On classification of XML document transformations*

Jana Dvořáková

Department of Computer Science, Faculty of Mathematics, Physics and Informatics, Comenius University, Bratislava, Slovak Republic
Jana.Dvorakova@dcs.fmph.uniba.sk

Abstract. As XML has become a very popular standard for data in many fields, the domain of XML documents transformations is becoming more and more important. In this paper we propose classification hierarchy for XML document transformations. We assign implemented transformation systems into defined groups according to the type of possible transformations. This enables user to choose appropriate transformation system according to the requirements for transformation. Secondly, we are concerned with the group of transformation systems, which enable type transformation. We define underlying formal models in common framework and discuss their applicability for these transformations.

Key words: XML, Structured document, Document type, Document transformation, Transformation classification

1 Introduction

XML (Extensible Mark-up Language) [28] is a meta-language recommended by W3 Consortium in order to create structured documents. In XML document structure, content and presentation are strictly separated. Unlike plain text, it contains special tags, which decompose document into logical parts. Therefore we say that XML belongs to the group of mark-up languages. Presentation of XML document can be described by several languages, usually in external file. Most common are XSL (Extensible Stylesheet Language) and CSS (Cascading Stylesheet).

Nowadays the usage of XML is growing very fast. It is a suitable tool in every field, where it is necessary to create document standards. Furthermore it has become very popular as a format for data exchange among applications since for programs it is necessary to mark semantics of data explicitly.

For this reasons various transformations are needed. Many transformation systems for structured documents have been implemented, some of them are based on formal models while others were created only ad-hoc. We have seen several reviews of existing transformation systems ([19, 18, 20, 11]). The author

* This work was supported in part by the grant VEGA 1/0131/03

K. Richta, V. Snášel, J. Pokorný (Eds.): Dateso 2005, pp. 69–83, ISBN 80-01-03204-3.

in [20] introduces also some basic categorization, however in none of these works a complete classification system has been defined.

The rest of this paper is organized as follows: After defining some notions used through-out the paper we introduce defined classification system in Section 3. We discuss reasons for choosing particular criteria and we assign implemented transformations systems for structured documents into defined groups. In section 4, we are dealing with one specific group of transformations, namely type transformations. We are examining formal models on which these transformations are based and we define them in a common framework. At last we present several results obtained by comparing these formal models. Section 5 contains a brief conclusion and outlines our future research.

2 Notions and notations

2.1 XML document

Basic logical unit of an XML document is *an element*. The content of an element is delimited by a start-tag and an end-tag. It can contain text and other nested elements as well. Variables called *attributes* can be assigned to an element. Obviously, an XML document has a hierarchical structure and it can be represented by a tree, where internal nodes are elements and leaves contain textual content.

The framework considered in this paper is restricted in two ways. Firstly, we do not consider element attributes of XML documents since we are concerned mainly with transformations at the level of elements. Secondly, we assume that element names of XML documents are from a finite and known set denoted by E. This enables us to define XML documents as trees over finite alphabets, which is the most natural way as far as we consider XML documents transformation. Furthermore we will use symbol $Char$ for alphabet of characters allowed to appear in the textual content of XML document. $Char$ is specified in W3C recommendation [28].

Definition 1. *Let Σ, Γ be alphabets. The set of trees over (Γ, Σ) denoted by $T_\Gamma(\Sigma)$ is defined as follows:*

- *$a \in \Sigma$, then $t = a \in T_\Gamma(\Sigma)$ and $root(t) = a$.*
- *$A \in \Gamma$, $t_1, \ldots, t_k \in T_\Gamma(\Sigma)$, $k > 0$, then $t = A(t_1, \ldots, t_k) \in T_\Gamma(\Sigma)$ and $root(t) = A$.*
- *nothing else belongs to $T_\Gamma(\Sigma)$.*

We call Σ *a leaf alphabet* and Γ *an internal node alphabet*. Alphabets Σ, Γ do not need to be disjoint.

Obviously the set of XML documents denoted by D equals to the set of trees over $(E, Char)$, i.e., $D = T_E(Char)$.

2.2 DTD

An XML document type is a class, which contains XML documents with similar structure. It is specified by *a type definition*, which describes the set of elements, that should be contained in the document as well as relationships among the elements. W3C has defined two ways of notation - DTD and XML schema. DTD is simpler and describes particularly syntax of the type while XML schema introduces more aspects like namespaces, restrictions for attribute values, etc. However, syntactic features of both models can be easily captured by context-free grammar. We will use this concept in the rest of this paper. Now we will define context-free grammars and their subset - type grammars, which describe XML document types.

Definition 2. *Context-free grammar (CFG) is a 4-tuple* $G = (N, T, P, S)$*, where* N *is an alphabet of nonterminal symbols,* T *an alphabet of terminal symbols,* $P \subseteq N \times (N \cup T)^*$ *is a finite set of production rules and* $S \in N$ *is a starting symbol.*

Definition 3. *Let* $G = (N, T, P, S)$ *to be a CFG, then a set of derivation subtrees of* G *denoted by* S_G *is defined as follows:*

- $a \in T$*, then* $t = a \in S_G$*.*
- $A \in N$*,* $t_1, \ldots, t_k \in S_G$ *and* $A \to root(t_1), \ldots, root(t_k) \in P$*, then* $t = A(t_1, \ldots, t_k) \in S_G$*.*
- *nothing else belongs to* S_G*.*

A set of derivation trees of G *-* T_G *is a subset of* S_G *such that* $t \in T_G \Leftrightarrow t \in S_G$ *and* $root(t) = S$*.*

A type grammar is a context free grammar $G = (N, T, P, S)$, such that $N = E$, $T = Char$. We denote its set of derivation trees D_G since we consider documents rather then general trees. Obviously it holds $D_G \subseteq D$. Nonterminals of a type grammar represent element names and the set of terminals equals to the text alphabet.

