ELECTRONIC PUBLISHING, VOL. 8(4), 319–341 (DECEMBER 191995)

Transformation of structured documents

E. KUIKKA[1]

Department of Computer Science and Applied Mathematics
University of Kuopio
P.O.Box 1627, 70211 Kuopio, Finland

M. PENTTONEN

Department of Computer Science
University of Joensuu
P.O.Box 101, 80101 Joensuu, Finland

SUMMARY

Many documents have a definable structure. Some document formatting systems, like the LaTeX formatter, use a structural notation. In recent years the general mark-up language SGML has gained popularity.

In this work we study the transformation of a structure to another. For example, technical journals have their structure definitions, and an article originally written for one journal must be restructured before it can be submitted to another journal. We assume that structure definitions are grammatical, and study what kind of transformations can be automatized or at least semiautomatized.

We took a collection of computer science journals and compared their structure definitions. We classified differences as simple, local and global. As transformation techniques we studied syntax directed translation schemata and tree transducers. Our conclusion was that simple and local transformations can be automatized or semiautomatized, depending whether additional information is not needed, while global transformations are difficult to automatize.

Transformations were tested in our prototype syntax-directed document processing system. The system has one module for editing a document under one structure definition, and another module for changing a document from one structure definition to another.

KEY WORDS Structured document Tree transformation Context-free grammar Parse tree

INTRODUCTION

One motivation for electronic text processing is avoiding retyping of documents. If the reuse of a document requires only updating the content and modifying the layouts of portions of the text, the current systems, especially those that process documents in a structured way, can manage these alterations very well. If the new representation of a document deviates significantly from the original representation and if the required modifications are to be applied to a set of documents, most systems offer only restricted possibilities, and usually the author has to use several systems to achieve the desired result.

We consider electronic document processing as transformations between different representations of documents. Such representations may be seen on the computer screen for creating, updating or browsing the content, or on paper representing a specific form for a document. It can also be an internal representation in a computer memory. Similarly, various representations are external representations for storing documents or exchanging

[1] This work was done during the author's stay in the University of Waterloo in Canada.

CCC 0894–3982/1995/040319–23

Received 4 December 1995
Revised 22 July 1996

documents between different environments. A document processing system must work between all the different representations. In this work all these representations are considered as trees and transformations are methods where one can change one tree to another.

Changes which do not modify the hierarchic structure of documents are in the literature often called *representation modifications*. In such transformations only the layout of a document is changed. Changes of hierarchic structures of documents can be of two types [1]. In *dynamic* modifications, the structure of a document instance is modified without any change in its structure definition. Typically, dynamic transformations happen when a structured document is edited in an interactive text processing system [2,3]. In *static* modifications, the structure definition of a document has been modified and document instances according to an old definition should be changed according to a new definition. Static transformations are needed in situations when the structure definition is no longer valid or it needs to be developed, for some reason. Dynamic transformations consider parts of single documents. Static transformations are usually applied to the whole document. Representation modifications are typical in both cases. We consider static transformations and apply them to changes of the structure and representation.

There are many systems and languages that carry out transformations between different representations of structured documents to be found in the literature. Some of them use document structure definitions to define the transformation [1, 4–7] others use actual document structures [8–11]. There are also translators available for representations of particular formatters and/or text processing systems (some are described in [12]). On the other hand, before structure transformations have been studied concerning documents, the same problem has been solved, for example, in syntax-directed programming environments [13–15] and in databases [16], nowadays especially in object-oriented databases [17–20].

Trees are very common data structures in many applications and techniques. Tree transformations are defined in theories for graphs, trees and terms and their languages [21–26], in the research of tree pattern matching and replacement methods in many application areas [27–29] as well as in tree editors [30]. The aim of this work was to select or develop tree transformation methods which have the following features:

1. definitions for translations can be made either automatically from the structure definitions for trees or at least interactively with the help of the user,
2. translators can be generated automatically from the definition of a translation,
3. translators are applied to parse trees for one grammar and produce parse trees for another grammar accompanied by its grammar, and
4. translations are automatic or at least interactive needing help from the user.

Methods will be applied to documents represented as trees. Thus, first, instead of trying to build special software for each framework for transformations we would like to find some typical differences which occur between tree-structured document representations. And, second, instead of using ad-hoc implementations for different types of transformations, we will study the possibility of tree transformation methods to manage various types of differences.

The rest of the paper is organized as follows. The next section presents a classification for the differences in the structure methodology for trees. After that, the next two sections briefly describe transformation of documents in each difference class. The following section presents our model for a transformation system which has facilities for various

transformations. Subsequently, some similar works are compared to our model. The last section of the paper presents our conclusions.

A CLASSIFICATION OF DIFFERENCES BETWEEN TREES

This section presents a set of categories to classify structure differences in a pair of trees. However, before formally defining different classes, we first briefly describe an example concerning differences in document representations and then provide a set of basic definitions concerning trees.

Example: Manuscript styles in scientific journals

In order to identify differences between various document representations, we analyzed a situation where authors have written an electronic document, or a set of documents, using a specific form and where they would later need them in one or several different forms. In the ideal situation, the author could change automatically the electronic representation to a new one. We considered manuscript styles of articles in eight scientific journals in the field of Computer Science. Scientific journals give usually very detailed rules for the layout and structure concerning submitted papers. We chose the following Computer Science journals:

ACM Transactions of Information Systems;
Journal of American Society of Information Science;
The Computer Journal;
Communications of the ACM;
Journal of Computer and System Sciences;
Information Systems;
Electronic Publishing - Origination, Dissemination and Design; and
SOFTWARE - Practice and Experience.

Although the journals have been selected from a single field, our study revealed that there are many differences. Some of them concern specific parts of texts (fonts, letter types, order and existence of elements), characters between text elements (spacing and various kinds of separators of the content elements, and codes for the layout), and whole structures of texts (such as places of footnotes, figures, tables, etc.). Since formats of scientific articles are not usually dependent on the subject, but rather the whims of the publisher we are sure that no other differences would have been observed if journals from different disciplines had been chosen. A detailed representation of the comparison of journal styles is found in the earlier version of this paper [31]. In this paper we describe only some examples.

Preliminary definitions

We consider documents that have a treelike structure. A *tree* is a collection of elements called *nodes*, one of which is distinguished as the *root*, along with a relation that places a hierarchical structure on the nodes [32]. A pair of nodes (a,b) in the relation is called an *edge* and it is said to leave node a and enter node b. For a tree it is true:

1. A single node by itself is a tree, being the root of the tree.

2. Let n be a node and $T_1, T_2, \ldots, T_k$ trees with roots $n_1, n_2, \ldots, n_k$, respectively. A new tree is constructed by making n to be the *parent* of nodes $n_1, n_2, \ldots, n_k$. In this tree, n is the root and $T_1, T_2, \ldots, T_k$ are the *subtrees* of the root. Nodes $n_1, n_2, \ldots, n_k$ are called the *children* of the node n.

