Multiple Hierarchies: New Aspects of an Old Solution[1*]

Andreas Witt
Universität Bielefeld

In this paper, we present the Multiple Annotation approach, which solves two problems: the problem of annotating overlapping structures, and the problem that occurs when documents should be annotated according to different, possibly heterogeneous tag sets. This approach has many advantages: it is based on XML, the modeling of alternative annotations is possible, each level can be viewed separately, and new levels can be added at any time. The files can be regarded as an interrelated unit, with the text serving as the implicit link. Two representations of the information contained in the multiple files (one in Prolog and one in XML) are described. These representations serve as a base for several applications.

1 Introduction

Markup expresses characteristics or interpretation of text. It is obvious that there is, at least potentially, more than one view for a given text. Often it is necessary to express these different or alternative views of text explicitly, i.e. by markup. At the moment, it seems to be a tendency to annotate more and more information. This development definitely takes place in the field of linguistics, where language data is associated with information from several linguistic levels of description, e.g. semantics, syntax, morphology, phonology – levels which

---

[1] This paper is a slightly modified reprint. (Originally published in the Online-Proceedings of the Extreme Markup Languages 2004, see http://www.extrememarkup.com).

[*] The different aspects of this approach are used within several projects of 'Research Group: Text-technological Modeling of Information' which is funded by the German Research Foundation (DFG). I would like to thank Harald Lüngen and Neill Kipp for their help and all the reviewers of this paper for their helpful comments.

Interdisciplinary Studies on Information Structure 02 (2005): 55–85
Dipper, S., M. Götze and M. Stede (eds.):
Heterogeneity in Focus: Creating and Using Linguistic Databases

are (relatively) independent of each other. But also text simply published on the web is combined with more and more meta-information. Since markup expresses meta-information about text, the amount of markup will increase, especially if the semantic web will emerge. And, of course, more markup implies that it becomes more likely to encounter multiple hierarchies.

This paper deals with two different problems:

1. the problem of annotating overlapping structures, and
2. the problem that occurs when documents should be annotated according to different, possibly heterogeneous tag sets.

As a solution of both problems the technique of annotating documents in multiple forms is proposed and described in detail. The paper also discusses the disadvantages of the approach, disadvantages that are definitely the reason why a lot of projects reject this solution: “An obvious and also simple solution would be to make a separate file for each transcription. However, this makes comparison between levels unnecessarily cumbersome, and it is notoriously difficult to keep track of revisions in parallel files.” (Haugen, 2004)

This paper shows how it is possible and what is needed to overcome these problems.

2 Multi-hierarchically Structured Text

Publishing, especially print publishing, was the driving force behind the development of markup languages. Text was viewed as an *ordered hierarchy of content objects* (OHCO). Consequently most markup languages are based on the OHCO assumption. The term and the acronym were introduced by DeRose et al. (1990) and were further discussed by Renear et al. (1996).

2.1 Problems of OHCO-based Markup-Languages and Possible Solutions

From a formal point of view, SGML-based markup systems allow for the representation of exactly one hierarchy. Hence, in principle, only one structure can be represented in one document. In practice, this restriction often does not receive special attention as different structures often can be expressed within one hierarchy. Thus, e.g., the logical structure of a text, i.e. the division into captions, lists, sections etc., differs completely from the syntactic structure such as the division of the text into sentences and phrases. Especially, none of the elements belonging to the different tag sets overlap. Hence, it is possible to project both structures into one hierarchy without problems. The disadvantage is, however, that this necessarily results in a mixture of these structures, in the annotated text as well as in the corresponding document grammar.

The problem of multiple hierarchies is often discussed. The main reason for this might be the view of document engineers, who are faced with the fact that ranges of text marked up by SGML or XML elements must not overlap. Single-hierarchically structured text is a consequence of this restriction. If overlapping does not occur, the problem of combining heterogeneous tag sets is often ignored. Hence, a mixture of structures can be found quite often in text represented in one syntactic hierarchy. One example was already given, another example is HTML. Even in its ‘strict’ version, different structures can be mixed, at least through the often promoted use of the elements `span` and `div` combined with an assignment of a `class` information.

To avoid confusion when talking about multiply structured text and text ideally organized by multiple hierarchies, the terms ‘level’ or ‘level of description’ are used when referring to a logical unit, e.g. visual document structure or logical text structure. When referring to a structure organizing the text technically in a hierarchically ordered way, the terms ‘layer’ or ‘tier’ are

used. A level can be expressed by means of one or more layers and a layer may/can include markup information on one or more levels (cf. Bayerl et al., 1999).