If we have a given type grammar, it generates a class of XML documents, which equals to the set of its derivation trees. *Validation* of XML document against a grammar is a process, which gives us as a result an answer, whether given XML document belongs to the class of documents generated by given grammar. If the answer is yes, we say that the XML document is *valid* for a given type grammar (or *correct*).

2.3 XML transformations

A transformation of an XML document takes some source documents as an input, it processes them according to the transformation specification and it gives target documents as an output.

It is reasonable to define a transformation of XML documents as a relation on the set of XML documents rather then a function [1]. As usual we begin with more general definition, i.e., tree transformation.

Definition 4. *Let Σ, Σ', Γ, Γ' be alphabets. A tree transformation from (Γ, Σ) to (Γ', Σ') is a relation $\tau \subseteq T_\Gamma(\Sigma) \times T_{\Gamma'}(\Sigma')$.*

If we apply previous definition on XML documents, we obtain restricted case again. Thus, *an XML documents transformation* is a relation $\tau \subseteq D \times D$. Obviously if we consider any XML document transformation, the source leaf alphabet equals to the target leaf alphabet.

2.4 Tree properties

Now we introduce several definitions related to trees that we will need later in this paper.

Let $\Sigma, \Sigma', \Gamma, \Gamma'$ be alphabets.

Let $X = \{x_1, x_2, \ldots\}$ be a set of *variable symbols*. For each $k > 0$, we denote by X_k a set of first k variables, thus $X_k = \{x_1, \ldots, x_k\}$. We denote by $T_\Gamma(\Sigma \cup X)$, $T_\Gamma(\Sigma \cup X_k)$ sets of trees with these variables, where it must hold $\Sigma \cap X = \emptyset$.

Definition of *a tree substitution* follows. Let $t \in T_\Gamma(\Sigma \cup X_k)$, $k > 0$ and $t_1, \ldots, t_k \in T_{\Gamma'}(\Sigma')$. Then $t[x_1 \leftarrow t_1, \ldots, x_k \leftarrow t_k]$ is a tree obtained from t by replacing each occurrence of x_i by a tree t_i, $1 \leq i \leq k$[2].

Let $Z = \{z_1, z_2, \ldots\}$ be a set of variables disjoint to X and $k > 0$. A set of (Γ, Σ, k)-*contexts* denoted $C_\Gamma(\Sigma, k)$ is the set of trees t from $T_\Gamma(\Sigma \cup Z_k)$, such that for each $1 \leq i \leq k$ the symbol z_i appears exactly once in t.

We will use a simpler notation for context substitution. Let t to be a (Γ, Σ, k)-context and $t_1, \ldots, t_k$ trees. Then we use $t[t_1, \ldots, t_k]$ instead of $t[z_1 \leftarrow t_1, \ldots, z_k \leftarrow t_k]$.

Let $t, t' \in T_\Gamma(\Sigma)$. Then t' is a *subtree* of t if there exists such a context $\beta \in C_\Gamma(\Sigma, 1)$ that $t = \beta[t']$.

For each tree $t \in T_\Gamma(\Sigma)$ $path(t) \subseteq \{1, 2, \ldots\}^*$ is a set of all *paths* of the tree t, so that each of them unambiguously identifies a node of the tree t. Then symbol ε represents $root(t)$. For each path $w \in path(t)$, we denote by $label_t(w)$ a label of the node identified by w and by $t|_w$ a subtree under this node.

Let $t \in T_\Gamma(\Sigma)$, $t' \in T_{\Gamma'}(\Sigma')$. We obtain tree $t[w \leftarrow_p t'] \in T_{\Gamma \cup \Gamma'}(\Sigma \cup \Sigma')$ from t by replacing subtree in $w \in path(t)$ by tree t'.

Let $t \in T_\Gamma(\Sigma)$, $patt \in T_\Gamma(\Gamma \cup \Sigma)$. We say, that t and *patt match*, if there is such a context $\gamma \in C_\Gamma(\Sigma, k)$ that $t = \gamma[t_1, \ldots, t_k]$ for some $t_1, \ldots, t_k \in T_\Gamma(\Sigma)$ and it holds $patt = \gamma[root(t_1), \ldots, root(t_k)]$.

[1] For example. in the system SynDoc [17], several target documents can be generated as the result of transformation and user can choose the most appropriate one.

[2] In the definition of tree substitution variables are not typed, however, later we will require newly connected subtrees to satisfy some specific conditions.

3 Classification hierarchy

There are several possibilities, how to divide XML document transformations into categories. According to the driving element of the transformation we obtain three basic categories as follows. In each of them different techniques for implementing particular transformation systems are used, thus, it is reasonable to start with this criterion.