If $n_1, n_2, \ldots, n_k$ is a sequence of nodes in a tree such that n_i is the parent of n_{i+1} for $1 \leq i \leq k-1$, then the sequence is called a *path* from node n_1 to node n_k. Node n_i is a *predecessor* of node n_j and node n_j is a *successor* of node n_i for $1 \leq i < j \leq k$. A child of node n is a *direct successor* of n and the parent of node n is the *direct predecessor* of node n. A node with no direct successors is called a *leaf*, nodes with direct successors are *internal nodes*. The *depth of a node* in a tree is the length of the path from the root to the node. The *depth of the tree* is the length of the longest path.

In an *ordered* tree, children of internal nodes are ordered from left to right. A child node has *sisters* to its left and to its right. In a *labelled* tree, nodes have labels.

Let T be a tree and n an internal node in T. A *parttree PT* rooted at node n is a subtree of T whose root is n, or a subtree of T whose root is n and from which one or more subtrees are replaced by their roots. If one child of an internal node n is in a part-tree then also all other children of node n are in the part-tree.

A *local part-tree* is a part-tree whose number of nodes is bounded by a small constant. A local part-tree cannot be very wide or very deep. A part-tree is considered *global* if it is not local.

Bases for the classification

The main question in the transformation from one tree to another is what kind of modifications should be made to which parts of a tree and how. Dissimilarities of two trees may be measured by the distance from one tree to another tree defined as a minimum cost sequence of edit operations (for example, change place, delete or insert one node) needed to effect the transformation [33]. A classification of differences in two trees using different values of the distance, however, does not support our declarative approach in document transformations. It is based on how a transformation should be carried out, rather than describing where the difference is and what type it is.

We will base our classification of differences between two trees on a pair of parttrees determining nodes which specify the difference and the nature of the difference between nodes. This kind of classification has a relation to the selection of transformation methods. The greater the number of nodes under consideration in a single translation step, the more complex the difference will be and the more information for the definition and control of the transformation is needed. The fundamental assumption for classifying differences in trees is that the association between the nodes of two trees under consideration is given.

Definition 1.

An *association* is a mapping h from a subset of nodes of a tree T to a subset of the nodes of a tree T' such that if u is a successor of v then $h(u)$ is a successor of $h(v)$.

Definition 2.

An association is *total* if its domain is the set of internal nodes of a tree T and its range is the set of internal nodes of a tree T'.

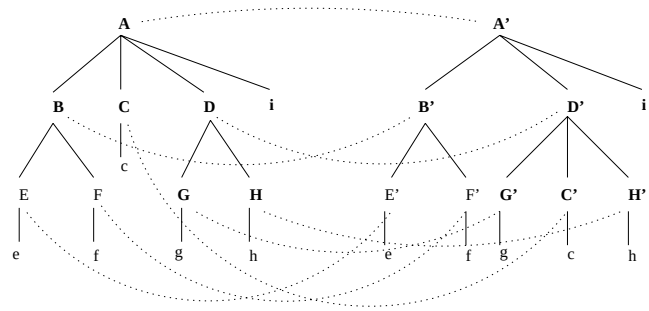

Figure 1.

Definition 3.

An association *keeps the structure* between trees T and T' if there is an association h between their internal nodes such that

(i) h is bijection between internal nodes of T and T', and
(ii) whenever $u_1, \ldots, u_n$ are internal children of v from left to right, then $h(u_1), \ldots, h(u_n)$ are internal children of $h(v)$ from left to right.

The total association h which keeps the structure defines an *isomorphism* between internal nodes of trees T and T'; thus, it defines that internal structures of trees are identical except for labels.

Definition 4.

An association is *local* if there are constants c and c' such that whenever nodes in a part-tree PT of a tree T with at most c nodes are mapped, the associated nodes are in a part-tree PT' of a tree T' with at most c' nodes. Part-Trees PT and PT' are called associated part-trees. An association is *basic* if it is local and depths of associated part-trees are equal to one.

In the following trees, labels of internal nodes are written by upper case letters and labels of leaves by lower case letters. The association is denoted by dotted lines between internals nodes and by identical labels between leaves. The association defines, for example, that nodes with the labels A and A' are associated and are roots of trees. Nodes whose labels are written in the bold font and edges between them present a pair of associated local parttrees of these trees. The association between them defines a local association between trees; it is total but does not keep the structure.

1) Simple differences: same structure, differences in leaves

Example 1.

The following bibliographic references for a book are presented according to the rules from two journals: Journal of Computer and System Sciences, and Information Systems.

Journal of Computer and System Sciences:	Information Systems:
1. A. V. AHO, AND J. D. ULLMAN, "The Theory of Parsing, Translation and Compiling, Vol. I: Parsing", Prentice-Hall, Inc., Englewood Cliffs, N. J., 1972.	1. A. V. Aho and J. D. Ullman. *The Theory of Parsing, Translation and Compiling, Vol. I: Parsing.* Prentice-Hall, Inc., Englewood Cliffs, N. J. (1972).

Figure 2.

The elements of both of these references are the reference number, the names of authors, the title of the book, the publishing company and its address, and the publishing year. Their orders are the same, but their layouts differ. Differences are in character strings that separate elements of the actual content (such as spaces, commas, periods, double quotes and parenthesis) and in the fonts (the title of the book).

Definition 5.

Two trees have *no structure difference* if there is a total association between trees which keeps the structure.

In the following trees the association *h* is denoted by dotted lines for internal nodes and using the identical letters for labels of leaves. Hence, there is no structure difference between trees. Differences are in leaves.

2) Local differences

a) Order difference

Example 2.
The following bibliographic information for a conference article reference is made according to the rules for a manuscript for two journals: Communications of the ACM, and Journal of Computer and System Sciences.
The elements in these references (reference number, names of the authors consisting initials and the last name, the title of the article, the name of the conference proceedings, the place and time of the conference, and page numbers of the article in the proceedings) are the same, there are however differences in their order. The last name and initials of the authors and, similarly, the time and the place of the conference are in the reverse order. In addition, there are font and separator differences.

Definition 6.

Two trees have an *order difference* if there is a total association between trees that does not keep the structure. Order differences are determined by local associations between nodes of trees.

In the following trees, the association is denoted by dotted lines for internal nodes and using the same letters for labels of leaves. In both figures, a pair of associated local part-

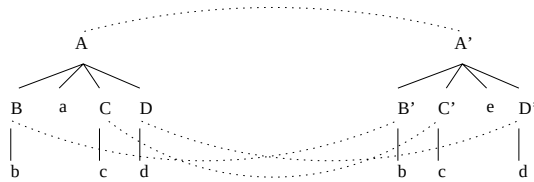

Figure 3.

Communications of the ACM	Journal of Computer and System Sciences:
1. Franchi-Zannettacci, P., and Arnon, D. S. Context-sensitive semantics as a basis for processing structured documents. In *WOODMAN'89, Workshop on Object-Oriented Document Manipulation* (Rennes, France) 1989, pp. 135-146.	1. P. FRANCHI-ZANNETTACCI, AND D. S. ARNON, Context-sensitive semantics as a basis for processing structured documents, *in* "WOODMAN'89, Workshop on Object-Oriented Document Manipulation, 1989", Rennes, France, pp. 135-146.

Figure 4.

