An interactive approach to rule–based transformation of XML documents[*]

Marek RŮŽIČKA, Vojtěch SVÁTEK

Department of Information and Knowledge Engineering,
University of Economics, Prague
Nám. W. Churchilla 4, 130 67 Praha 3, Czech Republic
ruza.m@volny.cz svatek@vse.cz

Abstract. Transformation of XML documents is typically understood as non-interactive. In contrast, we formulate the specific task of XML–based transformation of knowledge contained in semi–formal documents, which heavily depends on human understanding of element content and thus requires frequent user intervention. Yet, many aspects of this process are pre-determined, and their automation is highly desirable. We implemented a software tool (called Stepper) supporting interactive step–by–step transformation of ‚knowledge blocks'. The transformation is governed by rules expressed in a new 'interactive transformation' language (called XKBT), while its non–interactive aspects are handled by embedded XSLT rules.

Keywords: XML, transformation, document content.

1 Introduction

Abundant use of XML-based *mark–up* languages in the last years obviated the need for a related but distinct type of languages: those defining *transformations* from one mark–up language to the other [1, 2, 8] or from a mark–up language to a different (e.g. relational) representation and vice versa [2, 4]. Transformations are typically *content–independent* (or, merely depend on syntactical properties of the content), and thus can be performed fully automatically. In this paper, in contrast, we consider the situation when the transformation result depends on semantic (machine–undetectable) distinctions of marked–up *content*. The transformation thus has to rely on co–operation between an interactive software tool and a human user.

A prototypical situation arose in our original domain of interest: *computerised clinical guidelines*, where the need for transformation of mark–up structures emerges in two distinct flavours. First, the original free text of the guideline document (defining the recommended course of actions to be undertaken by the physician if a particular disease is encountered with the patient) has to be converted to a more formal representation [7]. Second, several formal models (with existing XML syntax) of computerised guidelines have been developed, and knowledge has to be transferred

[*] The research has been partially supported by project no. 201/00/D045 of the Grant Agency of the Czech Republic, 'Construction of knowledge models in connection with text documents'.

between them (for a comparative overview of guideline models, see [9]). In both situations, ambiguities arise that have to be resolved by human user, often even by a clinical expert. Yet, many sub–processes can be automated, and even for the remaining ones, the user should not have full freedom to manipulate the XML structures. Support for interactive but foreseeable transformation is thus needed.

The structure of the paper is as follows. In section 2 we formulate the principles of our approach to interactive 'knowledge block transformation'. In section 3 a concrete transformation language (XKBT) is proposed and illustrated on a complex example. Section 4 describes an implemented tool (Stepper) that conforms to the principles and uses the XKBT language. Finally, section 5 summarises the whole approach.

2 Principles of stepwise interactive transformation of knowledge blocks

The key principle, previously formulated in [7] for the particular case of medical guideline formalisation, is explicit separation of *partial transformation steps*, the input and output of which conforms each to a distinct *schema language*. This implies that interventions of the user into the transformation process take place in the context of a given step, and are constrained by the output language. Each step is carried out via *rules* from a certain *rule set*. The role of user is to repeatedly select the *input* 'knowledge block' to be processed, the *rule* to be applied on it, and the way the *output* 'knowledge block' is assembled.

The rest of this section consists of formal definitions framing our approach. Due to limited space, some definitions are however 'sloppy' and refer to notions explained informally in section 3, which is devoted to the XKBT transformation language.

Definition 1. A *knowledge block* is either *atomic* or *compound*.
An *atomic knowledge block* is an XML element *e* together with its (arbitrary) tree structure of sub–elements; *e* is denoted as *top element* of the knowledge block.
A *compound knowledge block* is a sequence of two or more atomic knowledge blocks; the sequence of top elements of these knowledge blocks is denoted as *top sequence* of the compound knowledge block.

The use of adjective 'knowledge' reflects the implicit assumption of our approach: XML documents under consideration should be 'rich in knowledge' interpretable by humans. The term is essentially used for compatibility with previous, less formal papers; it is likely to be replaced by a more canonical one in the future.

Definition 2. A *transformation rule* is a triple r = (*SD*, *TD*, *Type*), where

- *SD* is a source block definition
- *TD* is a target block definition
- *Type* is one of predefined rule types.