1. *Source grammar transformations* - Transformation process is driven by the source structure. First, the source document is parsed and then according to recognized syntactic elements parts of the output documents are constructed. Usually it is possible to check correctness of input documents, but target correctness is not ensured in this case. If necessary, user must perform validation explicitly. Next we can divide this category into two more specific groups according to the way how the source document is parsed:
   - *Event-driven transformations* - The source document is read as a data stream. As the result, we obtain a list of recognized events. The event can be an occurrence of a start-tag, an end-tag, an attribute, etc. The transformation system reads the list o events and performs corresponding actions. If we allow side-effect functions, an output can be generated already at parsing time. As a formal model, an attribute grammar is often used. Event-driven models require little memory, and provide fast and effective way of accessing XML data. On the other hand, only simple transformations can be performed since it is not possible to return to an earlier part of the document. In some document transformers (e.g. CoST [12, 7], OmniMark [8]) the list of recognized events is represented by *ESIS (Element Structure Information Set)*, which is the part of ISO SGML standard [13]. However, currently especially SAX (Simple API for XML) [26] is more and more in use as event-driven mechanism for parsing XML documents. It generates specific SAX events and sends them to the application.
   - *Tree-based transformations* - First, an internal syntactic tree of the source document is constructed and a target document is generated via queries on this tree. In most of the cases transformation systems are based on tree pattern matching and replacement. We can assign widely used XSLT [29] and its predecessor DSSSL [14] as well as systems Scrimshaw [2], Metamorphosis [22], Balise [3] and TranSID [20] to this group.

     XSLT, a recommendation of W3 Consortium, has become very popular recently. It is a language, itself written in XML, with powerful capabilities for specifying transformations of XML documents. XSLT program (called stylesheet) consists of a set of template rules. A template rule associates a pattern, which matches nodes in the source document with corresponding template that can be instantiated to form the target tree. Although XSLT is a powerful transformation language, it has several drawbacks. Firstly, it appears to be a complex language and therefore

transformation specification must be written by an expert. Secondly, XSLT processor cannot guarantee the correctness of target documents as well as other systems in this group. However, an XSLT stylesheet can be produced as an output of some two-grammars systems (see next section). Then it is ensured, that generated stylesheet specifies a type transformation and XSLT processor can be used to perform transformation.

2. *Target grammar transformations* - Target document is created according to the rules of a given target grammar while relevant data from a source document are extracted via queries. This ensures target correctnes. However, we do not need to have any information about source document type. We found only one transformation system, which belongs to this group - TREX [30].
3. *Two grammar transformations (type transformations)* - Documents of a given source type are translated into documents of a given target type. Transformation systems are mostly based on one of these formal models - syntax directed translation schema, tree transformation grammar, descending tree transducer and higher order attribute grammar. We discuss these models as well as two grammar transformation systems in detail in the next section. However, there are few systems, which performed two-grammars transformations, but underlying formal model cannot be clearly recognized. These are Grif [25] and Thot [4].
   From a different point of view, we distinguish *static type transformations*, which translate whole documents, and *dynamic type transformations*, which are used to perform dynamic operations on XML documents. An example is *cut and paste* operation, which is a standard operation in XML document editors. Unlike in common editors, in this case it is necessary to preserve document structure. Thus, a local transformation must be performed in the place, where a part of another document has been pasted.

4 Type transformations

The group of type transformations (or two-grammar transformations) is a specific one. In this case there must be only correct documents taken as an input and correct documents must be generated on the output side. To satisfy these conditions, it is usually inevitable to implement transformation system performing this kind of transformations on some underlying formal model. In this section we introduce four formal models on which some of the implemented transformation systems are based. We developed common framework in which we define these models. Our intention was to create a formal base, so that the models can be studied and mutually compared later. We also present current results obtained by comparing some of them according to their transformational power. Obviously there is a direct proportion between the complexity of transformation specification and the set of possible transformations. Thus, the more powerful particular model is, the more complex specification the user is supposed to write.

4.1 Syntax-directed translation schema

Definition 5. *A syntax-directed translation schema - SDTS [1] is a 5-tuple* $\Omega = (N, \Sigma, \Delta, R, S)$*, where* N *is an alphabet of nonterminals,* Σ*,* Δ *are input and output alphabets,* $S \in N$ *is the start symbol and* R *is a finite set of rules of the form* $A \to \alpha, \beta$*, where* $\alpha \in (N \cup \Sigma)^*$*,* $\beta \in (N \cup \Delta)^*$ *and nonterminals in* β *are a permutation of nonterminals in* α*.*

In a rule $A \to \alpha, \beta$ with each nonterminal from α there is associated an identical nonterminal of β. If there is a multiple occurence of some nonterminal, we indicate association by using integer superscripts. Obviously if we extend definition of a SDTS by a renaming homomorphism for nonterminals, we can use this model also in the case, when it is neccesary to associate nonterminals with different names.

A context-free grammar $G_s = (N, \Sigma, P_s, S)$, where $P_s = \{A \to \alpha \mid \exists \beta \in (N \cup \Delta)^*, A \to \alpha, \beta \in R\}$ is a *a source grammar* and $G_t = (N, \Delta, P_t, S)$, where $P_t = \{A \to \beta \mid \exists \alpha \in (N \cup \Sigma)^*, A \to \alpha, \beta \in R\}$ is *a target grammar* of SDTS Ω.