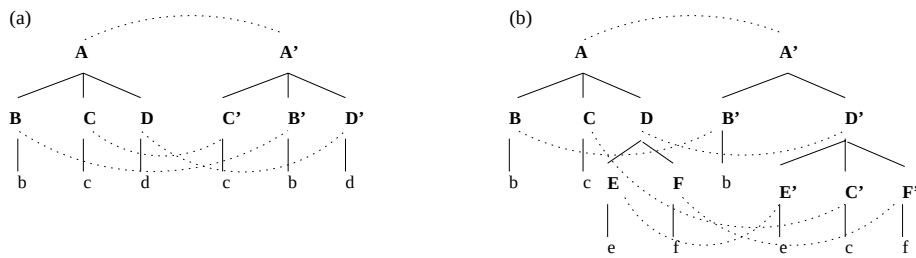

Figure 5.

trees contains nodes whose labels are written by the bold font. In (a), the local association determining associated part-trees is basic. In (b), the local association is non-basic.

b) Existence difference

Example 3.

Figure 6 presents rules for the title page of two different scientific journals: The ACM Transactions on Information Systems, and The Computer Journal. There are many structural

The ACM Transactions on Information Systems (TOIS):	The Computer Journal (CJ):
Title and Abstract. Use a specific and informative title. Typically, a title might contain .... Authors' names should be given without titles or degrees, along with the name of the sponsoring organization. Current mailing addresses, including email addresses, should be given in a footnote.	*Title page* This should contain the following information: the full and short title, and a complete list of authors, their affiliations and addresses. The correspondence author should be identified along with both his/her postal and e-mail addresses, and telephone and fax numbers.

Figure 6.

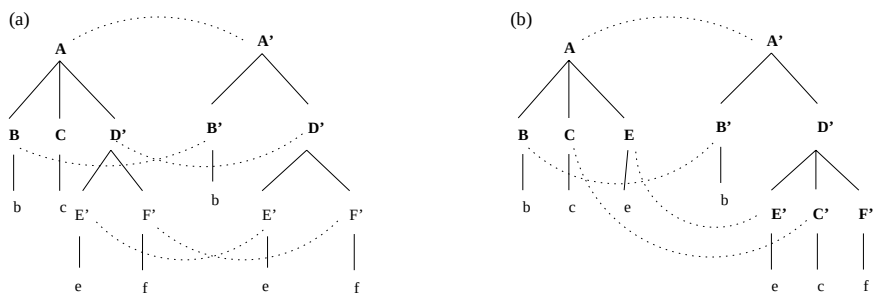

Figure 7.

differences between representations according to these rules. TOIS has a footnote which is missing from CJ. On the other hand, CJ requires a short title which TOIS does not. In TOIS, the author's information after the title consists of the name of the author and the name of the university or a company, the detailed address is in the footnote. In CJ, also the name of the department of the university or company is required. Further, in CJ only one author has a detailed mailing address which contains also telephone and fax numbers.

Definition 6.

Two trees have an *existence difference* if there is an association which is not total. Existence differences are determined by local associations between nodes of trees.

In the trees in Figure 7, the association is denoted by dotted lines between internal nodes and using the same labels for leaves. In both trees, a pair of associated local part-trees contains nodes whose labels are written by the bold font. In (a), associated part-trees

Journal of Computer and System Sciences (JCSS):	The Computer Journal (CJ):
Footnotes in text should be avoided if at all possible. If they must be used, identified by superscript numbers and type together on a separate page, double- or triple-spaced.	*Footnotes* Footnotes to the main text are acceptable and should be identified by superscripted numbers. Footnotes should appear on the page of citation.

Figure 8.

with existence difference are determined by a basic association. In (b), the associations are non-basic.

Global differences in the structure

Example 4.

The rules for footnotes in two journals, Journal of Computer and System Sciences, and The Computer Journal, are as follows.
JCSS requires that footnotes be placed in a list at the end of the manuscript. In a manuscript for CJ, footnotes are among other elements inside the text. Otherwise, footnotes are represented similarly.

Definition 7.

Two trees have a *global difference* if the association does not keep the structure, and there are no local associations.

In the following trees nodes labelled by C and C' are associated nodes which are located in very different places in these two trees. Associated part-trees contain nodes whose labels are written in the bold font and in addition, in the worst possible case, all the nodes from subtrees which are denotated by triangles. In this case, part-trees are "large" meaning that the numbers of nodes of part-trees are not bounded by small given constants.

In addition to the situation presented in Example 4, large associated part-trees arise also, if trees are represented using very different structures, meaning that, in the worst case only leaves of trees can be associated.

TREE TRANSFORMATION METHODS

There are many transformation methods suitable for trees to be found in the literature. This section presents brief descriptions of four different methods potentially useful for our purposes: syntax-directed translation, tree transducer, pattern matching and replacement methods, and manual methods. The syntax-directed translation requires that trees are parse trees for context-free grammars and transformation definitions specify syntactic structures of trees. Other methods are applied to labelled trees and use structural information of tree instances. They do not require structure definitions for the trees. Further, rules of a syntax-directed translation and tree transducer contain implicit mechanisms to locate places for changes in trees. In other methods, the user must specify the application order of rules.

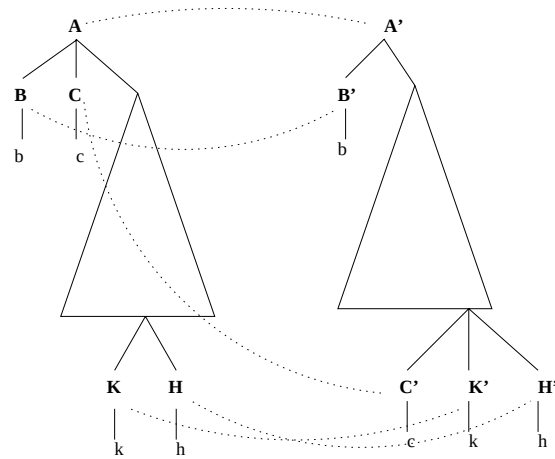

Figure 9.

Syntax-directed translation

Before representing the definition of syntax-directed translation, we will define context-free grammar, derivation and parse tree, concepts which are the basis of the method.

Context-free grammar, derivation and parse tree

A *context-free grammar* [21] consists of a finite set of non-terminals, one of them being a distinguished non-terminal called a start symbol, a finite set of terminals, and a finite set of productions in the form $A \rightarrow \alpha$, where A is a non-terminal and α is a sequence of non-terminals and terminals. A context-free grammar defines a formal language by specifying the symbols that can be used and the ways these symbols can be combined in a legal character string of the language. Such a string can be *derived* according to the grammar from the start symbol. In derivation, a non-terminal A in a sequence $\alpha A \beta$ (α and β are sequences of non-terminals and terminals) is replaced by a sequence on the right-hand side of a production $A \rightarrow \gamma$ to generate a new sequence $\alpha\gamma\beta$. If in each step the derivation is applied to the first non-terminal from the left, the derivation is called a *leftmost derivation*.

Parse trees for a context-free grammar G are defined as follows:

1. The root is a node labelled by the start symbol of G.
2. If a node is labelled by a non-terminal A then its children from the left are labelled by $\alpha_1, \ldots, \alpha_k$, where $A \rightarrow \alpha_1 \ldots \alpha_k$ is a production of G.