We will discuss block definitions and rule types in sections 3.1 and 3.2.

Definition 3. A *transformation suite* is a pair (L, S), where

- $L=\{dtd_1, dtd_2, \ldots, dtd_n\}$; each dtd_i is a Document Type Definition (DTD)[1]
- $S=\{rs_{1,2}, rs_{2,3}, \ldots, rs_{n-1,n}\}$; each $rs_{i,i+1}$ is a set of *transformation rules.*

Namely, there are n definitions of 'levels' corresponding to DTDs, and $n-1$ definitions of transformation 'steps' corresponding to rule sets. Each 'step' definition pertains to two subsequent 'level' definitions and should be consistent with them: if a rule r=(SD, TD, $Type$) belongs to $rs_{i,i+1}$ then SD should conform to dtd_i and TD to dtd_{i+1}, in order for r to be properly applicable.

Next, we will shift from the *definition* of the transformation to elements of the actual transformation *process*.

Definition 4. A *transformation act* is a 5–tuple (Src, Sel, r, $DestIn$, $DestOut$), where

- Src is an XML document conforming to dtd_k (from the given transformation suite, same applies below)
- $Sel = (e_1, e_2, \ldots, e_n)$, denoted as *selection*, is an ordered set of elements from Src such that for each e_i, e_j, the node corresponding to e_j follows the node corresponding to e_i in XML tree
- r = (SD, TD, $Type$) is a rule from ruleset $rs_{k,k+1}$; SD is satisfied by the knowledge block with top sequence Sel
- $DestIn$ is an XML document conforming to dtd_{k+1}; it consists[2] of a sequence of m atomic knowledge blocks
- $DestOut$ is an XML document conforming to dtd_{k+1}; it consists of a sequence of $m+r$ atomic knowledge blocks: the first m blocks are the same as in $DestIn$, and the last r blocks form a knowledge block that satisfies TD.

Simply said, rules that append new 'derivations' to *DestIn* are activated according to 'selection' in *Src*. The selection is assumed to be flat, i.e. not covering a (proper) ancestor–descendant relationship; it may however contain elements from different levels of the input document. The mechanism how blocks satisfy definitions will be informally sketched in section 3.1, for a concrete language (XKBT).

Definition 5. A *transformation step* is a sequence of transformation acts, (ta_1, ta_2, ... , ta_n), such that for the individual ta_i = (Src_i, Sel_i, r_i, $DestIn_i$, $DestOut_i$) holds:

- all r_i belong to the same rule set $rs_{k,k+1}$
- $Src_1 = Src_2 = \ldots = Src_n$ (let us denote it further as Src) conforms to dtd_k
- all $DestIn_i$, $DestOut_i$ conform to dtd_{k+1}
- $DestIn_1 = \theta$; $DestIn_i = DestOut_{i-1}$ *holds for every* i, $1<i \leq n$

Namely, a transformation step contains transformation acts that apply the same set of rules on the same source document and successively append new knowledge blocks to the destination document.

[1] Or, possibly an XML Schema (or similar) document.

[2] Apart from XML header and document 'envelope'; the same applies to the following item.

Definition 6. Let $Sel^* = Sel_1 \cup Sel_2 \cup ... \cup Sel_n$, where Sel_1, Sel_2, ... Sel_n are selections of all transformation acts of transformation step s. Then s is *complete* if and only if for every element $e \in Src$ exists an element $e' \in Sel^*$ such that e' is ancestor of e in *Src*.

Completeness of a transformation step means that the selections cover the whole input document, i.e. no information was lost 'along the way'. This does not entail that all elements have their counterparts in the subsequent level: they can be 'forgotten' if not needed, but only by means of explicit 'dead–end' rules. On the other hand, selections from the same transformation step may overlap or even be subsumed.

3 The XKBT transformation language

The formal model itself imposes no requirements on the syntax and semantics of transformation rules. However, in implemented tools, it has to be defined by means of a concrete (desirably, simple) language. We developed and implemented a prototype of such language; we call it XKBT, for "eXtensible Knowledge Base Transformation". The typology of XKBT rules has been presented in more detail in [6], and full syntax and semantics of the language can be found at the website[3] [5]. Here we only summarise the crucial points: the nature of source/target *knowledge blocks* (incl. the way they are evaluated), the *types* of rules, and relationship with the *XSLT* language. A comprehensive example is shown in the end of the section.