Now we mentioned two modifications of SDTS, which differ from the basic model in a definition of rules:

• *Simple SDTS - SSDTS*

R is a finite set of rules of the form $A \to \alpha, \beta$, where $\alpha \in (N \cup \Sigma)^*$, $\beta \in (N \cup \Delta)^*$, nonterminals in α, β are identical and the order of nonterminals is preserved.

• *Extended SDTS - ESDTS*

R is a finite set of rules of the form $A \to \alpha, \beta$, where $\alpha \in (N \cup \Sigma)^*$, $\beta \in (N \cup \Delta)^*$ and nonterminals in β are a permutation of a subset of nonterminals in α.

A syntax-directed translation schema simulates derivations of two context-free grammars with similar set of rules simultaneously. While using a SDTS in the domain of syntax analysis it was sufficient to store only frontiers of the derivation trees in configurations, if we consider tree transformations it is more natural to keep the whole derivation tree. Then a transformation consists of pairs of trees, where the first one is a derivation tree of the source grammar and the second one is a derivation tree of the target grammar. We started from an algorithm for structured documents transformations using an ESDTS presented in [19]. We skip an operation of adding new nodes into the source tree since authors do not specify exactly what subtree is embedded to the newly created node.

Definition 6. *A configuration of SDTS (SSDTS, ESDTS)* Ω *is a tree from the set* $T_{N \cup \{\lceil\rceil\}}(\Sigma \cup \Delta)$*.*

Definition 7. *We say, that a pair of trees* $(A(t_1, \ldots, t_n), A(s_1, \ldots, s_m))$*,* $A \in N, t_1, \ldots, t_n, s_1, \ldots, s_m \in T_{N \cup \{\lceil\rceil\}}(\Sigma \cup \Delta)$ *realizes a rule* $A \to u_1 \ldots u_n, v_1 \ldots v_m \in R$ *if the following holds:*

1. $u_1 \ldots u_n = root(t_1) \ldots root(t_n)$ *and* $v_1 \ldots v_m = root(s_1) \ldots root(s_m)$*,*

2. $h_N(u_1 \dots u_n) = u_{i_1} \dots u_{i_r}$ *and* $h_N(v_1 \dots v_m) = v_{j_1} \dots v_{j_p} = u_{i_{k_1}} \dots u_{i_{k_p}}$, $i_1, \dots, i_r \in \{1, \dots, n\}$ *(different),* $j_1, \dots, j_p \in \{1, \dots, m\}$ *(different),* $k_1, \dots, k_p \in \{1, \dots, r\}$ *(different),*
3. $s_{j_l} = t_{i_{k_l}}$ *for* $l \in \{1, \dots, p\}$, $s_l = v_l$, $v_l \in \Delta \cup \{\varepsilon\}$ *for* $l \notin \{j_1, \dots, j_p\}$ *and* $t_l = u_l$, $u_l \in \Sigma \cup \{\varepsilon\}$ *for* $l \notin \{i_1, \dots, i_r\}$.

Definition 8. *A translation step of SDTS (SSDTS, ESDTS)* Ω *is a relation* $\Rightarrow_\Omega$ *over the set of configurations defined as follows:*

1. $\beta[\restriction(a)] \Rightarrow_\Omega \beta[a]$, $a \in \Delta \cup \{\varepsilon\}$.
2. $\beta[\restriction(A(t_1, \dots, t_n))] \Rightarrow_\Omega \beta[A(\restriction(s_1), \dots, \restriction(s_m))]$ *and a pair of trees* $(A(t_1, \dots, t_n), A(s_1, \dots, s_m))$ *realizes some rule* $r \in R$.

In both cases β *is a context from* $C_{N \cup \{\restriction\}}(\Sigma \cup \Delta, 1)$.

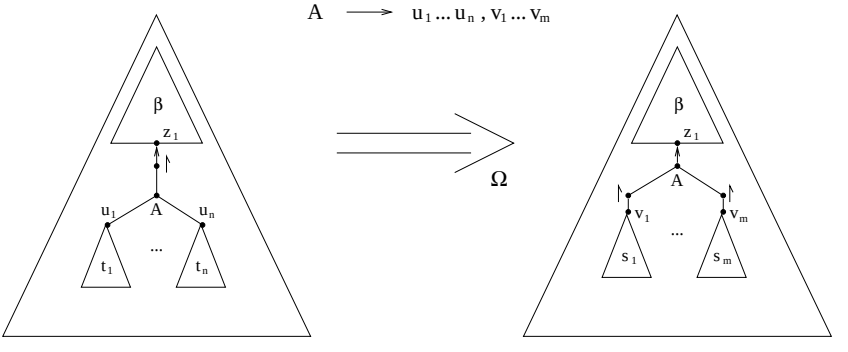

Fig. 1. A translation step of a SDTS

An input tree is processed from the root to the leaves, while unprocessed subtrees are marked by symbol $\restriction$. Initially, the whole source tree is marked as unproccessed. If the root of a currently processed subtree has a label from output alphabet or ε, then it is a leaf and we stop translation in this branch by removing the symbol $\restriction$. If we are processing a subtree, whose root is an internal node, first we reorder and delete children subtrees according to corresponding rule. Subsequently, if there are some leaf childrens, we reorganize them in such a way, that the result adheres to the output side of the rule.