2.1.1 SGML/XML Approaches

The problem of representing multiple hierarchies has often been addressed and several solutions have been proposed, especially in the field of humanities computing, which is by nature concerned with text and its interpretation or its description. Consequently, the best collection of techniques is presented by the *Text Encoding Initiative* (TEI, see ACH/ACL/ALLC (1994) and Barnard et al. (1995)). The TEI describes the techniques for using SGML for annotating multiple hierarchies. (1) `CONCUR`: an optional feature of SGML (not available in XML) which allows multiple hierarchies to be marked up concurrently in the same document, (2) milestone elements: empty elements which mark the boundaries between elements, in a non-nesting structure, (3) fragmentation of an item: the division of what logically is a single element into two or more parts, each of which nests properly within its context, (4) virtual joins: the recreation of a virtual element from fragments of text, (5) redundant encoding of information in multiple forms.

With the exception of the extremely rarely implemented option `CONCUR`, in effect, all of these techniques are workarounds:

- Milestones do not allow for making use of a key concept of XML, namely elements containing a range of text. This leads to several consequences:
    - No content model restriction can be stated by a document grammar for the range of text between the milestones marking the begin and the end of the region. This results in not being able to use an XML editor for annotating these regions.

    - Standard SGML parsers cannot check whether milestone elements marking the begin and the end of a region match.
    - It is more difficult or impossible to process these regions by means of a style sheet, e.g. by XSLT or, respectively, by CSS.
- The technique of fragmentation results in ‘containers’ containing only a part of the text. So for instance, an element `sentence` or `para` that is fragmented simply does not contain a sentence or a paragraph.
- The technique of virtual joins requires a separate interpretation of the SGML document.
- Redundant encoding in multiple forms results in multiple files which are not integrated in a larger unit containing all the information of the different layers.

Another technique not mentioned directly by the TEI guidelines is stand-off annotation, i.e. (new) layers of annotation are added by building a new tree whose nodes are SGML elements which do not contain textual content (`#PCDATA` in terms of the DTD syntax), but links to another layer.

In some respects stand-off annotation is a generalization of virtual joins, because not only contents of elements are joined, but also ranges between points within the document. Sometimes these ranges make use of markup already contained in a layer, sometimes special pointers are used to refer to the specific text elements which are the object of the annotation (Pianta and Bentivogli, 2004). With the first introduction of this concept (Thompson and McKelvie, 1997) this second approach was described.

In practice, however, most often an already-annotated layer is taken as the primary annotation tier, to which the stand-off annotation is linked. In the case of linguistic annotation often the annotation level ‘word’ is used as the primary annotation layer. In most of its applications, stand-off annotation makes use of

one layer as the link target of the new tier, but it is also possible to link to several already existing layers (see Carletta et al., 2003).

In any case, stand-off annotation results in new hierarchies established by new annotation layers that are linked to already existing annotations. Sometimes the new layer is included in the same document, sometimes the layers are separated.

This approach has the advantage that it is based on SGML/XML and that different levels of description are kept separate. However, this approach has some drawbacks too:

- The new layers require a separate interpretation.
- The layers, although separate, depend on each other. They can only be interpreted by reference to the layer(s) they point to.
- Although all information is included, the information is difficult to access using generic methods. As a consequence, standard parsing or editing software cannot be employed.
- Standard document grammars (e.g. the TEI Relax NG scheme, the XHTML-DTD, or the W3C Schema for DocBook) can only be used for levels containing both markup and textual data.
- Linking to a sub-element range, or to textual data not annotated at all is difficult. The pointing mechanism defined by the TEI or by XPointer can be used, but requires another special software solution.
- The primary layer should be a (primary) level. The choice of such a primary level is not an easy task. Often its declaration is arbitrary and artificial.

Despite these disadvantages the technique of stand-off annotation is used in a lot of projects faced with the problem of multiple hierarchies, especially in the area of annotating linguistic data.