The parse tree defines a graphical representation of an equivalence class of derivations for a context-free grammar.

Syntax-directed translation method

As a method for compilers applied to formal languages, a syntax-directed translation (SDT) [21,22] combines a syntax analysis according to a grammar and code generation according

to a second grammar. A transformation is defined by a *schema* comprising of a finite set of non-terminals, one of which is a start symbol, a finite set of input terminals, a finite set of output terminals, and a set of rules defining an input form and an output form for each non-terminal of the schema. For example, in the following rule for a non-terminal S,

$$S \rightarrow ABb, BAb,$$

A and B are non-terminals and b is a terminal. The first part of the right side of the rule defines the input form and the second part the output form. In a rule, for each non-terminal of the input form, one will find associated an identical non-terminal of the output form. If a non-terminal occurs only once the association is obvious. If the same non-terminal exists more than once, then integer superscripts are associated with different occurrences of the same non-terminal name to indicate the associations.

An SDT schema has an input grammar and an output grammar. In a production of the input grammar, the right side of the production for the left side non-terminal of the SDT schema is the input form. The output form, containing the same non-terminals as the input form, is the right side of a production for the same non-terminal in the output grammar. For the above SDTS rule, the production of the input grammar is $S \rightarrow ABb$ and the production of the output grammar is $S \rightarrow BAb$.

In *syntax-directed translation schema* (SDTS) rules, the non-terminals in the output form are a *permutation of the non-terminals* in the input form. An SDTS defines translations from a parse tree for an input grammar to a parse tree for an output grammar reordering non-terminal children of a node and changing terminal children. In simple syntax-directed translation schema (SSDTS) rules, associated non-terminals in the input and output forms must be in the same order. An SSDTS defines transformations which only allow one to change the terminal children of a node.

The syntax-directed translation schema can be formed automatically from two grammars whose productions are paired and in each pair of productions, all non-terminals are associated. Aho and Ullman [21] provide an algorithm for the automatic transformation of a parse tree via an SDTS. The algorithm processes a parse tree from the root to leaves using the depth-first traversal of nodes of a modified tree. Each translation step is applied to a node and all its children. It removes all input terminal children of a node, reorders associated non-terminal children, and adds new output terminal children. An algorithm which uses an SSDTS can be employed in a similar way.

Another way to implement syntax-directed translations is to use the Szilard-word of the context-free grammar [34]. For a grammar whose *productions are associated with labels*, the Szilard word is a sequence of labels for a derivation according to the grammar. The tree transformation via Szilard word forms the leftmost Szilard word corresponding to the leftmost derivation for the input tree, maps the labels in the Szilard word to production labels for the output grammar and generates the output tree according to the mapped labels.

Tree transducer

A *tree transducer* [25,24] is a tree automaton with output that replaces a finite labelled tree with another finite labelled tree so that every transition rule of the transducer is defined for an occurrence of a subtree in the input tree, and a state of the transducer. A tree transducer is a *descending tree transducer* if it reads trees from the root towards the leaves. An *ascending*

tree transducer processes trees from the leaves towards the root.

In the transition rule of a tree transducer, an occurrence of a subtree in the input tree is defined by an input pattern and an occurrence of the replacing subtree for the output tree by an output pattern. In the descending tree transducer, the input pattern consists of the label of an internal node in an input tree associated with a state of the transducer, and variables for all subtrees of the node. The output pattern consists of labels of internal and leaf nodes in the output tree, and combinations which are formed from a state of a transducer and a variable in the input pattern. A *transition rule*, which is used to relabel internal nodes may take the form

$$qS(x_1,x_2,\ldots,x_n) \rightarrow R(px_1,px_2,...,px_n) \tag{1}$$

where q and p are states of a tree transducer, S and R labels for the internal nodes, and $x_1,x_2,\ldots,x_n$ variables corresponding to subtrees of a node labelled by S. The rule can be applied to the occurrence of a subtree whose root node is labelled by S and associated with the state q. It produces a subtree where the root is labelled by R and roots of subtrees of R are associated with the state p and labelled by labels of roots of subtrees corresponding to variables $x_1,x_2,...,x_n$. If the input and output patterns are specified as syntactic structures of trees and expressed with the use of the semantics for labels of the trees defined by grammars, tree transducers can be applied to parse trees.

In a stronger form of a tree transducer, rules are of the form

$$q(t(x_1,\ldots,x_n)) \rightarrow t'(p_1y_1,\ldots,p_my_m), \tag{2}$$

where t and t' are terms with variables in leaves. In this generalization, the terms t and t' can be of arbitrary depth, and the states $p_1,\ldots,p_m$ need not be equal. The set $\{y_1,\ldots,y_m\}$ of variables is not necessarily the same as $\{x_1,\ldots,x_n\}$. Either each y_i is some of variables $\{x_1,\ldots,x_n\}$ or there can be new variables among $y_1,\ldots,y_m$. In the latter case, the new variables remain unknown in the transformation. Another way to increase the transformational capability of tree transducers is to consider the context of an internal node of the input tree in the input patterns. This approach was used in various tree transducers introduced, for example, in [35–38].

Tree pattern matching/replacement methods

Transformations with the use of a tree pattern matching and a tree replacement consist of three steps according to [28]. The first stage, pattern matching, is the process of locating substructures in a larger structure by comparing them to a given form of a pattern. The second stage is the process of deciding between different replacements. The third step, pattern replacement, is a process that takes one substructure and replaces it with a new one. Languages for tree pattern matching/replacement methods specify a tree-to-tree transformation as sets of

```
pattern { replacement }
```

rules.

The tree pattern matching/replacement methods may be implemented in many ways. If the pattern is defined using syntactical structures, the transformations may be implemented with methods such as syntax-directed translation, tree transducer and term rewriting system

[26]. A term rewriting system defines the pattern for matching and replacement as terms in a similar way as a tree transducer, but has no states to define where to apply a rule. The syntax-directed translation approach is also used in the method presented in [27] which specifies transition rules using a paired sets of productions of grammar for the input and output patterns. Algorithms for the tree pattern matching problem presented in [28] are mainly extensions to string pattern matching algorithms. Whereas, algorithms in [29] are based on a tree inclusion. Pattern matching is also an essential part of query languages for structured text [39]. Usually efficient pattern matching methods require some kind of pre-processing for patterns and/or for input trees providing additional information (tables, indices).

Manual transformations

In manual transformations, the user modifies individual trees and defines both the rules and the control flow for transformation. Only the environment (a specification language or an editor) can be offered as a tool for the user.

As a filter we understand a sequence of filtering operations whose execution is manually controlled. The transformation is implemented usually by a programme which is made either by a common or by a special programming language [8–10, 23]. The transformation is specified as a set of

```
pattern { action }
```

rules. Transformations implemented by filter programmes are often employed for specific tasks.

An interactive tree editor, for example such as in [30], allows the user to make modifications in any single part of a tree. The editor represents a tree on the screen and the user modifies the tree with the use of editor commands or a mouse. This permits any modifications, but there is no way to carry out similar modifications to all trees of one type.