Definition 9. *A transformation generated by a SDTS (SSDTS, ESDTS)* Ω *is a set* $\tau(\Omega) = \{(t_s, t_t) \mid t_s \in T_N(\Sigma), root(t_s) = S, t_t \in T_N(\Delta), \restriction(t_s) \Rightarrow_\Omega^* t_t\}$.

A SSDTS enables very simple transformations of trees. It does not change structure of the source tree, it is only possible to delete, add or reorder leaves. A SDTS enables reordering of subtrees connected to the same node moreover. An ESDTS is the most powerful modification, unlike SDTS it enables also deleting of arbitrary subtree in the source tree.

A transformations performed by SSDTS as well as SDTS satisfy the definition of the type transformation. In the case of ESDTS a problem arises, because if

we delete a subtree in the source tree, it is not processed anymore. Thus, it is not possible to check, whether this part of the source tree was correct.

There are several implemented transformation systems that are using SDTS and its modifications as underlying model. System SynDoc [19] is based on ES-DTS, and SDTT [5], ICA [21], HST [16] are based on SDTS.

4.2 Tree transformation grammar

A tree transformation grammar is similar to SDTS, however in this case we associate two groups of production rules. The first group - *a source subgrammar* contains some of the source grammar production rules and the second one - *a target subgrammar* conatins some of the target grammar production rules. Unlike in SDTS, nonterminals in the source and target subgrammar must be associated explicitely. This means, that it is possible to associate also nonterminals with different names. Additionally, tree-transformation grammars work with as many levels in the source tree as necessary.

Definition 10. *A tree transformation grammar - TTG is a 6-tuple* $G = (G_s, G_t, Sub_s, Sub_t, PA, SA)$*, where*

- $G_s = (N_s, \Sigma_s, P_s, S_s)$ *and* $G_t = (N_t, \Sigma_t, P_t, S_t)$ *are the source and target grammars,*
- $Sub_s \subseteq 2^{P_s}$ *and* $Sub_t \subseteq 2^{P_t}$ *are sets of source and target subgrammars,*
- $PA \subseteq Sub_s \times Sub_t$ *is a set of production group associations,*
- $SA \subseteq N_s \times N_t$ *is a set of symbol associations.*

There are some restrictions put on the source subgrammar. It must contain a single start nonterminal, and every other nonterminal in the source subgrammar must be derivable from this start nonterminal. Source subgrammars represent subtree patterns to be matched against in the source tree; target subgrammars represent subtrees, that are to be generated as part of the target tree. A target subgrammar can contain several start nonterminals, thus we can obtain a forest of target subtrees as a result. Transformation via TTG is performed by processing source tree nodes one by one. For a particular node we look for matching source subgrammars. If there is some, corresponding target subtrees are constructed and links among nodes in source subgrammar and target subgrammar are marked according to symbol associations. After the whole source tree has been processed, target subtrees can join to bigger trees according to the marked symbol associations.

Tree transformation grammars are described in [15]. Authors' intention was to define a powerful two-grammar transformations model and, at the same time, provide simple and natural way to write transformation specification.

Several modifications of tree transformation grammars have been implemented:

- *Dual grammar translation scheme (DGTS)* - system SSAGS [24].
- *Single input production-explicitely qualified (SIPEQ)* - system SIPEQ [15].
- *TT grammar* - system Alchemist [20].

However, semantics of the model remains unclear as no formal definitions have been introduced in any of the resources mentioned. Now we present definitions, that we created according to the alghorithm of tree transformation via TT grammars mentioned in [20].

Definition 11. *A configuration of TTG Ω is a pair (t_s, T_t), $t_s \in T_{N_s \cup \{\restriction\}}(\Sigma_s)$ and $T_t \subseteq \{(t, W) \mid t \in T_{N_t}(N_t \cup \Sigma_t), W \subseteq 2^{(path(t) \times path(t))}\}$.*

A tree t_s is the source tree with marked unprocessed nodes. T_t is *a set of target linked trees*, i.e., it contains pairs of the form (t, W), where t is a particular target tree and W is a set of path pairs that we call *a set of links of a tree t*. We use the notation W_t as well.

Definition 12. *A translation step of TTG Ω is a relation $\Rightarrow_\Omega$ over configurations defined by $(t_s, T_t) \Rightarrow_\Omega (t'_s, T'_t) \iff$*

(1) generating step

- *there is such a path w_s in a tree t_s that $t_s|_{w_s} = \restriction (A(t_1, \ldots, t_n))$,*
- *$t'_s = t_s[w_s \leftarrow_p A(\restriction (t_1), \ldots, \restriction (t_n))]$,*
- *there is such a source subgrammar $sg \in Sub_s$ that t_{sg} (a derivation tree corresponding to the rules sg) and $A(t_1, \ldots, t_n)$ match,*
- *there is such a target subgrammar $tg \in Sub_t$ that $(sg, tg) \in PA$ and $T_{tg} = \{r_1, \ldots r_m\}$ (derivation trees corresponding to the rules tg). Let us denote $R_i = (r_i, \{(w_1, w_s w_2) \mid (label_{r_i}(w_1), label_{t_{sg}}(w_2)) \in SA\})$, $i \in \{1, \ldots, m\}$. Then $T_{t'} = T_t \cup \bigcup_{1 \leq i \leq m} R_i$.*

