Published in: Gonzàles Rodríguez, Manuel/Suarez Araujo, Carmen Paz (Eds.):
Proceedings of LREC 2002. Third International Conference on Language Resources and Evaluation, 29 May - 31 May 2002. -
Las Palmas: Universidad de Las Palmas de Gran Canaria, 2002. Pp. 432-436.

Converting a Corpus into a Hypertext: An Approach Using XML Topic Maps and XSLT

Eva Anna Lenz, Angelika Storrer

Universität Dortmund, Institut für deutsche Sprache und Literatur
Emil-Figge-Str. 50, D-44227 Dortmund, Germany
lenz, storrer}@hytex.info

Abstract

In the context of the HyTex project, our goal is to convert a corpus into a hypertext, basing conversion strategies on annotations which explicitly mark up the text-grammatical structures and relations between text segments. Domain-specifi knowledge is represented in the form of a knowledge net, using topic maps. We use XML as an interchange format. In this paper, we focus on a declarative rule language designed to express conversion strategies in terms of text-grammatical structures and hypertext results. The strategies can be formulated in a concise formal syntax which is independend of the markup, and which can be transformed automatically into executable program code.

1. Project Framework

We illustrate a model for converting a text corpus into a hypertext. The approach is being developed in the context of the HyTex project (Hypertext conversion based on textgrammatical annotation, http://www.hytex.info/) funded by the Deutsche Forschungsgemeinschaft (DFG), which investigates methods and strategies for text-to-hypertext conversion. The central idea of the project is to base conversion strategies on annotations which explicitly mark up the text-grammatical structures and relations between text segments, e.g. co-reference relations, semantics of connectives, text-deictic expressions, and expressions indicating topic handling (Storrer, to appear). The project will develop a methodology which (semi-)automatically constructs hypertext layers and views, using the text-grammatical annotations.

By storing the hypertext as additional document layers, this methodology preserves structure and content of the original text documents, thus ensuring their recoverability. The multiple-layer approach facilitates the publishing of the same content in different media (cross-media-publishing), and, in addition, provides the reader with the choice beween sequential and selective reading modes. Selective hypertext readers will be supported in findin coherent pathways through the document network.

Feasability and performance of the methodology will be tested and evaluated using a German text corpus. The corpus documents will deal with the same global subject domain, namely "text technology". The corpus comprises various genres (including papers, project reports, textbooks, FAQ, technical specifications glossaries/lexicons, and newsgroup postings). These genres differ with respect to parameters which are crucial for hypertext conversion: length, topic handling, linearity. The corpus will be transformed into a hyperbase by a stepwise approach, the steps being:

- corpus collection and automated annotation of the hierarchical document structure,
- (semi-)automatic text-grammatical annotation, e.g. segmentation of paragraphs into smaller functional units,
- annotation of co-reference relations between text units, in particular expressions used in topic-handling and expressions indicating the rhetorical function of smaller segments,
- annotation of definition for technical terms of the subject domain and of term instances,
- modelling terminological knowledge of the global subject domain, using topic maps (Pepper and Moore, 2001),
- implementation and evaluation of procedures for segmentation, reorganisation, and linking. These operate on the text-grammatical annotations and the topic map model.

In this paper, we concentrate on the technical realisation of the last step. In particular, we focus on a declarative rule language for describing the transformation process. In section 2, we outline this approach. We then give a brief overview of the model's components in section 3 before we focus on the design of the rule language and its implementation in section 4. Conclusions and an outlook are presented in section 5.

2. Simplifying the Transformation Process

The transformation process can be viewed as a mapping from the corpus to the hypertext. It can be described by rules consisting of two parts: a condition (on the documents) and an action that is to be executed when the condition holds. In particular, on the basis of various coherence relations modelled by annotations (conditions), a linking structure of the hypertext is to be generated (actions).

Traditionally, transformations are implemented using programming languages where these rules remain implicit. In the project context – as in other contexts – many rules are similarly structured, leading to either redundant program code or complex programming expressions. Furthermore, a programming language includes many constructs not needed in our context.

The goal of the approach presented here is to design a declarative rule language optimised for expressing conditions on text-grammatical structures and hypertext-generating actions. This approach makes the rules more

compact, easier to understand, and easier to maintain than program code. We will further discuss the advantages and drawbacks of the approach in section 5.