METHODS FOR VARIOUS TYPES OF DIFFERENCES IN DOCUMENTS

Because in this work we want to apply methods described in the previous section to structured documents, our requirement is that the method takes as its input a parse tree for a grammar and produces a parse tree for another grammar, accompanied by the grammar. In the following subsections, for each difference class we describe several methods which in various situations are capable of making the required modifications. Our aim is to describe the transformation capabilities of the methods in general although we have not implemented all of them yet. Since documents are assumed to be grammartical, internal nodes are called *non-terminal nodes* and leaves are called *terminals nodes*.

Transformations that keep structure

A simple syntax-directed translation schema (SSDTS) can be used to define tree transformations which only delete existing terminal nodes and add new ones. Terminals are character strings which are the same for all documents defined by a grammar, and therefore may be deleted or added without losing information. Other nodes in a parse tree, either

defining the structure of a document or containing the actual content text, are considered to remain unchanged.

All the information needed for the transformation of a parse tree, in order to change terminals, is contained in an SSDTS which can be created automatically from the input and output grammars. An output grammar, if one does not exist, is made from an input grammar by an editor which permits only the modifications of terminals. This guarantees that structures of documents remain the same in a transformation. An automatic batch-oriented transformation programme can be made either according to the algorithm in [21], the algorithm which uses Szilard words, or by making a compiler which compiles a transformation programme automatically from an SSDTS as we implemented in [40]. Our purpose in [40] was to generate various layouts from an internal representation of a structured document in a syntax-directed text processing system. Generated programmes transform document instances automatically. The SSDTS can also be used interactively to execute incremental transformations in the input phase of structured texts. As described in [40], we used an SSDTS also to implement the interactive input of structured documents in a syntax-directed text processing system. The input tree is created on the computer screen from the pieces of text written by the user according to an input grammar. An internal representation is transformed from this according to an SSDTS. The automatic transformation via SSDTS is well defined, the difficulties in applying it to documents are in the definitions of grammars.

A parse tree for a document can also be represented as a character string that uses tags interleaved with the content to mark the structure. The tags are terminals of the grammar for the string representation and correspond to terminal nodes in a parse tree. The transformation to change tags to other tags can be implemented by a filter programme if there is a string homomorphism [21] between tags on the input and output representation. In this case, the automatic filter programme only substitutes terminals for new terminals.

Transformations for local order differences

a) Order differences determined by basic associations

A syntax-directed translation schema (SDTS) offers a method to reorder non-terminal children (with their denominated subtrees) of associated non-terminal nodes. The automatic implementation of transformations managing this kind of order differences can be achieved similarly as described for the SSDT in the previous subsection. If one has an editor to create an output grammar from an input grammar, the user is permited to move non-terminals in the right side of a grammar production, but not to delete them. As an SSDTS, also an SDTS can define also transformations for terminal differences if needed. Our implementation of the transformation of structured documents described in [41] suits also for transformations via an SDTS.

An SDT is based on syntactic structures of trees. It can process a list of unknown number of nodes labelled by the same non-terminal only in sequential order and it cannot locate some particular nodes in the list. If reordering or recognition of non-terminal nodes in such a list is needed transformation must be implemented using tree pattern matching/replacement methods which permit the user to locate nodes according to the order or content. In the tree pattern matching/replacement methods, patterns are described by the language of the selected method. Only in restricted cases, structure grammars may be used to help the user

to generate patterns (check relations or give warnings), not to define them. The user is responsible for the order in which the rules are applied in the input tree. If the control of the transformation is not defined, rules are applied to a parse tree of a document as long as there are matching structures.

b) Order differences determined by non-basic associations

A descending tree transducer allows local moving of non-terminal nodes from a place to another. If transition rules are represented in the form represented in Equation (2) on page 330, it permits to move non-terminal nodes from a hierarchy level to another, and in addition, to rename labels of non-terminal nodes and delete or insert terminal nodes. This kind of descending tree transducer can be used used to parse trees of document representations to employ modifications (including order differences) which concern more than one, however bounded number of hierarchy levels. In order to guarantee that structures of documents obey the grammars of the old and the new structure, an interactive editor to help the user to generate transition rules would be needed to represent two subsets of grammar productions as two trees, to allow the user to specify associations between their leaves, and then to form two terms from these trees to correspond to the input and output pattern of a rule. It remains however open, how the construction of a tree transducer can be automatized. In this work, we made transformation rules manually and the transformation programme was compiled automatically from these rules.

Tree pattern matching/replacement methods allow the recognition of any kind of local structures in document instances according to their structure and content, and replace them with new instances. These methods should be used if the order difference is not defined according to syntactic structures between non-terminal nodes.

Transformations for local existence differences

Existence differences determined by basic associations

In the simplest case, an existence difference means that a deletion of a node as a child of another non-terminal node deletes its subtrees as well and an insertion of such a node adds an atomic non-terminal node, without any subtrees. The modification in each step of a transformation concerns only a non-terminal node and its children. In [41] we describe an *extended syntax-directed translation schema* (ESDTS) as a natural extension to an SDTS for these kinds of changes. The difference compared with the SDTS is that an ESDTS does not require that all the productions of input and output grammars are paired and that all the non-terminals in input and output forms are associated. Tree transformations via an ESDTS can reorder, add and delete non-terminal children with their denominated subtrees and change terminal children of a non-terminal node. Adding a non-terminal child however, means only adding a new non-terminal node as a non-terminal leaf. The user will subsequently generate its subtrees according to the output grammar. An ESDTS can be formed from productions of input and output grammars [31]. An editor can be implemented to help the user to create an output grammar from an input grammar. For a non-associated production, either the input form or the output form is an empty string. Our algorithm for a tree transformation via an ESDTS [31] is a modification from the corresponding SDTS algorithm [21]. A single step of the algorithm cuts input terminal children and non-associated non-terminal children (and their subtrees) from the input tree, reorders the

associated non-terminal children and their denominated subtrees, and adds nodes labelled by output terminals as terminal leaves and nodes labelled by non-associated output non-terminals as non-terminal leaves. The automatic, batch-oriented transformation using an ESDTS can be implemented by a programme for using our algorithm mentioned above, or by making a compiler which automatically generates the transformation programme from an ESDTS as we employed in [41].

Example 5. Local transformation via ESDTS

The examples of manuscript styles in the second section of the paper presented structure differences in manuscripts of eight scientific journals. Transformations of the manuscript of this paper written using our structure for an article to the required structures of all of these journals would entail structure and layout modifications. In order to test our ideas of structure transformations, we defined grammars for structures of the front part and the bibliographies of manuscripts and implemented the transformations from the structure of our manuscript to the structures requested by these journals.

An ESDTS was sufficient to define the transformation to the format of The Computer Journal and Software — Practice and Experience journals. The differences were only in order or existence of non-terminals on the right sides of grammar productions, for example, a missing short title, postal address, telephone and fax number, and different places for the publishing year of a bibliographic reference.

b) Existence differences determined by non-basic associations

In addition to modifications needed for order differences and mentioned earlier, a descending tree transducer allows one locally to add, delete and duplicate non-terminal nodes (with or without their denominated subtree2s). Subtrees of a deleted node can be deleted or moved to be subtrees of other nodes. An inserted node can be without subtrees or subtrees for it can be moved or copied from subtrees of other nodes. In this work we used the implementation based on syntactic structures of parse trees as described for order differences.

