generated by XML parser Xerces from source article file art.xml. Secondly, we apply <xsl:variable name="id"> element to store the value of id attribute of each cite element. Then, we can perform the tree search for element "cite" by the <xsl:for-each select="//cite"> element and add the bibliography entry by the <xsl:copy-of select = "$bib/bib/bibitem[@id = $id] /child::*" /> element.

As XSLT document is parsed, a transformation tree is generated. The root of the tree is passed to Xalan before Xalan can conduct the transformation process. In general, the automatic bibliography generation procedure is fixed, thus we can save the translation tree in memory for the sake of efficiency. The transformation tree is saved in memory other than in disk, partly because bringing it back from disk takes longer than recompiling the origin, which is largely due to its increased size. We can reuse the tree so long as it remains in memory. It is quite common in a server environment to use the same stylesheet repeatedly to transform documents. To

accomplish this, the compiled stylesheet is set to be strictly read-only at execution time, so it is allowed to be used in multiple executive threads simultaneously.

CONCLUSIONS

Knowledge is power and, bibliography database is the clue of knowledge. Research communities highly request for having an accurate and well-managed bibliographic database in composing an article. However, the continuous growth of information sources and the subsequent increase in the size of bibliography database have made bibliography management one of the most frustrated tasks researchers must face. The XACS is emerging as an evolution of XML applications. It makes easier the composing task as to include the citation-reference service and various presentation purpose. In this paper, we have presented how the XACS provides a general solution for efficient bibliography management in writing an article. It exploits the power of XML to provide a framework for composing an article in which authors can directly use favorite editors to compose articles, efficiently invoke the citation retrieval service and automatically generate the desired document format, such as PDF, TeX, or other XML format. From the user's perspective, this avoids the need to have different collections of word processors, as well as the need for managing the bibliography referencing. Besides, in an attempt to explore the transformation process of XACS, we also provide a forest grammar and macro forest transducer, MFT^{ET}, to describe the operations of XML documents. With MFT^{ET}, XML transformation process in XACS can be explicitly expressed. Two basic properties of a transducer, namely the concatenation property and the composition property are also verified. Through these verification steps, we are sure that MFT^{ET} can effectively represent the transformation operations of XACS and consequently, XACS is properly functioning for composing an XML-based article with efficient bibliography reference.

Several enhancements can be done in future XACS study. As in the editing phase, checking for the completeness and repetition for bibliography entries are ignored in our system. However, most common XML-aware editors can accomplish the examination without a great deal of modification. According to the wide accessibility of Internet, the article file and the bibliography database may reside in different machines or even different platforms. Distributed location requires extra communication efforts in the validation phase. In other words, by setting up a servlet to use XSLT processor (such as Xalan-Java[30]) to respond to requests for automatic bibliography generation, the distributed feature should be taken into account in the validation phase. Extending the association phase with extra XSLT stylesheets to make use XSLT as a query language, then authors can query the bibliography database with respect to the elements, such as type, author, title, year and so forth. Furthermore, since macro forest transducers support concatenation of forests as their basic operation, they are appropriate for modeling XML transformation. Future work may include using macro forest transducers to describe XML query languages, such as XQuery, XQL, .. etc.

APPENDIX

```
<body>
<section>
<title>Introduction</title>
<para> Detailed information about XML can be found on
the
web <cite id="W3C04"/>.
</para>
<para> Next, we focus on computation by query
automata<cite id="NeSc"/>.
</para>
</section>
</body>
```

Fig. A1: The art.xml file

```
<bib>
<bibitem id="W3C04">
<author>W3C</author>
<title>Extensible Markup Language (XML) 1.1</title>
<pub>http://www.w3.org/TR/2004/REC-xml11-
20040204</pub>
<year>2004</year>
</bibitem>
<bibitem id="NeSc">
<author>F. Neven, T. Schwentick</author>
<title>Query Automata on finite trees</title>
<pub>Theoretical Computer Science</pub>
<vol>275</vol>
<spage>633</spage>
<epage>674</epage>
<year>2002</year>
</bibitem>
</bib>
```

Fig. A2: The bib.xml file

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl=
"http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="xml"/>
<!--load the merge file -->
<xsl:variable name="bib" select="document('bib.xml')"/>
<!-- combine the files -->
<xsl:template match="/">
<paper>
<xsl:copy-of select="."/>
<bibliography>
<xsl:for-each select="//cite">
<xsl:variable name="id">
<xsl:value-of select="@id"/>
</xsl:variable>
<xsl:copy-of select="$bib/bib/bibitem[@id=$id]/child::*"
/>
</xsl:for-each>
</bibliography>
</paper>
</xsl:template>
</xsl:stylesheet>
```

Fig. A3: The bib.xsl file