3.1 Knowledge block definitions and their evaluation

We syntactically and semantically distinguish between the *source block definition* and the *target block definition* of a transformation rule. What they have in common is their recursive structure: a *compound source block definition* may consist of either *elementary* or *compound* source blocks definitions (elements `<source>` and `<compound-source>`, respectively), similarly, a *compound target block definition* may consist of either *elementary* or *compound* target blocks definitions (elements `<target>` and `<compound-target>`, respectively). The operational semantics however differs: the source block definition is matched with the *selection* of the given transformation act (cf. Definition 4), while the target block is generated on *output* when the rule fires. The source block definition may also contain 'control' elements, `<cond>` and `<copy>`, which will be discussed later.

The 'way of composition' of compound blocks is determined by a number of interdependent attributes. The following attributes can be used in a *source block*:

- *type*: values 'iteration', 'optional', 'selection', 'sequence'
- *minOccurs*, *maxOccurs*, *occurs* (all integer–valued)
- *order*: values 'free', 'fixed'
- *conditions*: 'all', 'any', 'none'.

[3] The latest version of the language (2.1) referred to in this paper slightly differs from the version 2.0 implemented in the Stepper tool (section 4). The differences however do not affect the core of the method.

If the type is ‘iteration’, repetition of same elements is expected in *Sel*; the number of elements is determined by *minOccurs*, *maxOccurs* (lower/upper bound) or *occurs* (exact number). Type ‘optional’ is shortcut for repetition of zero or one element. Type ‘selection’ refers to alternative elements to be matched in *Sel* (i.e. ‘disjunctive condition’). Finally, type ‘sequence’ refers to elements that must all be present in *Sel* (i.e. ‘conjunctive condition’), in the predefined or free order (depending on *order*).

While the above mentioned attributes represent simple syntactical constraints similar to definitions in DTD or XML Schema, the *conditions* attribute relates to the evaluation of more complex conditions over the candidate source block. The conditions themselves are represented as `<cond>` sub–elements of the given `<source>` element, and mostly relate to the XML environment of the given element (ancestors, descendants, attributes etc.). Their semantics thus rather resembles that of XPath expressions. A `<source>` element may also be equipped with a `<copy>` element, which defines how (and which) sub–elements and attributes of the given source element transcend to the target element. The commented example in section 3.4 gives a clearer idea on the overall outlook of source block definition.

The definition of *target block* is simpler, since the attribute *conditions* and the sub–elements `<cond>` and `<copy>` would not make sense therein. The attributes *type*, *minOccurs*, *maxOccurs* and *occurs* may appear in target block definition, and their values may have the same ranges as for source block. Their semantics is now however specific: it relates to the *interaction* with the user. S/he is allowed to set up the number of repetitions (within the *minOccurs* and *maxOccurs* limits), choose from ‘selection’, or decide about an ‘optional’ element. A compound block defined as ‘sequence’ is processed non–interactively. For ‘sequence’, there is again an associated *order* attribute, with values ‘original’ and ‘fixed’; the former keeps the elements in the original order (assuming it is consistent with the target DTD, and ‘free’ order was allowed in source), while the latter enforces the order given in the definition.

3.2 Transformation rule types

The following types of rules (cf. Definition 2) have been devised in XKBT up to now:

- ‘One–to–one’ rule, mapping a (source) atomic knowledge block on a (target) atomic knowledge block. For the convenience of both designer and user of the rules, the target definition may involve an explicit alternative (elements to be chosen from); this is equivalent in effect to multiple rules.
- ‘Decomposition’ rule, mapping a (source) atomic knowledge block on a (target) non–atomic knowledge block.
- ‘Aggregation’ rule, mapping a (source) non–atomic knowledge block on a (target) atomic knowledge block.
- ‘Dead–end’ rule, applicable on a (source) atomic or non–atomic knowledge block, which is not needed at the subsequent level.

It is important to say that the current repertory of rules is merely tentative (though intuitive). Its main role was to evaluate the functionality of the *Stepper* tool described in section 4. More thorough analysis (both theoretical and empirical, based on practical use) will probably lead to additions and revisions in the subsequent versions of XKBT. The choice of rule types will primarily be guided by the need of user interaction, and by complementarity with XSLT rules, cf. section 3.3.