Example 6. Local transformation via tree transducer

We consider the situation described in the Example 5.

A descending tree transducer was used to carry out transformations for Journal of American Society of Information Science (JASIS), Communication of the ACM (CACM), Journal of Computer and System Science (JCSS), Information Science (IS) and Electronic Publishing - Origination, Dissemination and Design (EPODD) journals. The tree transducer was used because associated non-terminals exist in different grammar productions (for example, authors' names and affiliations, information in footnotes, and first page number in a bibliographic reference) and input and output patterns correspond trees whose depths are greater than one. Actually, transformations of the bibliography to JASIS, JCSS, IS and EPODD could have been implemented with the use of an ESDTS. Their structure differences were restricted to non-terminals on the right side of the productions.

Tree pattern matching/replacement methods, allowing the recognition of any kind of local structures in document instances according to their structure and content, must be

used in local transformations to add, delete or duplicate nodes not recognized by syntactic structure definitions.

A transformation needs a filter programme, if operations applied to elements are needed, for example, if an element must be replaced with several elements by parsing it, or if several elements should be replaced with one element by unparsing them. The user is responsible for the definition of the location of the elements and writing of the required procedures.

Example 7. Local transformation via filters

We consider the same situation as described in Example 5.

The transformation for The ACM Transactions of Information Systems (ACMTOIS) journal would have needed a filter programme because the transformation should have had to parse names of authors in our manuscript to the initials and the last names of the authors when names were copied to the author information part at the end of a manuscript for ACMTOIS. Since we have not yet implemented filter-based transformations in our prototype, we made the transformation with the use of a tree transducer. The transformation generated new atomic elements for the initials and for the last name of an author. The user should add the content later.

3) Transformations for global differences

The transformation between structured documents having global differences in their structures requires either that one reorder elements, delete or insert a set of elements, and move or copy elements or a set of elements from one place to another if the old and new locations of associated elements are not situated inside local parts of a trees.

Filter programmes and tree editors allow modifications that are not restricted to some local parts of parse trees and can be used in the above kinds of transformations. For document transformations, the tree pattern matching in a filter programme should be based on the integration of the structure and content as well as in the order or quantity of elements. The filter must be able to save structures temporarily and replace them later, and process a document more than once. In a tree editor, it should be possible to spread an executed single modification in a tree to all similar structures, thus, taking care of the cases where the same elements have the same structure in a document.

The global modification may be employed in some cases also by local methods (an SDTS, an ESDTS, a tree transducer, or tree pattern matching/replacement methods) if the modification can be processed incrementally applying the methods sequentially so that the output of a previous transformation is the input of the next transformation.

Global transformations are always made under the control of the user only. Either the user edits each single document separately, writes a filter programme which achieves the transformation in its entirety, or creates grammars or transformation rules for the incremental steps of the transformation. If there is an output grammar, the methods should check that the modified document is valid according to the grammar. If the user does not have an output grammar, the transformation system should use methods that can generate the grammar of a transformed document [42].

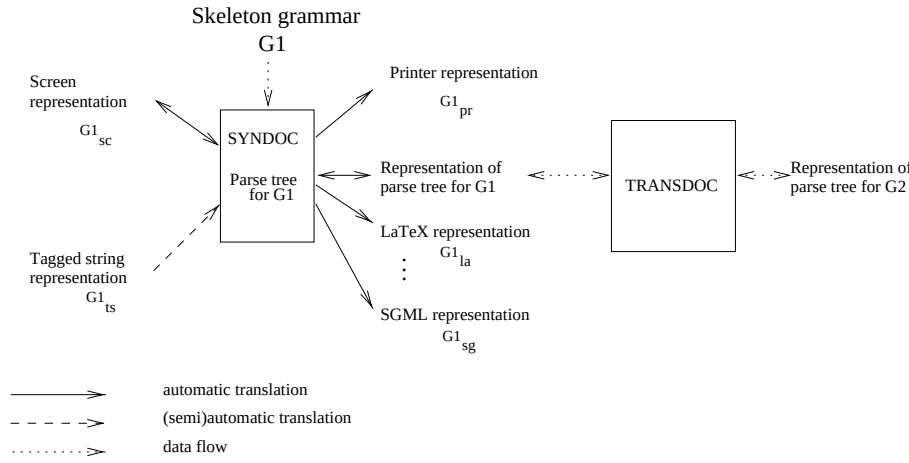

Figure 10. A model for a document transformation system

MODEL FOR A DOCUMENT TRANSFORMATION SYSTEM

In the previous sections, methods suitable for different kinds of document transformations were characterized. As we saw, if the hierarchical structure of a document is not changed or if only reordering of subelements of an element is needed, the transformation can be made automatically. Otherwise, transformations can be managed automatically only in specific cases which, however, may need the user's help to define the transformation rules.

Our model for a document transformation system separates transformations that are simple and easy to automatize from transformations that need more sophisticated methods. We divide the system into two tools (Figure 1). The first tool allows only transformations that do not change the structure and the second one makes transformations from one structure to another. In the first tool, document editing and transformations that keep structure are possible. The second tool has a set of available methods for structure modifications and offers the possibility of analyzing differences between the definitions of an old and new representation and use different methods according to the result of the analysis.

The first tool called SYNDOC [40], was originally planned as a tool for a SYNtax-directed DOCument processing system including inputting, editing and outputting. It contains those transformations which produce representations for the user of the system. The input, output, storage, import and export of a document in different forms need transformations in this group. The internal representation for documents in SYNDOC is a parse tree for a context-free grammar in extended Backus-Naur Form [21] which defines the hierarchical, logical structure of a document and contains only non-terminals. We call the grammar *skeleton grammar* and it is not used, for example, to generate or parse a string representation of a parse tree. Grammars for external representations are skeleton grammars added by terminals. Whereas the internal representation is only for the use of the system, external representations are those that are generated for the user. With this tool, transformations are always between the internal representation and an external representation. They are defined automatically from the grammars for the input and output representations and are executed automatically or semi-automatically. Figure 10 describes those transformations that are automatic with solid arrows and those transformations which may need help

from the user with dashed arrows. User's help is needed if the input representation of the transformation contains ambiguous notations for the structure of a document (for reasons stated in [5]). Dotted arrows describe data flows.

The second tool called TRANSDOC, TRANSlation system for DOCuments, manages transformations from a structure to another. The input of TRANSDOC is a representation of a parse tree for the skeleton grammar and the output is a representation of a parse tree for the new skeleton grammar. The representation must define the parse tree unambiguously. It can be a tree representation of SYNDOC or a string representation of a parse tree produced by SYNDOC or some other system and accepted by the methods of TRANSDOC. The only assumption in TRANSDOC is that transformation methods create a representation for one grammar from a representation to another grammar. Each available method contains its own tools to help the user to define the transformation definitions. Methods are selected in TRANSDOC based on the difference type between the input and output representation. Hence, TRANSDOC needs methods to compare grammars or representations, to identify differences existing in parse trees.