2.1.2 Namespaces

The Namespace standard provides a mechanism to specify where a specific element has been defined (Bray et al. 1999). Connecting elements with their defining document grammars is done by adding a prefix to the element or the attribute names. The prefix points, at least conceptually, to a document grammar, in which the element or the attribute is defined. Thus the logical structure of a text can be marked up with e.g. XHTML elements for captions, sections, lists etc. and its syntactic structure can be marked up by using an adequate module of the DTD of the TEI. If a corresponding namespace has been defined, a caption belonging to the logical structure of the text can be referenced by `html:h2` instead of only `h2`, whereas a word or a morph can be marked up by `tei:w` or `tei:m` instead of `w` or `m`. This enrichment of the annotation facilitates the recognition of the relation between the annotation and a specific level (here text structure and morphology).

Unfortunately, some problems remain. Sometimes a document grammar defines several different structures, possibly in a modular way. The document grammars defined by the TEI-DTD are a good example of this. As an ad-hoc solution, one could try to define different namespaces for the same document grammar. A first prefix `teins1` and a second prefix `teins2` could be defined. Because the prefixes have only the function of a place holder for the expanded name spaces, it is necessary to declare several different 'real' namespaces for one DTD. But this would definitely be against the intention of the standard.

Nonetheless namespaces are an important help when using markup that belongs to different levels of description, since it provides a means to refer to an element not only by its name or its generic identifier but additionally by its defining document grammar.

A minor problem of namespaces might occur when using schema languages which allow for context-sensitive definitions of content models. With this technique it is possible to define a different content model for regions marked up with elements with the same element name. For example, Relax NG and XML Schema allow for such definitions. The (slightly) different definitions of an element `para` in sections and `para` in the context footnote, where (embedded) footnotes should be prohibited, is an often used example of the use of this option. But since the namespace points to the document grammar and not to the element definition, context-sensitively defined elements cannot be distinguished.

One problem has not been addressed by the namespace recommendation at all: the problem of overlapping hierarchies.

2.1.3 Non SGML-based Markup languages

Some non-SGML-based markup languages have been proposed in the last few years. An example of such a markup language is the Multi-Element Code System (MECS, Sperberg-McQueen and Huitfeld 1999) or TexMECS (Huitfeldt and Sperberg-McQueen, 2001). Its major extension with respect to SGML and XML is that overlapping ranges are admitted within documents.

In 2002 another markup language was proposed, called *Layered Markup and Annotation Language* (LMNL, Tennison and Piez (2002)). LMNL is a markup language which not only allows for annotating overlapping elements but also for connecting the element names to corresponding annotation levels. All structures modeled by XML can also be modeled by LMNL.

2.1.4 Discussion

The problem of annotating multiple hierarchies can be divided into two different and relatively independent problems: (1) SGML-based markup systems cannot handle ‘overlapping hierarchies’ and (2) the tag sets used or needed for a certain

annotation task are sometimes quite heterogeneous. The first problem is addressed by the solutions proposed in the TEI guidelines, by stand-off annotation, and by the TexMECS markup language, which does not conform to SGML. The second problem is addressed by the namespace recommendation.

LMNL provides a solution for both problems: regions marked up by different elements may overlap and its layered annotation approach is specially designed for this task. But, since LMNL does not conform to SGML, not to mention XML, it has not been applied up to now.[2]

Another possibility mentioned above is redundant encoding in multiple forms. This approach is rarely used by the markup community. The reasons for this seem to be clear: First, most people try to avoid redundancy. Second, and more important, multiple encodings in different forms are independent of each other, but people who deal with annotated text are only interested in an integrated format.

On the other hand, it is also an advantage if one annotated document is not related to another document, because then the document is an independent unit of information. This leads to several more advantages.

- If a document is used for separate annotation levels, this results in each level being able to be viewed separately and new levels to be added at any time, without reference to and dependence on existing files.
- Standardized document grammars can be used for some annotation levels and specialized document grammars can be defined in an intuitive way, i.e. declaring that an element can contain text and not only attributes whose values point to some other element in some other annotation layer.

---

[2] One exception is described by Alexander Czmiel (2004). He implemented a subset of LMNL in an XML-conformant way. Of course, some of the advantages of LMNL cannot be achieved by such an XML-based representation.

Moreover, the approach (as well as stand-off) has additional advantages over the milestones and the fragmentation approach:

- The modeling of alternative annotations based on different theoretical assumptions is possible (see Sasaki et al. (2003) for the usefulness of this point in the field of linguistics).
- Each document instance uses its own DTD (or Schema), i.e. document grammars are not mixed up.

We therefore conclude that this approach has a lot of advantages with respect to the aspects of editing, maintenance, interchange, and reusability of XML-annotated data. What remains to be solved is the main drawback of independent annotations: How is it possible to connect these layers?