```
<?xml version="1.0" encoding="UTF-8"?>
<paper>
<body>
<section>
<title>Introduction</title>
<para> Detailed information about XML can be found on
the
web <cite id="W3C04"/>.
</para>
<para> Next, we focus on computation by query
automata<cite id="NeSc"/>.
</para>
</section>
</body>
<bibliography>
<bibitem>
<author>W3C</author><title>Extensible Markup
Language (XML) 1.1</title>
<pub>http://www.w3.org/TR/2004/REC-xml11-
20040204</pub><year>2004</year>
</bibitem>
<bibitem><author>F. Neven, T. Schwentick</author>
<title>Query Automata on finite trees</title>
<pub>Theoretical Computer
Science</pub><vol>275</vol>
<spage>633</spage><epage>674</epage><year>2002<
/year>
</bibitem>
</bibliography>
</paper>
```

Fig. A4: The pub.xml file

REFERENCES

1. Bray, T., P. Jean and C.M. Sperberg, 2004. McQueen, editors. Extensible Markup Language (XML) 1.1. W3C Recommendation, World Wide Web Consortium. Available online at org/TR/2004/REC-xml11-20040204.
2. Clark J. and Steve DeRose, 1990. XML Path Language (XPath) Version 1.0. W3C Recommendation, World Wide Web Consortium. Available online at http://www.w3.org/TR/xpath.
3. Berglund, A., 2005. Extensible Style Language (*XSL*) Specification. W3C Working Draft, World Wide Web Consortium. Available online at http://www.w3.org/TR/xsl11.
4. Vi, A., B. Steve and C. Mike *et al*., 1998. Document Object Model (DOM) Level 1 Specification, Version 1.0. W3C Recommendation, World Wide Web Consortium. Available online at http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001.
5. WinBibDb: http://www.mackichan.com/index.html?bibdb/default.htm~mainFrame.
6. EndNote: http://www.endnote.com.
7. ProCite: http://www.procite.com.
8. Biblioscape:http://www.biblioscape.com/biblioscape.htm.
9. RefWorks: http://www.refworks.com.
10. zNote:http://www.zope.org/Members/JMaxwell/zNote.
11. World Wide Web, 1996. Consortium. Cascading Style Sheets level 1. W3C Recommendation, World Wide Web Consortium. Available online at http://www.w3.org/TR/CSS1.
12. Clark, J., 1999. XSL Transformations (XSLT) *Version 1.0*. W3C Recommendation, World Wide Web Consortium. Available online at http://www.w3.org/TR/xslt.
13. xmlspy: http://www.altova.com/products_ide.html.
14. Neumann, A., 1999. Parsing and querying XML documents in SML, PhD thesis, Universitat Trier.
14. Apache project. http://xml.apache.org/xalan-c/index.html
15. World Wide Web Consortium. *XSL Formatting Objects*. http://www.w3.org/Style/XSL
16. Neven, F., 2002. Automata theory for XML researchers. Sigmod Record, 31.
17. Maneth, S., 2004. Models of Tree Translation, PhD thesis, Universiteit Leiden.
18. Tseng-Chang, Y. and S.J. Kao, 2003. XML Document Transformations: A Practical Approach. The J. Three Dimensional Images, 17: 129-136.
19. Engelfriet, J. and S. Maneth, 2003. A comparison of pebble tree transducers with macro tree transducers, Acta Informatica, 39: 613-698.

20. Perst T. and H. Seidl, 2004. Macro Forest Transducers. Information Processing Letters, 89: 141-149.
21. Neven, F., 2001. Automata, Logic and XML. CSL.
22. Kilpelainen P. and D. Wood, 2001. SGML and XML document grammars and exceptions. Information and Computation, 169: V230-V251.
23. The Unicode Consortium, 1996. The Unicode Standard, Version 2.0. AddisonWesley Developers Press, Reading, Massachusetts.
24. International Organization for Standardization, 1996. Document Style Semantics and Specification Language (DSSSL). Ref. No. ISO/IEC, 10179:(E).
25. Engelfriet J. and H. Vogler, 1985. Macro tree transducers, J. Comput. System Sci., 31: V71-V146.
26. Fischer, M.J., 1968. Grammars with macro-like productions, PhD thesis, Harvard University, Cambridge, MA.
27. Fülöp, Z. and H. Vogler, 1998. Syntax-Directed Semantics; Formal Models Based on Tree Transducers, Springer-Verlag, Berlin.
28. Bex, J.G., 2002. Sebastian maneth, Frank neven, A formal model for an expressive fragment of XSLT. Information Systems, 27: 21-39.
29. Apache project. http://xml.apache.org/xerces-c/index.html
30. Apache project. http://xml.apache.org/xalan-j/index.html