In our earlier works we have developed the prototype for the SYNDOC [40]. The simple syntax-directed translation schema is used for the syntax-directed input of documents and for the generation of the outputs for documents. A grammar directs the syntax-directed input. Different kinds of layout representations can be produced for the same structured content using various output grammars. The implementation of transformation methods for TRANSDOC has started as a part of SYNDOC. The implementation of transformations using an extended syntax-directed translation schema is described in [41]. Rules for an ESDTS were generated automatically using input and output grammars. A tree transducer was implemented as described briefly in this paper and in detail in [31]. To obtain a tree transducer, transition rules were created manually using a text editor.

RELATED WORKS

In contrast to our system, many transformation methods for structured documents found in the literature are intended for some specific task. The method developed by Brown *et al.* [6] for documents defined by attribute grammars, or the method defined by the standard for Document Style Semantics and Specification Language (DSSSL) [9] for SGML documents employ transformations for the layout process of documents. Part of the transformations defined in DSSSL may be used also in static transformations of SGML documents. The methods developed by Furuta and Stotts [4] and Akpotsui and Quint [1] and Scrimshaw language developed by Arnon [8] are suitable for static transformations of documents defined by context-free grammars. GOEDEL language [10] may be used to static transformations for structured documents which do not have a structure definition.

The systems and methods closest to our document transformation model are listed below. The first two are mainly aimed for representation transformations, the second two for static transformations.

The translation programme qwertz/FORMAT [43] is a software to change SGML [44] representation of a document to representations containing some other markups (for example, for LaTeX and nroff/troff formatters). Transformations do not change the structures of documents. Integrated Chameleon Architecture (ICA) [5] is the software environment to generate translators for automatic modifications of documents' character string representations from the form of a text formatter to the form of another formatter. Transformations in

ICA mainly do not change the structure, but reordering of elements inside another element is possible, however. A transformation from one representation to another is made using an intermediate representation defined by SGML Document Type Definition [44] and represented as a parse tree and used only by the system. Grammars for document representations are used to generate programmes for the transformation.

More complex structure modifications are possible in SIMON [7] and using the method represented by Chiba and Kyojima [11]. SIMON, a grammar-based transformation system for structured documents is suitable for transforming documents with local and global differences. Documents are defined by attribute grammars and a transformation is defined by a higher-order extended attribute grammar. Chiba and Kyojima used a syntax-directed translation schema to define transformations between instances of different document types. Similarly to us, they require that the input and output document have to have a structure definition represented as a context-free grammar. Their method applies the syntax-directed translation method to a specific string representation for a tree. The syntax-directed translations schema is generated from two grammars for string representations for the input and output document trees, respectively, not from grammars for logical structures of documents. This produces a system capable of achieving transformations between documents with certain local differences, for example, transformations which need to localize members of lists of the same elements. The additional power of SIMON and the method of Chiba and Kyojima is however counterbalanced by the complexity in the way transformation definitions are created.

CONCLUSION

Transformation of documents occurs very often when documents have to be reprocessed. The reuse of documents has made it even more important because today's computer systems contain a huge amount of information in electronic form and via nets like the Internet they are available worldwide. This information is represented in very many ways, more and more often in a structured form. Automatic transformations of these many representations are only possible if the system has some information about the documents that it has to process. A structure defined by a grammar often contains sufficient information.

This work has analysed the suitability of tree transformation methods for tree-structured documents. Our goal was to model a document processing as transformations between all different representations for documents. Structures of documents are defined by context-free grammars and different document representations are parse trees for their own grammars. Transformations either change the structure of a document or leave it unchanged. A classification for differences in trees was defined to get a basis for the selection of the most suitable method for each transformation. After brief descriptions of different tree transformation methods, this paper described which methods could be used for each kind of difference class and, finally, described a model for a system for structured documents based on transformations.

In such a transformation system, methods like a simple syntax-directed translation schema, a syntax-directed translation schema, an extended syntax-directed translation schema, a descending tree transducer, pattern matching/replacement methods, and a language to generate tree pattern matching based filters as well as a tree editor, form a powerful environment for making modifications in documents whose structures are defined by context-free grammars. But instead of asking the user to select the method, according

to our model, the most suitable method may be decided by the system. The translation system is divided into two modules. The first module manages transformations which do not change the hierarchical structure of a document and uses a simple syntax-directed translation method. The second module is used when structure modifications are needed and it uses several methods. Each method would contain its own tools to generate transformation programmes from the grammars of the different representations either automatically or with help of the user. The transformations produce documents for a grammar, or at least check the validity of a document against a grammar.

In this work we continued the implementation of this kind of a system as one part of our prototype for a syntax-directed document processing system. The prototype itself implements transformations when the structure is not changed. Current implementation of the second module has started in the prototype and contains the main components for the syntax-directed translation methods and a tree transducer. The development of a separate document transformation system will be tackled in our future research. We will attempt to extend the use of these novel methods to cover more complex differences as well as to apply the model to SGML documents.

ACKNOWLEDGEMENTS

The authors would like to thank Magnus Steinby and Frank Tompa for their help and comments during this work, and the referees whose comments significantly improved the paper. The research was financed by the Academy of Finland and supported by grants of the Saastamoinen Foundation and the Emil Aaltonen Foundation, which are gratefully acknowledged.

REFERENCES

1. E. Akpotsui and V. Quint, 'Type transformations in structured editing systems', in *EP92, Proceedings of Electronic Publishing, 1992*, eds., C. Vanoirbeek and G. Coray, 27–41, Cambridge University Press, Cambridge, (1992).
2. D.D. Cowan, E.W. Mackie, G.M. Pianosi, and G. de V. Smit, 'RITA - an editor and user interface for manipulating structured documents', *Electronic Publishing - Origination, Dissemination and Design*, **4**(3), 125–150, (1991).
3. C. Roisin and E. Akpotsui, 'Implementating the Cut-and-Paste operation in structured editing system', in *Second Workshop on Principles of Document Processing, PODP'94*, (1994).
4. R. Furuta and P.D. Stotts, 'Specifying structured document transformations', in *Document Manipulation and Typography*, ed., J.C. van Vliet, 109–120, Cambridge University Press, Cambridge, (1988).
5. S. Mamrak, C.S. O'Connell, and J. Barnes, *Integrated Chameleon Architecture*, PTR Prentice Hall, Englewood Cliffs, N.J., USA, 1994.
6. A.L. Brown Jr. and H.A. Blair, 'A logic grammar foundation for document representation and document layout', in *EP90, Proceedings of the International Conference on Electronic Publishing, Document Manipulation & Typography*, ed., R. Furuta, 47–64, Cambridge University Press, Cambridge, (1990).
7. A. Feng and T. Wakayama, 'SIMON: A grammar-based transformation system for structured documents', *Electronic Publishing - Origination, Dissemination and Design*, **6**(4), 361–372, (1993).
8. D.S. Arnon, 'Scrimshaw: A language for document queries and transformations', *Electronic Publishing - Origination, Dissemination and Design*, **6**(4), 385–396, (1993).
9. ISO DIS 10179.2, *Information technology - Text and office systems - Document Style Semantics and Specification Language (DSSSL)*, 1994.