We also conclude that a special representation model for these data is needed, because of the redundancy in the data. This representation format is desired for storing and processing this information. From a theoretical point of view, LMNL would be an ideal format. From a practical viewpoint a stand-off annotation approach is most suited for these tasks and, in fact, is used most frequently.

2.2 Multiple Annotations and their Representation

Beside the advantages of the annotation in multiple form, the main problem of this approach has been addressed: the independence of the tiers. But interrelations of annotation layers are of interest for many persons concerned with structuring and modeling of information. In this section a method is presented which complements the advantages of *redundant encoding of information in multiple forms* with possibilities to link these multiple forms and represent them uniformly. Furthermore, conversion tools for the annotation format and possible representation formats are described.

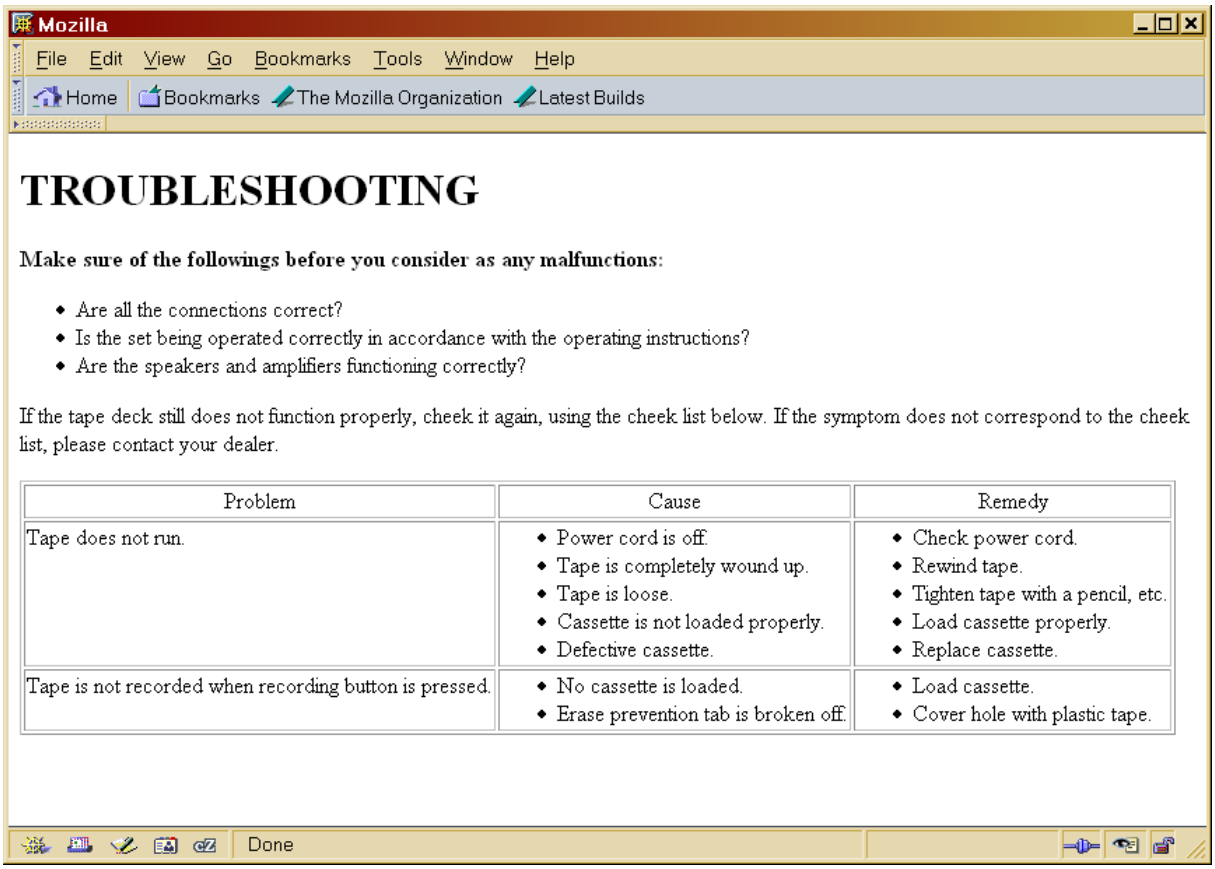

Fig. 1: Screenshot of the rendering of the HTML-version of the example-text

2.2.1 XML-based Multi-layer Annotation

One obvious way to interrelate different annotations of same textual data exists. The different annotations could be regarded as transformations of each other. Hence, the relations between the XML documents can be declared in an XSLT-program or an XSLT-stylesheet. This stylesheet can be viewed as a description of relations between two XML vocabularies. But for composing such a stylesheet it is necessary to have information on the relation of the elements defined in the different vocabularies. Moreover, this approach could only be successful, if the relations between the elements can be stated unambiguously.