(2) connecting step

- *$t'_s = t_s$,*
- *there are such $R_1, R_2 \in T_t$, $R_1 = (r_1, W_1)$, $R_2 = (r_2, W_2)$ that*
  - *$root(r_2) = A \in N_t$,*
  - *there is such a path $w \in path(r_1)$ that $r_1|_w = A$,*
  - *there is such a path $w_s \in path(t_s)$ that $(w, w_s) \in W_1$ and $(\varepsilon, w_s) \in W_2$.*

  Let $r_3 = r_1[w \leftarrow_p r_2]$, $W_3 = W_1 \cup \{(ww_1, w_2) \mid (w_1, w_2) \in W_2\}$ a $R_3 = (r_3, W_3)$. Then $T_t = T_{t'} - R_1 - R_2 \cup R_3$.

Transformation works in similar way as with SDTS. The source tree is passed in the top-down manner, while unprocessed nodes are marked by symbol $\restriction$, initially the whole source tree is marked.

During the generation step, we first choose such a source subgrammar (pattern), that matches a subtree under currently processed node (Fig. 2). The model is nondeterministic, i.e., there can be more suitable source subrammars. Then we add target subtrees (corresponding to the associated target subgrammar) together with links to the set of linked target trees. Links are created according to symbol associations (Fig. 3). At last we remove symbol $\restriction$ from current subtree and again we mark all connected subtrees to indicate they need to be processed.

If we apply connecting step (Fig. 4), we first choose two subtrees r_1 and r_2 from the set of target linked trees. These trees must satisfy some conditions;

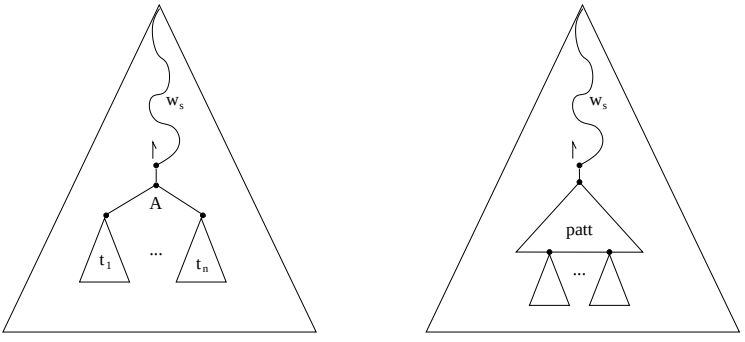

Fig. 2. Different views of the same source tree

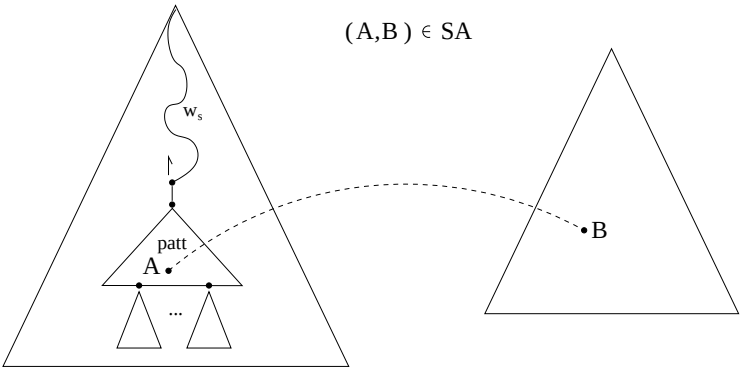

Fig. 3. New links created by the generation step

the root of the first one and a leaf of the second one must have the same label and must be linked to the same node in the source tree. Consequently, the set of target linked trees is updated so that subtrees r_1, r_2 are removed and a new subtree created by connection of r_1, r_2 is added. Links between this new subtree and the source tree will be updated as well according to the links in old subtrees. However, we want to obtain one tree as a result of transformation. Therefore in following definition of tree transformation via TTG we require the last configuration to consist of a single element.

Definition 13. *A transformation generated by TTG Ω is a set* $\tau(\Omega) = \{(t_s, t_t) \subseteq T_{N_s}(\Sigma_s) \times T_{N_t}(\Sigma_t) \mid (\restriction (t_s), \emptyset) \Rightarrow^*_\Omega (t_s, \{(t_t, W)\})\}$.

4.3 Descending tree transducer

A descending tree transducer is an automaton, which passes a source tree from the root to the leaves and performs changes according to the current state, node

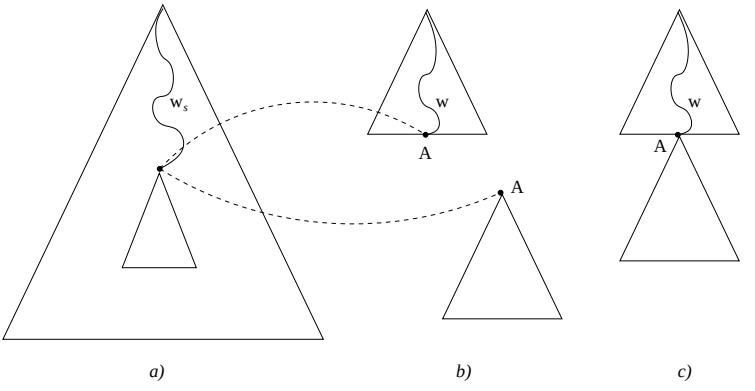

Fig. 4. Linking trees by connecting step a) source tree b) target trees before the connecting step c) new target tree