10. E. Blake, T. Bray, and F.W. Tompa, 'Shortening the OED: Experience with a grammar-defined database', *ACM Transactions on Information Systems*, **10**(3), 213–232, (1992).
11. K. Chiba and M. Kyojima, 'Document transformation based on syntax-directed tree translation', *Electronic Publishing - Origination, Dissemination and Design*, **8**(1), 15–29, (1995).
12. E. Kuikka and E. Nikunen, 'Rakenteisten tekstien käsittelyjärjestelmistä (Systems for structured documents)', Technical Report A/1994/4 (in Finnish, English version avaible at WWW address `http://www.cs.kuopio.fi/~kuikka/systems.html`), University of Kuopio, Department of Computer Science and Applied Mathematics, Finland, (1994).
13. A.N. Habermann and D.S. Notkin, 'Gandalf: Software development environments', *IEEE Transactions on Software Engineering, SE-12*, **12**, 1117–1127, (1986).
14. *The Synthesizer Generator Reference Manual*, eds., T.W. Reps and T. Teitelbaum, Springer-Verlag, New York, USA, 1989.
15. D. Garlan, C.W. Krueger, and B.S. Lerner, 'TransformGem: Automating the maintenance of structure-oriented environments', *ACM Transactions on Programming Languages and Systems*, **16**(3), 727–774, (1994).
16. J.P. Fry (Ed.), 'Conversion technology, an assessment', *ACM SIGBDP Data Base*, **12&13**(4&1), 39–61, (1981).
17. M. Ahlsen, A. Björnestedt, S. Britts, C. Hulten, and L. Söderlund, 'Making type changes transparent', Technical Report No. 22, University of Stockholm, SYSLAB, Sweden, (1984).
18. J. Banerjee, H.-J. Kim, W. Kim, and H.F. Korth, 'Schema evolution in object-oriented persistent databases', *ACM SIGMOD Record*, **16**(3), 311–321, (1987).
19. A. Borgida, 'Language features for flexible handling of exceptions in information systems', *ACM Transactions of Data Base Systems*, **10**(4), 565–603, (1985).
20. A.H. Skarra and S.B. Zdonik, 'Type evolution in an object-oriented database', in *Research Directions in Object-Oriented Programming*, eds., B. Shriver and P. Wegner, 393–416, The MIT Press, Cambridge, Massachusetts, (1987).
21. A.V. Aho and J.D. Ullman, *The theory of parsing, translation, and compiling, Vol. I: Parsing*, Prentice Hall, Inc., Englewood Cliffs, N.J., USA, 1972.
22. A.V. Aho and J.D. Ullman, *The theory of parsing, translation, and compiling, Vol. II: Compiling*, Prentice Hall, Inc., Englewood Cliffs, N.J., USA, 1973.
23. J.R. Cordy and I.H. Carmichel, 'The TXL programming language syntax and informal semantics, version 7', Technical Report 93-355, Department of Computing and Information Science, Queen's University at Kingston, Canada, (1993).
24. F. Gécseg and M. Steinby, *Tree Automata*, Académiai Kiadó, Budapest, 1984.
25. J.W. Thatcher, 'Tree automata: An informal survey', in *Currents in the Theory of Computing*, ed., A.V. Aho, 143–172, Prentice Hall, Inc., Englewood Cliffs, N.J., USA, (1973).
26. *Term Graph Rewriting, Theory and Practice*, eds., M.R. Sleep, M.J. Plasmeijer, and M.C.J.D. van Eekelen, Wiley & Sons, Chichester, UK, 1993.
27. S.E. Keller, J.A. Perkins, T.F. Payton, and S.P. Mardinly, 'Tree transformation techniques and experiences', *SIGPLAN Notices*, **19**(6), 190–201, (1984).
28. C.M. Hoffman and M.J. O'Donnell, 'Pattern matching in trees', *Journal of the ACM*, **29**(1), 68–95, (1982).
29. P. Kilpeläinen and H. Mannila, 'The tree inclusion problem', in *TAPSOFT'91, Proceedings of the International Conference on Theory and Practice of Software Development, Vol. 1: Colloquium on Trees in Algebra and Programming (CAAP'91)*, eds., S. Abramsky and T.S.E. Maibaum, pp. 202–214. Springer-Verlag, Berlin, (1991). Lecture Notes in Computer Science No. 493.
30. P. Desain, 'Tree Doctor, a software package for graphical manipulation and animation of tree structures', in *Human-Computer Interaction: Psychonomic Aspects*, eds., G. van der Veer and G. Mulds, 223–236, Springer Verlag, Heidelberg, (1988).
31. E. Kuikka and M. Penttonen, 'Transformation of structured documents', Technical Report CS-95-46, University of Waterloo, Department of Computer Science, (1995).
32. A.V. Aho, J.E. Hopcroft, and J.D. Ullman, *Data Structures and Algorithms*, Addison-Wesley, Reading, Massachusetts, 1983.
33. K.-C. Tai, 'The tree-to-tree correction problem', *Journal of the Association for Computing Machinery*, **26**(3), 422–433, (1979).
34. A. Salomaa, *Formal languages*, Academic Press, NewYork, N.Y., USA, 1973.

35. J. Engelfriet, 'Top-down tree transducers with regular look-ahead', *Mathematical Systems Theory*, **10**, 289–303, (1977).
36. J. Engelfriet and H. Vogler, 'Macro tree transducers', *Journal of Computer and System Sciences*, **31**, 71–146, (1985).
37. B. Courcelle and P. Franchi-Zannettacci, 'Attribute grammars and recursive program schemes, I and II', *Theoretical Computer Science*, **17**, 163–191, 235–257, (1982).
38. J. Engelfriet and H. Vogler, 'High level tree transducers and iterated pushdown tree transducers', *Acta Informatica*, **26**, 131–192, (1988).
39. R. Baeza-Yates and G. Navarro, 'Integrating contents and structure in text retrieval', *ACM SIGMOD Record*, **25**(1), 67–79, (1996).
40. E. Kuikka, M. Penttonen, and M.-K. Väisänen, 'Theory and implementation of SYNDOC document processing system', in *Proceedings of the Second International Conference on Practical Application of Prolog*, pp. 311–327, London, UK, (April 1994).
41. E. Kuikka and M. Penttonen, 'Transformation of structured documents with the use of grammar', *Electronic Publishing - Origination, Dissemination and Design*, **6**(4), 373–383, (1993).
42. H. Ahonen, H. Mannila, and E. Nikunen, 'Generating grammars for SGML tagged texts lacking DTD', in *Second Workshop on Principles of Document Processing, PODP'94*, Darmstadt, Germany, (1994).
43. Institut for Applied Information Technology, German National Research Center for Computer Science, Schloss Birlinghoven, Germany, *The qwertz SGML Document Types, Version 1.2, Reference Manual*, October 1992.
44. C.F. Goldfarb, *The SGML Handbook*, Oxford University Press, Oxford, UK, 1990.