3.3 XKBT and XSLT

XSLT (XSL Transformations) is a 'canonical' XML–based transformation language [2]. Similarly to XKBT, it is based on rules, it however does not allow for *user interaction* either in processing the *source knowledge blocks* or in building the *target knowledge block*. Let us clarify the differences of both languages in more detail.

An XSLT–based transformation is activated once for the *whole input document*, whose elements are systematically, one–by–one (starting from the root), matched against the transformation rules. When more rules are suitable for the same element or element sequence, one of them is selected automatically according to pre-defined priorities. In contrast, the starting point for an XKBT–based transformation act is a *selection* of source knowledge blocks by the user (cf. Definition 4); the transformation is thus only fired within local scope. The user may select *distant parts* of source text as adjacent knowledge fragments for the output model; this conforms to the experience that related pieces of knowledge are often split apart in text. Next, if multiple rules match in XKBT, the user can select one or more based on his/her intellectual interpretation of source block content.

Furthermore, XSLT for the same input always yields the same *output*. In contrast, after firing one of XKBT rules, the user can specify how the *target block* will be constructed. The target block has to fulfil the constraints defined in the `<compound-target>` element; yet, there is room for modifications.

Another problem of XSLT is its *complexity*, resulting in relative opacity of larger rules, while XKBT is readable even for users with limited XML experience. Note that a set of XKBT rules is typically dedicated to a specific domain (e.g. subset of medicine), and its preparation thus requires participation of domain expert.

XKBT is however not replacement for XSLT, since its representational power is limited. Instead, we use both in a complementary fashion. While XKBT is tuned for the interactive part of transformation, non–interactive aspects are handled by *embedded XSLT rules*[4]. The difference from traditional use of XSLT is in separate use of short XSLT files yielding individual target knowledge blocks (in contrast to application of one complex XSLT file on the whole document). In this scenario, a user has (beside the composition of target knowledge block) the possibility to choose one of XSLT transformations for each final target knowledge block (for details see next section). This situation is likely to arise when transforming knowledge from one XML–grounded *formal model* (e.g. a medical guideline model) to another: the user has to semantically analyse the content and choose the right counterpart in the second model; the necessary syntactical structure is then generated automatically.

The situation is still different if we wish to export the knowledge content to a non–XML format, as often happens in the last step of *free–text document* (such as medical guideline) formalisation. In this scenario, the user typically converts the text (in several steps, using XKBT rules) to a well–structured XML knowledge base, and then 'exports' this knowledge base *as a whole* via XSLT into the final format.

[4] Embedded XSLT rules have to be designed by an XML expert; they typically deal with routine transformations, which do not require involvement of domain expert.

3.4 Commented example

Let us demonstrate the principles of XKBT on a simple example from the medical domain, namely on transformation of the knowledge block representing an elementary ‘goal’ of hypertension treatment. We will subsequently show and explain the codes of *source block*, an applicable *XKBT rule*, an embedded *XSLT rule*, and, finally, the resulting *target block*. Note that even the source block is assumed to have arisen by (XKBT–based) formalisation of the original free text. The meaning of the block is “antihypertensive treatment should lead to significant lowering of blood pressure” (otherwise the treatment is ineffective and the goal is not achieved):

```
Source knowledge block:
<goal direct="no" overall="no" id="g2" >
  <goal-of>
   <activity type="treatment"> antihypertensive treatment
   </activity>
  </goal-of>
  <is-goal> blood pressure lowering </is-goal>
</goal>
```

For such a source block, we could presumably have several XKBT rules. It could be for example *aggregation* of all elementary goals with similar ‘is–goal’ definition, *death–end* for unimportant goals, or *decomposition* into one obligatory target block `<goal>` and several optional knowledge blocks for detailed specification of ‘plain–text’ parts of the source block. For this example, we chose the last one:

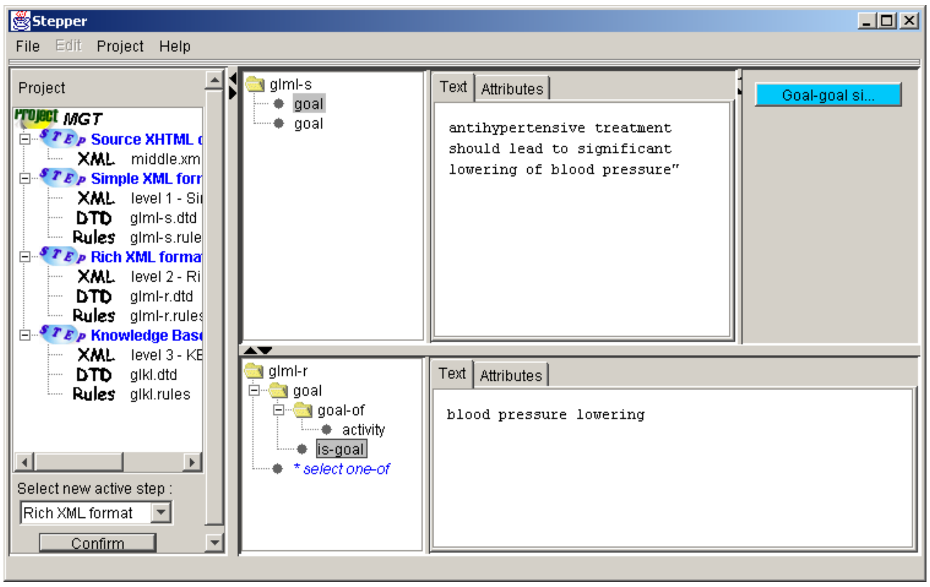
```
XKBT rule definition:
<decomposition name="goal-to-goal">
  <source element="goal" conditions=”all”>
    <cond attribute=”overall” operator=”equal” value=”no” />
    <cond attribute=”direct” operator=”equal” value=”no” />
  </source>
  <compound-target type="sequence">
    <compound-target type="optional">
      <target element="activity_def">
        <comment> used only for activities that should have
        detailed definition </comment>
        <applyXSLT externalLocation=”/activity_def.xsl”/>
      </target>
    </compound-target>
    <compound-target type="optional">
      <target element="goal_object_def">
        <applyXSLT externalLocation=”/goal_object_def.xsl”/>
      </target>
    </compound-target>
    <target element=”goal”>
      <applyXSLT externalLocation=”/goal_to_goal.xsl” />
    </target>
  </compound-target>
</decomposition>
```

The *source* part of the rule says that the rule is applicable only on `<goal>` elements, and that the attributes *overall* and *direct* should be set to *no* (see `<cond>` elements). The target part starts with the definition of two optional blocks – *activity_def* and *goal_object_def*. The last target block `<goal>` is obligatory; it is the main successor of source block. Each target block definition includes a link to an external XSL file used to deploy its sub–structure. The following code corresponds to an XSL file for target block *activity-def* (other XSL files would be much similar):

```
File activity-def.xsl:
<?xml version="1.0"?>
<xsl:stylesheet  xmlns:xsl=
     "http://www.w3.org/1999/XSL/Transform" version="1.0">
   <xsl:output method="xml"/>
   <xsl:template match="goal">
      <xsl:element name="activity-def">
         <xsl:attribute name="name">
            <xsl:value-of select="goal-of/activity"/>
         </xsl:attribute>
         <xsl:attribute name="type">
            <xsl:value-of select="goal-of/activity/@type"/>
         </xsl:attribute>
         <xsl:element name="description">
            <xsl:value-of select="goal-of/activity"/>
         </xsl:element>
         <xsl:element name="definition" />
      </xsl:element>
   </xsl:template>
</xsl:stylesheet>
```

Finally, we shall have a look at the result of transformation, i.e. target knowledge block. The successor `<goal>` element was generated (nearly completely) automatically via an embedded XSLT rule. This was however not the case for the `<activity-def>` and `<goal-object-def>` elements, since correct definition of their content requires human interpretation of text fragments. Only empty elements `<definition>` are thus created and have to be filled manually afterwards.

```
Target knowledge blocks:
<activity-def name="antihypertensive treatment"
              type="treatment">
  <description>antihypertensive treatment</description>
  <definition />
  </activity-def>
<goal-object-def name="blood_pressure_lowering">
  <description>blood pressure lowering</description>
  <definition />
  </goal-object-def>
<goal direct="no" overall="no" id="g2">
  <goal-of activity_id="antihypertensive_treatment" />
  <is-goal goal_object_id="blood_pressure_lowering" />
  </goal>
```



```
antihypertensive treatment
should lead to significant
lowering of blood pressure"
```

```
blood pressure lowering
```