Another way to link the *different forms* was proposed by Witt (2002). The central idea of this approach is that the annotated text itself serves as the link. This is achieved by annotating exactly the same text several times.

This approach is described by means of a simple example. Below the XHTML-source of a user's manual is given (see also Fig. 1)

```
<xhtml><h1>TROUBLESHOOTING</h1>
...
<table border="1">
 <tr>
  <td align="center">Problem</td>
  <td align="center">Cause</td>
  <td align="center">Remedy</td>
 </tr>
 <tr>
  <td valign="top">Tape does not run.</td>
  <td valign="top"><ul>
   <li>Power cord is off.</li>
   <li>Tape is completely wound up.</li>
   <li>Tape is loose.</li>
   <li>Cassette is not loaded properly.</li>
   <li>Defective cassette.</li>
 </ul></td>
 <td valign="top"><ul>
   <li>Check power cord.</li>
   <li>Rewind tape.</li>
   <li>Tighten tape with a pencil, etc.</li>
   <li>Load cassette properly.</li>
   <li>Replace cassette.</li></ul></td>
</tr>
<tr>
 <td valign="top">Tape is not recorded when recording button
is pressed.</td>
 <td valign="top"><ul>
   <li>No cassette is loaded.</li>
   <li>Erase prevention tab is broken off.</li>
 </ul></td>
 <td valign="top"><ul>
   <li>Load cassette.</li>
   <li>Cover hole with plastic tape.</li></ul>
 </td>
</tr>
</table></xhtml>
```

The same fragment of text can be annotated in a more content-oriented way or semantically:

```
<r><h1>TROUBLESHOOTING</h1>
...
<p-c-r>
 <description>
  <first>Problem</first>
  <second>Cause</second>
  <third>Remedy</third>
 </description>
 <case>
  <problem>Tape does not run.</problem>
  <potential_causes>
   <cause>Power cord is off.</cause>
   <cause>Tape is completely wound up.</cause>
   <cause>Tape is loose.</cause>
   <cause>Cassette is not loaded properly.</cause>
   <cause>Defective cassette.</cause>
 </potential_causes>
 <potential_remedies>
   <remedy>Check power cord.</remedy>
   <remedy>Rewind tape.</remedy>
   <remedy>Tighten tape with a pencil, etc.</remedy>
   <remedy>Load cassette properly.</remedy>
   <remedy>Replace cassette.</remedy></potential_remedies>
</case>
<case><problem>Tape is not recorded when recording button is
pressed.</problem>
 <potential_causes>
   <cause>No cassette is loaded.</cause>
   <cause>Erase prevention tab is broken off.</cause>
 </potential_causes>
 <potential_remedies>
   <remedy>Load cassette.</remedy>
   <remedy>Cover hole with plastic tape.</remedy>
 </potential_remedies>
</case>
</p-c-r></r>
```

As can be seen, the text content of both versions is identical, but the markup is different.

2.2.2 Representation

The multiply annotated XML documents are the basis of the representations. For further processing of the text it is necessary to represent them uniformly. Two alternative representations are described in the next subsections.

PROLOG

Sperberg-McQueen et al. (2001) discuss the *meaning and interpretation of markup*. For explaining their approach, annotated documents are represented in the programming language Prolog. In their representation, every element, attribute, and the content are saved as so-called Prolog facts. This approach has been extended, so that multiple annotations as described in the previous section can be represented. Through this all separate annotations can be associated in a data basis, which then can be used e.g. for automatic detection of relations between the annotation levels (see section 3.2).