and lookeahead. Unlike previous formal models, it works with trees, where labels of internal nodes are from a ranked alphabet. Basically, ranked alphabet is a pair $(\Gamma, rank)$, where Γ is an alphabet and $rank : \Gamma \to N$ is a ranking function, which assigns positive integer to each symbol in Γ. We usually omit the function $rank$ in the notation and we say that Γ is a ranked alphabet. If Σ is an alphabet and Γ a ranked alphabet, then each internal node in a tree over (Γ, Σ) must have exactly as many children as is the value of the rank of its label. We will not introduce formal definitions here, all of them can be found in [10] and the notation is very similar to that one used in previous cases.

A DTT does not represent a model for two-grammars transformations since it works on higher level of abstraction. However, it is easy to restrict definitions in such a way, that we obtain DTT, which transforms derivation trees of a given source grammar into derivation trees of a given target grammar.

A DTT with finite and regular lookahead has been used in the transformation system SynDoc [17]. This system generates also an XSLT stylesheet as an output and it is possible to use existing tools for XSLT if further processing is needed. However, in that case the target correctness is not guaranteed anymore.

4.4 Higher order attribute grammar

Basically, *an attribute grammar* is a context free grammar, such that each rule is associated with a set of semantics rules. These rules have a form $X.b := f(Y_1.c_1, \ldots, Y_n.c_n)$, where X, Y_i are nonterminals, b, c_i are attributes and f is an n-ary function.

All definitions presented in this section results from [27].

Definition 14. *An attribute grammar - AG is a triple* $AG = (G, A, R)$*, where*

- $G = (N, \Sigma, P, S)$ *is a context-free grammar,*

- $A = \bigcup_{X \in N \cup \Sigma} AIS(X)$ *is a finite set of attributes, where* $AIS(X)$ *is a set of attributes associated with nonterminal* X*,*
- $R = \bigcup_{p \in P} R(p)$ *is a finite set of attribute rules. If* $AIS(X) \cap AIS(Y) \neq 0$*, then* $X = Y$*. For every occurrence of a nonterminal in a derivation tree of* G *there must be exactly one attribute rule applicable for computation of a value for an attribute* $a \in A$*. Rules in* $R(p)$ *have a form* $\alpha = f(\ldots, \gamma, \ldots)$*, where* f *is a name of the function,* α *and* γ *are attributes of a form* $X.a$*.*

Definition 15. *For each* $p : X_0 \rightarrow X_1 \ldots X_n \in P$ *we define a set of attribute evaluation occurrences as* $AF(p) = \{X_i.a \mid X_i.a = f(\ldots) \in R(p)\}$*. An attribute* $X.a$ *is called a synthetized attribute if there is a rule* $r : X \rightarrow \chi$ *and* $X.a \in AF(r)$*; an inherited attribute, if there is a rule* $q : Y \rightarrow \mu X \nu$ *and* $X.a \in AF(q)$*.*

We denote by $AS(X)$ a set of synthetized attributes of a nonterminal X and by $AI(X)$ a set of inherited attributes of a nonterminal X. After evaluation process these attributes contain a subtree as a computed value. Thus, unlike in AG, in HAG the domain of the derivation tree and the domain of attributes overlap.

A higher-order attribute grammar (HAG) is an extended attribute grammar. A special type of attribute - *nonterminal attribute* is introduced.

Definition 16. *For each* $p : X_0 \rightarrow X_1 \ldots X_n \in P$ *a set of nonterminal attributes is defined:* $NTA(p) = \{X_j | X_j := f(\ldots) \in R(p)\}$*.*

An unevaluated nonterminal attribute represents a leaf node of the actual tree. At this point we call it *virtual nonterminal*, since there is no subtree connected to it. After the evaluation is performed, a subtree is computed as a value for nonterminal attribute and consequently it is connected to this attribute (leaf node). Evaluated nonterminals and common nonterminals we also call *instatiated nonterminals* since they have got an instance, i.e. connected subtree. The set of instantiated nonterminals represents internal nodes of the actual tree.

A HAG has been implemented in transformation system SIMON [9], which takes advantage of the power of this model and enables several complex transformations. However, there is one major drawback. System requires a user to write a complete HAG as a transformation specification and this is usually a nontrivial task.

For HAG there has been defined formal model - *a higher-order attribute tree transducer* [23], which represents some abstraction and therefore it is more suitable for studying its properties.

4.5 Comparison results

Finally, we briefly introduce results obtained by comparing formal models for type transformations according to their transformational power. Since we defined transformations as relations over sets of trees, we can consider them as sets containing pairs of trees as elements. Thus, we can also compare them as sets. See Table 1 for comparison results. Symbol N indicates that transformations of two corresponding formal models are not comparable. More details and all formal proofs can be found in [6].