Fig 1. Stepper: user interface for knowledge block transformation

4 Tool support – the *Stepper* system

A software tool supporting the above–described step–by–step transformation has been developed (in Java) under the name of *Stepper*. It provides for:

- *Display* and *editing*[5] of XML documents without showing the XML syntax
- Automated *retrieval* of transformation rules applicable on a source knowledge block delimited by the user
- *Execution* of the rule chosen by the user, and interactive (menu–driven) specification of target knowledge block configuration
- Automated generation and update of *XLink* references across formalisation levels, and *retrieval* of corresponding knowledge blocks or text fragments.
- Convenient creation and update of *XKBT transformation rules*
- Execution of XSLT rules and offline export to non–XML format via an integrated *XSLT processor*.

Fig. 1 shows the interface for interactive transformation. The upper part of the screen shows the source level and the lower part the target level of transformation; each consists of an XML tree and a pane for editing the text content and attribute values. Tree structures, attribute–value forms and rule–activation buttons (upper–right) are generated in runtime from the DTDs of the given formalisation levels and from the premises of applicable rules, respectively. The tool does not automatically

[5] PCDATA element content and CDATA attribute values can be edited freely, while changes to element names, attribute names, element parent–child relationships and element–attribute relationships are bound to execution of transformation rules.

assure completeness of a transformation step but enables to *detect incompleteness* through visualisation of the union of all selections (*Sel** from Definition 6).

In addition, *Stepper* includes a mark–up editor for *free–text* documents[6], similar to semantic annotation tools [3]. 'Transformation' rules for initial text mark–up (populating a given DTD with text–containing elements) are understood as a special form of aggregation rules.

5 Conclusions

The paper presents a technology of XML transformation suitable for documents with rich knowledge content, which has to be interpreted by human user during the transformation process. The user thus supplies the necessary semantics while automated processes accomplish purely syntactical transformations. For this sake, simple interactive XKBT rules are combined with richer but non–interactive XSLT rules. The whole approach is being tested in the medical domain.

References

1. Anutariya, C., Wuwongse, V., Wattanapailin, V.: An Equivalent–Transformation–Based XML Rule Language, *Proc. Int'l Workshop on Rule Markup Languages for Business Rules in the Semantic Web*, Sardinia, 2002.
2. Clark, J.: *XSL Transformations (XSLT) Version 1.0.* W3C, 1999. Online at `http://www.w3.org/TR/xslt`
3. Handschuh, S., Staab, S. (eds): *Annotation in the Semantic Web*. IOS Press, 2003.
4. Lee, D., Mani, M., Chu, W.W.:Schema Conversions Methods between XML and Relations Models, In: *Knowledge Transformation for the Semantic Web*, IOS Press, Amsterdam, 2003.
5. Růžička, M.: *XML Knowledge Block Transformation (XKBT).* Online at `http://euromise.vse.cz/stepper/xkbt`
6. Růžička, M.: Transformace znalostí mezi modely pro zachycování znalostí v XML, In: Sborník konference Znalosti 2003, Ostrava, s. 309-314.
7. Svátek, V., Růžička, M.: Step–by–Step Mark–Up of Medical Guideline Documents. *International Journal of Medical Informatics*, Vol. 70, No. 2-3, July 2003, 329-335.
8. Valenta, M.: Models of XML Data Transformations. In: Sborník Datakon 2002.
9. Wang, D., Peleg, M., Tu, S. W., Boxwalla, A. A., Greenes, R. A., Patel, V. L., Shortliffe, E. H.: Representation Primitives, Process Models and Patient Data in Computer–Interpretable Clinical Practice Guidelines: A Literature Review of Guideline Representation Models. *International Journal of Medical Informatics*, Vol. 68, No. 1-3, December 2002, 59-70.

[6] More precisely, of XHTML documents, which are displayed in the same way as in web browser but their (format–level) XML structure is addressed by XLink references within the corresponding knowledge blocks.