3. Model Overview

Our architecture contains three components (see figur 1):

- the corpus of annotated documents,
- the topic map modelling the technical terms of the subject domain in terms of WordNet relations, and
- a set of rules which declaratively describe how the corpus and the topic map are to be transformed into a hypertext.

All components are expressed in or transformed into XML syntax. This allows us to apply available tools (parsers, generators, browsers, etc.). For example, the syntax of every genre in the corpus, the topic map, and the rule language can be define and checked by document grammars (e.g., Document Type Definition (DTDs)). We can easily reuse documents available in XML or HTML, and produce documents for the web.

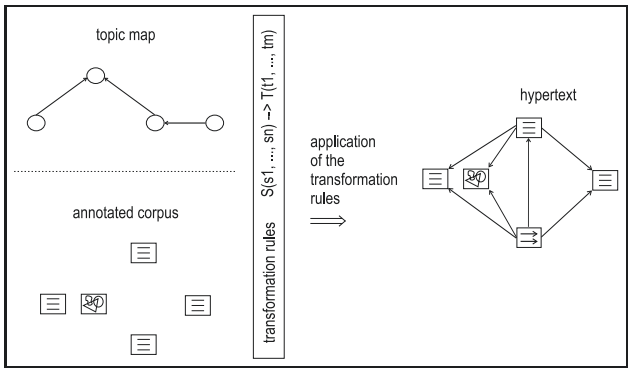

Figure 1: Model overview.

3.1. The Corpus

The German text corpus dealing with the subject domain "text technology" originally contains texts in various formats, including Word and HTML. In a firs step, these formats are converted to XML in order to have a consistent format.

The corpus is then annotated on different levels in subsequent steps in order to prepare it for hypertext conversion. We annotate the hierarchical document structure for each genre, the text-grammatical structure, coreference relations between text units, and definition and instances of technical terms. We gain these annotations semi-automatically, applying several steps of markup generation and filtering where intermediate levels of annotations (e.g., part-of-speech-tagging, lemmatisation, chunking) finall disappear.

Annotations of technical term definition and instances will serve us as examples throughout this paper. These enable us to implement a *linking strategy based on knowledge prerequisites*. This linking strategy is suitable in an application scenario where a reader is a semi-expert, not recognising or being acquainted with every technical term of the domain, for example in the contexts of interdisciplinary research, studies and education, journalism, or lexicography.

Application of the linking strategy provides the reader with a hypertext that supports a selective and problem-oriented reading mode. For example, when a reader comes across a term (the term instance), we can indicate that it is a technical term in the domain, and to provide him with the possibility to learn more about this term, e.g. by following a link to the term definition

We differentiate between three kinds of knowledge prerequisites:

intratextual knowledge prerequisite: the author of a text uses a term for which he has given a definitio in the same text,

extratextual knowledge prerequisite: the author indicates that he uses a term whose definitio is given in another corpus document, or

domain-specifi knowledge prerequisite: the author uses a term well known in the community of the subject domain without indicating that it is a technical term.

We semi-automatically annotate term definition as well as term instances together with the type of knowledge prerequisite. For example, we assume that the knowledge prerequisite is domain-specifi when no term definitio can be found prior to its use.

3.2. The Topic Map

In order to implement the linking strategy based on knowledge prerequisites, we need to model terminological knowledge of the subject domain. This knowledge comprises

- The terms and their relations to other terms, e.g. superordinates and subordinates, synonyms, etc. We organise this knowledge according to the principles of the lexical database WordNet (Fellbaum, 1998), i.e. in the form of a lexical network, and add domain-specifi relations.
- The relationships of terms to documents, e.g. to standard definitions or to occurrences of a term in specifi genres, e.g. in an FAQ ("Frequently asked questions").
- Attributions of terms to usages and communities. For example, we can express that one term is employed by the web community rather than the hypertext community.

Topic Maps (ISO, 2000) provide a standardised syntax for such networks. The main unit of organisation is called a "topic". Topics can be typed, related to other topics, and related to documents in a user-definabl and fl xible way. We model attributions of terms to usages using the topic map construct of scopes. A topic map can be expressed in XML syntax (Pepper and Moore, 2001); topic maps thus meet all our requirements.

