

Element classes in contextually specified document structures

Felix Sasaki
Computational Linguistics and Text-Technology
University of Bielefeld
Faculty of Linguistics and Literature
33501 Bielefeld
felix.sasaki@uni-bielefeld.de
<http://coli.lili.uni-bielefeld.de/~felix/>

Introduction

A common problem in the design of document grammars is to define a content model which is not too specific, but also not too general. Within the framework of DTDs, the user is able to specify the content model of an element like this:

```
<!ELEMENT word (prefix*, stem+, suffix*)>
```

This declaration is general enough to annotate word structure in languages with a agglutination system. But it is not possible to divide words in several classes, for example a class with words owning one obligatory prefix or a class with words for which the prefix is optional. If one is able to define the classes before the process of document creation starts, the problem might be solved with declarations like the following:

```
<!ELEMENT word1 (prefix, stem+, suffix*)>
```

```
<!ELEMENT word2 (prefix?, stem+, suffix*)>
```

Three problems remain. First, the user might want to use just one element for words, not a single declaration for each class of word structures. Second, the classes might be not defined by the content model, but by the context in which they appear. An example would be an adjective as an predicate ('Das Bier ist kalt') vs. adjectives as noun modifiers ('das kalte Bier'), which show different inflectional patterns. Third, the classification of elements might emerge during the annotation process or while analyzing the document. For software or the user, this is too late to change the document grammar.

Approach of this paper

To solve these problems, a conceptual and physical separation has to be made. The document grammar, written within the DTD format or using other schema languages, contains the most general declaration of a content model like the one above. Another document contains the declaration of element classes, defining alternative structural restrictions for the same elements, i.e. putting restrictions on their context in the document structure. The structure of such a document can be as follows:

```
<classlist mode="default">
  <class scopus="word">
    <caterpillar>
      <moveTest direction="First">prefix</moveTest>
    </caterpillar>
  </class>
  <class scopus="word">
    <caterpillar>
      <moveTest direction="First"></moveTest>
    </caterpillar>
    <caterpillar>
      <moveTest direction="first">prefix</moveTest>
    </caterpillar>
  </class>
</classlist>
```

This document we may call 'class specification document' (CSD). It is declaring caterpillar expressions, a context specification technique (cf. Brüggemann-Klein et al. 2000) which is working with basic moves (up,left,right,first) and basic

tests (isFirst, isLast, isLeaf, isRoot). The caterpillars are organized in classes. Two classes are necessary to divide the content models above: the first class holds for the word1-content model, the second class for the word2-content model. The "scopus" element defines to which element the classes are applied. The meaning of the "mode" attribute will be explained below.

Areas of usage: Validation, generation and exploration of classes

A CSD might be used in two ways. First, as a supplement to a valid document - in the sense that the document follows the requirements made by a document grammar -, the CSD can be used to test if the document is valid to some element classes, too. Different to document grammars written in schema languages like DTD or XML Schema (cf. Thompson et al. 01), this validation process is working upon partial document structures, validating only element classes declared in the CSD. Also, the classes contain several declarations for the same element, so that a general type of the element can be defined and several subtypes, which inherit the characteristics of the super-type. For example, it is possible to define a general pattern for words, e.g. a stem followed by an optional suffix, and several subtypes, e.g. a stem followed by certain suffixes in certain orders. Second, a CSD can also be used to generate classes. This process will be as follows: The user defines some classes in a CSD and sets the attribute "mode" to the value "class generation". The result will be a subset of classes that is in accordance to the classes defined. This generation of classes can be regarded as the exploration of an unknown domain. As in the examples above, the morphological structure of unknown languages is often explored using hypothesis about patterns common to many languages. A very general document grammar representing such patterns could be the starting point of research. With annotated corpora of the language, various hypothesis about the structure might be formulated as a CSD and tested.

Related work

The approach of this paper is related to two fields of research, certain annotation schemes and schema languages. In the first field, as a part of the Text Encoding Initiative (Sperberg-McQueen et al. 1994), the proposal for the declaration of feature structures is a useful approach for the declaration of general data structures. The constructions allowed by a CSD are - in one way - a subset of a feature structure. As a difference to feature structures, a CSD allows for moves. Another set of annotation schemes, specialized for dialogue annotation, was developed during the MATE project. A special focus was put upon the annotation of relations between certain levels of annotation, for example morphological units and dialogue acts. The same is true for a CSD. The difference between these two approaches is that in the MATE project, a single document contains only one level of annotation, connected to other layers with pointing mechanisms like XLink (DeRose et al. 2000).

In the second field, schema languages like Schematron seem to be very similar to CSD. Schematron can be used in the same way, as an supplement for validating documents with a method going beyond common schema languages. The difference is that Schematron is making use of XPATH-expressions to validate document structures. The advantage of XPATH (Clark et al. 1999) is the ability to navigate to any place in the document structure. This seems to be a disadvantage as well: with Schematron, it is not possible to write grammars about documents in a formal sense, because the XPATH Syntax is too unrestrictive. The CSD is restricted to the caterpillars as the only set of expressions. This makes it possible to describe the execution of most of the caterpillar expressions as deterministic, finite-state automata.

References

Brüggemann-Klein, A., and D. Wood. 2000. Caterpillars: A context specification technique. Markup Languages 1(2):81-106.
Clark, J., and S. deRose. 1999. XML Path Language (XPath) version 1.0. W3C Recommendation 16 November 1999.

DeRose, S., E. Maler, and D. Orchard. 2001. XML Linking Language (XLink) version 1.0. W3C Recommendation 27 June 2001.

MATE - Multilevel Annotation, Tools Engineering. Telematics Project LE4-8370. See <http://mate.nis.sdu.dk>.

Schematron - An XML structure validation language using patterns in trees. See <http://www.ascc.net/xml/schematron/>.

Sperberg-McQueen, C. M., and L. Burnard. 1994. Guidelines for electronic text encoding and interchange (TEIP3). Oxford: Oxford University computing services.

Thompson, H.S., D. Beech, M. Maloney, and N. Mendelsohn. 2001. XML Schema Part 1: Structures. W3C Recommendation 2 May 2001.