Automating XML Document Structure Transformations

Paula Leinonen
University of Kuopio
Department of Computer Science
P.O.B 1627
FIN-70211 Kuopio
leinonen@cs.uku.fi

ABSTRACT

This paper describes an implementation for syntax-directed transformation of XML documents from one structure to another. The system is based on the method which we have introduced in our earlier work. That work characterized certain general conditions under which a semi-automatic transformation is possible.

The system generates semi-automatically a transformation between two structures of the same document class. The system gets source and target DTDs as an input. There is a tool for a user to define a label association between the elements of the DTDs. From the two DTDs and from the label association, the system generates the transformation specification semi-automatically. The system has a tool to help the user to select a correct translation if the target DTD produces several possible structures.

Implementation of the transformation is based on the top-down tree transducer. From the transformation specification the system produces an XSLT script automatically.

Categories and Subject Descriptors

F.3.4 [**Mathematical Logic and Formal Languages**]: Formal Languages—*classes defined by grammars or automata, grammars and other rewriting systems*; I.7 [**Computing Methodologies**]: Document and Text Processing

General Terms

Documentation, Languages

Keywords

Document Structure Transformation, XML, XSLT

1. INTRODUCTION

XSLT [10], a recommendation of the World Wide Web Consortium is a powerful language for transforming XML documents [11] and it has several implementations. Its drawback is that every time when a structure of XML document should be transformed to follow some other structure definition, a new XSLT program has to be written. In addition to XSLT, many other languages and approaches for formatting and transforming documents have been introduced [2, 3, 6]. For avoiding writing a new program for each new transformation needed, some simpler definition languages and visualizing tools have been developed [9, 8, 5].

This paper describes a prototype for an implementation based on the method introduced in [7]. In our previous work, we sketched the semi-automatic method for so called local, hierarchical and dense tree transformation. The method defines a general transformation between two structures of the same document class. Documents which belong to the same document class normally have nearly the same elements so that it is reasonable to try to generate the automatic transformation. The transformation is based on the tree transducer [4].

In Section 2, we present some definitions related to our method. Section 3 describes the implementation for XML documents. Section 4 concludes the article.

2. HIERARCHICAL, LOCAL AND DENSE TRANSFORMATIONS

This section defines basic notions of the method. More strict definitions for the theoretical basis of the method and the algorithms to which the implementation is based on, can be found in [7].

Structure definition of an XML can be given as a *document type definition (DTD)* that is an extended context-free grammar in which the right-hand sides of productions are extended and restricted regular expressions. An *extended context-free grammar* (ECFG) is a tuple $G = (N, \Sigma, P, S)$ where N is a finite set of nonterminal symbols, Σ is a finite set of terminal symbols, disjoint from N and P is a finite set of production schemata and the nonterminal S is the sentence symbol.

Each production schema is of the form $A \rightarrow E$, where A is a nonterminal and E is a regular expression over $V = N \cup \Sigma$. The language $L(G)$ defined by an ECFG G is $L(G) = \{w \in V^* | S \Rightarrow^* w\}$. When $u' = u_1 A u_2 \in V^*$, $A \rightarrow E \in P$ and $v \in L(E)$, we say that the string $u_1 v u_2$ can be *derived* from the string u' and denote the derivation by $u' \Rightarrow u_1 v u_2$.

Derivations of a CFG can be represented in graphical form as a tree. A *derivation tree* [1] for a CFG is an ordered, labeled tree where the root node is labeled by the start symbol, the leaf nodes are labeled by terminal symbols or by an empty string denoted by ϵ and each interior node is labeled by a nonterminal. If A is a nonterminal labeling some interior node and $X_1, X_2, \ldots, X_n$ are the labels of the children of that node from left to right, then $A \rightarrow X_1X_2 \ldots X_n$ is a rule. Here, $X_1, X_2, \ldots, X_n$ stand for a symbol that is either a terminal or a nonterminal. As a special case, if $A \rightarrow \epsilon$ then a node labeled A has a single child labeled ϵ.

A local, hierarchical and dense structure transformation of documents can be implemented using a tree transducer [4]. A *finite state tree transducer* $M = (Q, V_1, V_2, X, q_0, \delta)$ consists of a finite set Q of *states*, a ranked *input alphabet* V_1, a ranked *output alphabet* V_2, a set of variables X, the *initial state* $q_0 \in Q$, and rules of the form

$$q : t(X_1, \ldots, X_m) \rightarrow t'(q_1 : X_{i_1}, \ldots, q_n : X_{i_n})$$

replacing a tree with state q at root and $X_1, \ldots, X_m$ at frontier by a tree with $X_{i_1}, \ldots, X_{i_n}$ and states $q_1, \ldots, q_n$ at frontier, where $X_{i_j} \in \{X_1, \ldots, X_m\}$.