Parts of the topic map are eventually transformed into an extended glossary, to be consulted by a hypertext reader when he needs domain-specifi background knowledge. In

addition to definitions the extended glossary for a term contains related terms as well as links to other relevant text passages.

The topic map syntax can be validated using the topic map DTD. In the future, topic map software may also be able to check the semantics of a topic map – for example, the allowed types for relations between topics – define in the form of a topic map schema (Rath, 2000).

4. Describing the Transformation in Terms of a Rule Language

Before explaining the rule language, we give an example of rules implementing the linking strategy based on knowledge prerequisites. These rules make use of the annotation of term definition and instances described above.

Two rules based on this linking strategy can be expressed as follows:

intratextual linking rule: For each technical term used intratextually (i.e. for each instance of a term define in the same text), generate a link from the term instance to its definition

domain-specifi linking rule: For each technical term used domain-specificall (i.e. for each instance of a technical term known in the community and not define by the author), generate an icon indicating a link to the glossary entry for this term.

4.1. The rule language

It should be possible to formulate hypertextualisation strategies – and transformation rules – independently of the markup actually used in the source and target documents. To this end, we introduce the notion of *conceptual constructs* (or *constructs*). Each time when a new construct is introduced, the correspondance between the construct and the markup has to be define (see section 4.2.). Once defined rules can operate on the constructs.

We distinguish between two forms of constructs:

Source constructs are constructs of the source documents, e.g. constructs for expressing coherence relations, term definition and instances, or WordNet synsets.

Target constructs are constructs of the target documents to be generated by the rules. Target constructs in the domain of hypertext comprise, for example, source and destination anchors for hyperlinks, as well as link types or stretchtext.

Each construct has a unique name and can have a number of arguments. The construct depends on the arguments; they form the construct.

In order to address a construct, we use the following syntax:

```
c (a1, a2, ..., an)
```

where `c` is the name of the construct, and a_1 to a_n are the arguments.

As a simplifie example from our domain, the construct

```
termDefinition (term, defText)
```

is a source construct which denotes a definitio of a technical term. It is named `termDefinition` and characterised by the technical term define (`term`) and the text passage containing the definitio (`defText`). Accordingly, the construct

```
termOccurrence
  (termRef, usageType, termInstance)
```

is a source construct denoting a term instance of a technical term. Here, the arguments are the technical term referenced (`termRef`), the way it is used in relation to its definitio (`usageType`: intratextually, extratextually, or domain-specifically and the actual word form used in the text (`termInstance`). As an example for a target construct, consider

```
sourceAnchor
  (destinationAddress, anchorContent)
```

which denotes a source anchor for a link (in HTML: `<a href="...">...</a>`), and

```
destinationAnchor
  (anchorName, anchorContent)
```

which denotes a destination anchor for a link (in HTML: `<a name="...">...</a>`).

The definitio of a construct comprises the following:

- the declaration of the construct's *name*,
- the declaration of its *arguments*, if any,
- and a fragment of program code relating the construct to its markup.

A rule consists of a source construct, serving as a condition on the source documents, an arrow, and a target construct, denoting an action that is to be performed when the condition holds. The arguments of the source construct can be bound to variables (indicated by a $ sign), whose values can be used in the target construct. It can also be expressed that an argument has to have a given value (indicated by single quotes) in order for the source construct to match. Consider the following two rules as examples:

```
termDefinition
  (term=$termVar, defText=$defTextVar)
⟹
destinationAnchor
  (anchorName=$termVar,
  AnchorContent=$defTextVar)
```

and

```
termOccurrence
  (termRef=$termRefVar,
  usageType='intratextual',
  termInstance=$termInstanceVar) ⟹
sourceAnchor
  (destinationAddress=$termRefVar,
  anchorContent=$termInstanceVar)
```

Together, these two rules create a link from a term instance

to its definition thus describing the above intratextual linking rule.

Target constructs can be nested. We use this property to express the domain-specifi linking rule described above:

```
termOccurrence
  (termRef=$termRefVar,
  usageType='domainspecific',
  termInstance=$termInstanceVar)
⟹
sourceAnchor
  (destinationAddress=
    concat($termRefVar, '.html'),
  anchorContent=glossaryIcon())
```

where `concat(`a`,`b`)` is a target construct concatenating a and b, and `glossaryIcon()` is a target construct without arguments that generates the icon. (We assume that the glossary entry already exists).