Table 1. Comparison results

	SDTS	**ESDTS**	**d-DTT**	**DTT**
SDTS		$\subsetneq$	$\not\supseteq$	$\subsetneq$
ESDTS	$\supsetneq$		N	N
d-DTT	$\not\subseteq$	N		$\subseteq$
DTT	$\supsetneq$	N	$\supseteq$	

SDTS: syntax directed translation schema, ESDTS: extended syntax directed translation schema, d-DTT: deterministic descending tree transducer, DTT: descending tree transducer

5 Conclusion and future work

In this paper we introduced a system for classification of XML documents transformations. Our intention was to simplify choosing of appropriate transformation system according to given requirements for transformation. We defined categories in bottom-up way, i.e. first we were examining implemented transformation systems and consequently we abstracted from them to higher-level groups of transformations sharing similar features.

We were examining in detail formal models used for type transformation. We defined common framework for these models and we introduced current results of their mutual comparison. In our future work, we plan to make more comparisons according to transformational power and complexity as well. We intend to include also other models, which might appear to be suitable for XML document transformations.

References

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., USA, 1972.
2. D. S. Arnon. Scrimshaw: A language for document queries and transformations. *Electronic Publishing, 6(4)*, 1993.
3. Berger-Levrault/AIS: *Balise Reference Manual, Release 3*, 1996.
4. S. Bonhomme. *Transformation de documents structurés, une combinaison des approches explicite et automatique.* PhD thesis, University Joseph Fourier - Grenoble, 1998.
5. K. Chiba and M. Kyojima. Document transformation based on syntax-directed tree translation. *Electronic Publishing, 8(1)*, 1995.
6. J. Dvořáková. *Transformácie XML dokumentov.* Master thesis, FMFI UK Bratislava, 2004.
7. Joe English. *CoST 2 Reference Manual. Release 3*, 1996. `http://www.art.com/cost/manual.html`.
8. Exoterica Corporation. *OmniMark Programmer's Guide*, 1993.
9. A. Feng and T. Wakayama. SIMON: A grammar-based transformation system for structured documents. *Electronic Publishing, 6(4)*, 1993.

10. Z. Füllöp and H. Vogler. *Syntax-directed semantics: Formal models based on tree transducers*. Springer-Verlag, 1998.
11. V. Hambálková. *Transformácie štruktúrovaných dokumentov*. FMFI UK Bratislava, 2000.
12. K. Harbo.*CoST Version 0.2 - Copenhagen SGML Tool*. Department of Computer Science and Euromath Center, University of Copenhagen, 1993.
13. ISO - International Organization for Standardization. *Information Processing - Text and Office Systems - Standard Generalized Markup Langugage (SGML), ISO 8879*, 1986.
14. ISO - International Organization for Standardization. *Information Technology - Text and Office Systems - Document Style Semantics and Specification Language (DSSSL), ISO/IEC DIS 10179:1996*, 1996.
15. S. Keller, J. A. Perkins, T. F. Payton and S. P. Mardinly. Tree transformation techniques and experience. *Proceedings of the ACM SIGPLAN '84 Symposium on Compiler Construction, SIGPLAN Notices, 19(6)*, Montreal, Canada, 1984.
16. P. Kilelinen, Greger Lindén, H. Manilla and E. Nikunen. A structured document database system. *EP90 - Proceedings of the International Conference on Electronic Publishing, Document Manipulation and Typography, Gaithersburg, Maryland*, Cambridge University Press, 1990.
17. E. Kuikka, P. Leinonen and M. Penttonen. Towards automating of document structure transformation. *DocEng '02*, 2002.
18. E. Kuikka and E. Nikunen. *Survey of software for structured text.* Technical report, Department of Computer Science and Applied Mathematics, University of Kuopio, 1998.
19. E. Kuikka and M. Penttonen. Transformation of structured documents. *Electronic Publishing, 8(4)*, 1995.
20. G. Lindén. *Structured document transformations*. PhD thesis, University of Helsinki, 1997. ISBN 951-45-7766-3.
21. S. A. Mamrak, C. S. O'Connel and J. Barnes. *Integrated Chameleon Architecture*. Prentice Hall, Englewood Cliffs, USA, 1994.
22. MID/Information Logistics Group GmbH. *MetaMorphosis Reference Manual*, 1995.
23. T. Noll and H. Vogler. The universality of higher-order tree transducers. *Theory of computing systems, 34(1)*, 2001
24. T. F. Payton. SSAGS: A syntax and semantics analysis and generation system. *Attribute Grammars. Definitions, Systems and Bibliography. Lecture Notes in Computer Science 323*, Springer-Verlag, Berlin, 1988.
25. V. Quint and I. Vatton. GRIF: An interactive system for structured document manipulation. *EP 86 - Proceedings of the International Conference on Text Processingand Document Manipulation, Nottingham, UK*, Cambridge University Press, 1986.
26. *SAX - Simple API for XML, version 2.0.2*, 2004. `http://www.saxproject.org/`.
27. D. Swierstra and H. Vogt. *Higher Order Attribute Grammars*. Technical report. Utrecht University, 1991.
28. W3C. *Extensible Markup Language (XML) 1.0 (Third edition), W3C Recommendation*, 2004. `http://www.w3.org/TR/REC-xml`.
29. W3C. *XSL Transformations XSLT Version 2.0, W3C Recommendation*, 1999. `http://www.w3.org/TR/xslt20/`.
30. A. Zhou, Q. Wang, Z. Guo, X. Gong, S., Zhengand H. Wu, J. Xiao, K. Yue and W. Fan. TREX: DTD-conforming XML to XML transformations. *SIGMOD 2003*, 2003.