The whole transformation is obtained by starting at initial state at root and applying the rules recursively until leaves are reached at final state. For exact definitions, see [7].

We are interested in *transformations* from the set of derivation trees of the grammar G_1 to the set of derivation trees of the grammar G_2.

The concept of label association defines a relation of "corresponding" elements between the source and target grammars. For the alphabets V_1 and V_2, a *label association* is a relation in $\lambda \subseteq V_1 \times V_2$. The associated labels of nonterminals must be labels of the same type, like terminating nonterminals or repeating nonterminals.

Consider two grammars G_1 and G_2 and a label association λ from V_1 to V_2 between these grammars. A *node association* is a relation ν between the nodes of a derivation tree t_1 by grammar G_1 and a derivation tree t_2 by grammar G_2 such that the labels of related nodes are associated in the label association λ.

We say that a tree transformation τ is *hierarchical* with respect to a label association λ, if for any $t_2 \in \tau(t_1)$, there is a node association ν respecting λ such that whenever $(x, u) \in \nu$, $(y, v) \in \nu$, y being a descendant of x implies v being a descendant of u.

The tree transformation τ is *(c,d)–local* (or *local*) for constants c and d and for label association λ, if there is a node association ν such that whenever the distance of nodes x and y in the path from the root to a leaf in t_1 is less than c and $(x, u) \in \nu, (y, v) \in \nu$, the distance of u and v in t_2 is less than d.

The tree transformation τ is *e-dense* (or *dense*) with respect to constant e and label association λ, if whenever a node has label X associated in λ (i.e. having Y such that $(X, Y) \in \lambda$), it has a descendant associated in λ within distance e, or it does not have any associated descendants.

Having a local tree transformation guarantees that a finite transformation can be defined. A hierarchical tree transformation can be built top-down. Denseness means that transformable elements are so closely situated that the next transformable part can be found by a finite lookahead.

3. IMPLEMENTATION FOR XML

This section introduces a research prototype of the method for automating a local, hierarchical and dense structure transformation of documents. For transforming one document we need to define a transformation specification by going through the phases of the method. After the user has defined the transformation once, the same specification can be used to transform all documents with the source DTD to follow the target DTD.

As an input, the system gets the source DTD for defining the structure of the source document and the target DTD for defining the structure of the target document. At first, the system generates syntax trees from the two DTDs. An internal representation form of the syntax trees in the system is XML.

3.1 Tool for the label association

The user interface for defining a label association shows the syntax trees of the source and the target DTDs in an illustrative form for the user (Fig. 1). The user only defines the matching between the elements which are in the leaves of syntax trees and contain the text of the document. In this paper, we call these elements *terminating elements*. When we have a label association for the terminating elements, the associations for the interior elements in a tree can be generated automatically.

3.2 Tool for the node association

For each interior element of the DTD, the system produces the set of associated terminating elements, what the element can derive. This information is needed in the transformation method, because in two DTDs we consider elements, which derive the same terminating elements, to have the same semantic meaning and associate them. Using sets of associated terminating elements, the system can generate the label association of the interior nonterminals automatically.

From the complete label association, the system automatically produces a node association between the derivation trees of DTDs and generates the substructure pairs. From the structures in Fig. 1, the system generates a substructure pair

```
<rule>
   <input>
      <article>
         <title type='string'/>
         <author repeat='*' type='string' minOccurs='0'
                                 maxOccurs='unbounded'/>
         <date type='string'/>
         <content>
            <abstract type='string'/>
            <section repeat='+' minOccurs='1'
                              maxOccurs='unbounded'/>
         </content>
      </article>
   </input>
   <output>
      <article assoc='article'>
         <writer repeat='*' type='string' minOccurs='0'
                     maxOccurs='unbounded' assoc='author'/>
         <title type='string' assoc='title'/>
         <intro>
            <abstract type='string' assoc='abstract'/>
            <keywords type='string'/>
         </intro>
         <body>
            <section repeat='+' minOccurs='1'
                 maxOccurs='unbounded' assoc='section'/>
         </body>
      </article>
   </output>
</rule>
```

The target DTD may generate several possible target substructures. The system uses a heuristic function to restrict