In order to keep the examples for target constructs simple, we have used examples close to HTML. Of course, the target constructs can also be formulated in terms of an abstract hypertext model like the Dexter Hypertext Reference Model, or the hypertext model by Tochtermann (1995). We also defin hypertext target constructs for the user interface, such as stretchtext or window expansions, the latter being a self-define concept.

Of course, the source constructs can operate on both levels of our model, i.e. on the corpus documents and the topic map. For example, a construct for WordNet word forms has as arguments for the word's synonyms, antonyms, and other related words.

4.2. Implementation

XSLT (Clark, 1999) is a functional programming language optimised for parsing and generating XML documents. An XSLT program takes one or more XML file as its input, and transforms them into one or more file in character string based formats (e.g. HTML or XML).

The following three properties make XSLT the best candidate for our requirements as a programming language:

- All the components of our model are in XML format, or – in the case of the rules – can easily be converted to XML.
- XSLT is a functional programming language supporting a rule-based, declarative programming style. In XSLT, the programmer specifie what output should be produced when particular patterns occur in the input, as distinct from a procedural program where the programmer has to specify what tasks to perform in what order (Kay, 2001, 39). This makes it relatively easy to transform our rules into XSLT.
- An XSLT program is written in XML syntax. This makes it easy to generate XSLT programs with XSLT.

The actual transformation process is twofold. An XSLT program is often called a stylesheet. An initial XSLT stylesheet – the *rule transforming stylesheet* – takes the rules and conceptual constructs as input and translates them into a second XSLT stylesheet, i.e. it translates the rules into an executable form. This second stylesheet – the *rule applying stylesheet* – then generates the hypertext from the corpus documents and the topic map.

For those who are familiar with XSLT, in the remainder of this section, we roughly outline how the rule applying stylesheet is generated automatically.

The basic idea of our algorithm is to generate a matching template for every source construct and a named template for every target construct. We stated in section 4.1. that with the definitio of a conceptual construct a fragment of program code has to be provided that relates the concept to its markup. In the case of a source construct, this is a fragment of XSLT code permitting to match the construct in the XML markup and to bind the arguments accordingly. This is achieved by a list of variable assignments, where each variable corresponds to one of the source constructs' arguments. Each variable is assigned a value which is usually taken from the XML tree, using XPath. In the case of a target construct, the code generates an output, possibly using the variables' values. In both cases, this code fragment is inserted into the matching template or the named templated, respectively.

Here is an example of the code associated with the source construct `termOccurrence` and the target construct `sourceAnchor`:

```
termOccurrence (termRef, usageType, termInstance)
<xsl:template match="termOccurrence">
  <xsl:variable name="termRef" select="@termRef"/>
  <xsl:variable name="usageType" select="@usageType"/>
  <xsl:variable name="termInstance" select=".">
</xsl:template>

sourceAnchor (destinationAddress, anchorContent)
<xsl:template name="sourceAnchor">
  <xsl:param name="destinationAdress"/>
  <xsl:param name="anchorContent"/>
  <!-- inserts
  <a href="$destinationAddress>$anchorContent</a>
  -->
  <xsl:element name="a">
    <xsl:attribute name="href">
      <xsl:value-of select="$destinationAddress"/>
    </xsl:attribute>
    <xsl:value-of select="$anchorContent"/>
  </xsl:element>
</xsl:template>
```

A rule S(s_1, …, s_n) $\Longrightarrow$ T(t_1, …, t_m) inserts a call of the named template for T into the matching template for the source construct S, transferring the variables (and their values) as parameters to the named template.

Basically, the rule language presented above provides a simplifie syntax for one of four possible programming styles supported by XSLT (Kay, 2001, 599 ff.), namely the rule based programming style.

5. Conclusion and Outlook

We have presented a model for the transformation of a corpus into a hypertext, using declarative rules and topic maps. In the context of the HyTex project, our main goal is to test different strategies for text-to-hypertext-conversion. The availability of a declarative way for expressing this conversion greatly facilitates the task.

Although we could use XSLT as a program language which already supports a declarative style of programming, our approach has major advantages over programming directly in XSLT.

We particularly benefi from the separation between conceptual constructs and the markup actually used in cases where the markup is complex, and especially when the information about a construct is scattered about an XML file as it is the case with topics in topic maps (due to the fact that a network structure cannot be presented hierarchically). In this case, a programmer can once defin the desired construct (e.g., a construct for WordNet word forms, with synonyms, antonyms etc. as arguments), writing an inevidently complex code fragment. An expert in hypertext, although not familiar with XSLT, can then apply the construct in a rule describing the transformation of a WordNet word form into a glossary entry.

The separation between constructs and markup has the further advantage that changes in the structure of the markup will only necessitate a change in the program code associated with the definitio of the corresponding construct, i.e. it will require a revision at only one particular and well-define place. Rules operating on the construct do not have to be changed at all.

We also consider it an advantage that at the time of definin a construct, one has to reflec on its arguments. We believe that thinking in an abstract way independently of the markup can enable us to fin out where information in the markup is missing, or where the markup should be reorganised.

However, a serious restriction of the present approach is the fact that target constructs can effectuate an output only at one place in the target document. We intend to enhance the rule language in such a way that an output at two or more places in the target document can be produced. This will enable us to defin target constructs such as "link" that can generate both a source and a destination anchor at different places in the target document. Likewise, the possibility to combine different source constructs in one rule is very desirable.

An interesting extension of the rule language would allow to defin constructs with different markup in different contexts. For example, the source construct `heading` could be marked up differently in different kinds of input documents, or it might be desirable to choose between the output formats HTML and XLink for target constructs generating hyperlinks.

Although performance is not crucial in our context, we also plan to optimise the implementation, using the "key" construct of XSLT which implements a hash mechanism.

It is also necessary to check the semantics of rules, construct definitions and other parts of the model automatically. For example, it should be ensured that rules contain only constructs that have been defined We intend to use Schematron (Cagle et al., 2001, cp.14), a schema language designed to generate this kind of messages for XML documents.

The model will also be extended to include the generation of adaptive hypertexts (Brusilovky, 2001), i.e. hypertexts adapted to a particular user. For example, the different usages of terms modelled in the topic map could be used to generate different hypertext versions for users from different domains or subdomains. Brusilovsky describes several adaptive hypertext techniques (e.g. stretchtext, sorting, visual annotation, and "dimming" of links). Conceptual target constructs for these techniques would greatly facilitate the task of adaptive hypertext generation.

6. References

Peter Brusilovky. 2001. Adaptive hypermedia. *User Modeling and User-Adapted Interaction*, 11(1/2):87–110.

Kurt Cagle, Jon Duckett, Oliver Griffin Stephen Mohr, Francis Norton, Nik Ozu, Ian Stokes-Rees, Jeni Tennison, and Kevin Williams. 2001. *Professional XML Schemas*. Wrox Press, Birmingham.

James Clark, editor. 1999. *XSL Transformations (XSLT) Version 1.0. W3C Recommendation*. W3C. http://www.w3.org/TR/xslt/.

Christiane Fellbaum, editor. 1998. *WORDNET: an electronic lexical database*. MIT Press, Cambridge, Massachusetts.

ISO. 2000. ISO/IEC 13250:2000 Document description and processing languages – Topic Maps. http://www.y12.doe.gov/sgml/sc34/document/0129.pdf, Geneva.

Michael Kay. 2001. *XSLT Programmer's Reference. 2nd edition*. Wrox Press.

Steve Pepper and Graham Moore, editors. 2001. *XML Topic Maps (XTM) 1.0. TopicMaps.Org Specificatio* . TopicMaps.Org. http://www.topicmaps.org/xtm/1.0/.

Hans Holger Rath. 2000. Topic maps self-control. *Markup Languages: Theory & Practice*, 2(4):367–388.

Angelika Storrer. to appear. Coherence in text and hypertext. *Document Design. Journal of research and problem solving in organizational communication*.

Klaus Tochtermann. 1995. *Ein Modell für Hypermedia – Beschreibung und integrierte Formalisierung wesentlicher Hypermediakonzepte*. Ph.D. thesis, Aachen. Shaker.