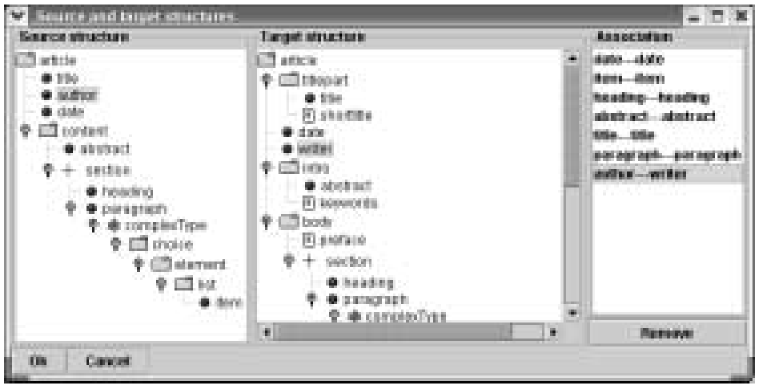

Figure 1: Association of elements `author` and `writer` in a label association

the number of possible translations and to give priorities for the alternative translations. More detailed description about heuristic function can be found in [7]. The user makes the final selection of the matching target substructure. As an output, this phase will produce a transformation specification.

3.3 Generation of the XSLT script

From the transformation specification, the system generates an XSLT script automatically. For each rule in the automatically generated XML rule file, one template will be created. The root node of the input part of the rule will be the node where the template will be matched in the source document. The contents of the output element of the rule defines the contents of the XSLT template.

For the automatically generated example rule above, the system will generate the following template.

```
<xsl:template match='article'>
   <article>
      <xsl:for-each select='./descendant::author'>
         <xsl:apply-templates select='.'/>
      </xsl:for-each>
      <xsl:apply-templates select='./descendant::title'/>
      <intro>
         <xsl:apply-templates select='./descendant::abstract'/>
         <keywords></keywords>
      </intro>
      <body>
         <xsl:for-each select='./descendant::section'>
            <xsl:apply-templates select='.'/>
         </xsl:for-each>
      </body>
   </article>
</xsl:template>
```

Because the system matches the interior elements of the derivation trees, the templates are small when the structures of the documents are near each other. Templates are generated in a recursive manner. From these reasons, the automatically generated XSLT script is easy to understand and tailor by an expert user if needed.

4. CONCLUSIONS

We have introduced a prototype which automates the definition of the document structure transformation. The aim has been to find how far it is possible to automate transformations and develop a method for generating a transformation specification as automatically as possible. The user of the system still needs to have some knowledge about document structures to define a transformation.

5. REFERENCES

[1] A. Aho, R. Sethi, and J. Ullman. *Compilers Principles, Techniques, and Tools*. Addison-Wesley Publishing Company, 1986.

[2] A. Berlea and H. Seidl. Transforming XML documents using fxt. *Computing and Information Technology, Special Issue on Domain-Specific Languages*, 2002.

[3] P. Cimprich, O. Becker, C. Nentwich, H. Jiroušek, M. Kay, P. Brown, M. Batsis, T. Kaiser, P. Hlavnička, N. Matsakis, C. Dolph, and N. Wiechmann. Streaming transformations for xml (stx) Version 1.0. `http://stx.sourceforge.net/documents/`, Working Draft 5 May 2003.

[4] F. Gécseg and M. Steinby. *Tree Automata*. Académiai Kiadó, Budapest, 1984.

[5] Induslogic. XSLWiz, 2001. `http://www.induslogic.com/products/xslwiz.html`.

[6] S. Krishnamurthi, K. Gray, and P. Graunke. Transformation-by-example for XML. In E. Pontelli and V. Santos Costa, editors, *Proceedings of the Second International Workshop on Practical Aspects of Declarative Languages (PADL'00)*, Lecture Notes in Computer Science, Vol. 1753, pages 249–262, Springer-Verlag, Boston, MA, USA, 2000.

[7] E. Kuikka, P. Leinonen, and M. Penttonen. Towards automating of document structure transformations. In R. Furuta, J. I. Maletic, and E. Munson, editors, *Proceedings of the 2002 ACM Symposium on Document Engineering*, pages 103–110, McLean, Virginia, USA, 2002.

[8] E. Pietriga, J.-Y. Vion-Dury, and V. Quint. Vxt: a visual approach to xml transformations. In E. Munson, editor, *Proceedings of the ACM Symposium on Document Engineering 2001 (DocEng '01)*, pages 1–10, Atlanta, 2001.

[9] X. Tang and F. Tompa. Specifying transformations for structured documents. In *Proceedings of 4th International Workshop on the Web and Databases (WebDB'2001)*, pages 67–72, 2001.

[10] W3C. *XSL Transformations XSLT Version 1.0, W3C Recommendation*, November 16, 1999. Available at `http://www.w3.org/TR/xslt`.

[11] W3C. *Extensible Markup Language (XML) 1.0 (Second Edition), W3C Recommendation*, October 6, 2000. Available at `http://www.w3.org/TR/REC-xml`.