In the simplest setting, for any element, attribute and text node of each annotation level a Prolog fact is built which contains the following information:

1. a cross reference to the annotation level;
2. the absolute start position of the text passage which is marked up;
3. the end position of that text passage;
4. the position of the unit in the tree representation of the annotation level;
5. the element name or — if necessary — the attribute name, respectively

Some Prolog facts containing information from the two levels of the above examples should serve as an illustration.

```
node('tape-xhtml.xml', 729, 786, [1,5,3,2], element('td')).
node('tape-xhtml.xml', 729, 786, [1,5,3,2,1], element('ul')).
node('tape-xhtml.xml', 729, 751, [1,5,3,2,…], element('li')).
node('tape-thema.xml', 729, 786, [1,5,3,2], element('pot…')).
node('tape-thema.xml', 729, 751, [1,5,3,…], element('cause')).
```

The first argument contains the name of a layer, i.e. `tape-xhtml.xml` and `tape-thema.xml`. The second element points to the beginning of a range annotated with the respective element (the fifth argument). In the example, all the ranges start at the same position. The end of each range is given as the third

argument. The position in the tree (argument four[3]) is given as a list, pointing to the nodes within the tree representation of the respective annotation layer.

Attributes are represented in a similar way, using the Prolog predicate `attr`:

```
attr('tape-xhtml.xml', 729, 786, [1, 5, 3, 2],
     'valign', 'top').
```

The textual content is given by the predicate `pcdata_node`:

```
pcdata_node(729, 730, 'N').
pcdata_node(730, 731, 'o').
pcdata_node(731, 732, ' ').
pcdata_node(732, 733, 'c').
pcdata_node(733, 734, 'a').
pcdata_node(734, 735, 's').
pcdata_node(735, 736, 's').
```

Such a collection of Prolog facts contains all the information of the different annotations and can serve as a data basis for further developments of Prolog programs.

XML-Based Representation

Multiply annotated XML files can also be represented in an XML-based format. Such a presentation could be achieved by transforming the Prolog facts into XML elements, e.g. the predicate `node` with its five arguments could be transformed to an empty XML element `node` with five attributes. However, such a *Prolog-in-XML* representation would not make much sense.

A representation using the technique of virtual joins, or stand-off annotation, is more interesting, because this technique is used to represent multiple hierarchies. Moreover, most of the above mentioned disadvantages of

[3] In first case this means: The element `td` is the second daughter of the third daughter of the fifth daughter of the root element.

this technique do not exist when this format is an add-on for the multiple annotation of XML layers.

The European language technology project NITE developed a format for representing heavily annotated data. This format is well suited for this task.

The NITE-format (Carletta et al., 2003) combines several files forming a corpus. These files are interrelated with each other. One way to represent the two annotation layers `tape-xhtml.xml` and `tape-thema.xml` is given in the next examples. The NITE-corpus consists of four separate files, in the examples these could be:

- `tape.corpus.xml` contains meta-information, e.g. names of the files of the corpus, names of the defined elements and attributes etc.
- `o1.stream.xml` contains the textual data supplemented with reference points for linking with the other layers
- `o1.tape-xhtml.xml` comprises the markup of `tape-xhtml..xml`
- `o1.tape-thema.xml` expresses the information provided by the markup of the file `tape-thema.xml`

One possible representation of the textual stream would supply any character with an ID:

```
<char nite:id="char_727">e</char>
<char nite:id="char_728">d</char>
<char nite:id="char_729">.</char>
<char nite:id="char_730">N</char>
<char nite:id="char_731">o</char>
<char nite:id="char_732"> </char>
<char nite:id="char_733">c</char>
<char nite:id="char_734">a</char>
<char nite:id="char_735">s</char>
<char nite:id="char_736">s</char>
```

Alternatively, in larger text single words could serve as the reference units.

The next example shows how the elements of the thematic annotation are linked to the text.

```
  <nite:child href="o1.stream.xml#id('char_727')" />
  <nite:child href="o1.stream.xml#id('char_728')" />
  <nite:child href="o1.stream.xml#id('char_729')" />
</problem>
<potential_causes nite:id="potential_causes_2" >
  <cause nite:id="cause_6" >
    <nite:child href="o1.stream.xml#id('char_730')" />
    <nite:child href="o1.stream.xml#id('char_731')" />
    <nite:child href="o1.stream.xml#id('char_732')" />
    <nite:child href="o1.stream.xml#id('char_733')" />
```

The elements `potential_causes` and `cause` begin at the character with the reference `char_730`, i.e. the first character of the string 'No cassette is loaded'. The string itself is given by references to the characters in the file `o1.stream.xml`.

2.2.3 Conversion

The conversion from XML to Prolog is implemented in Python. The program `xml2prolog.py` receives as an input one or more XML documents and outputs a collection of Prolog facts.[4]

The element `<Root>` is represented as the fact:

```
node(AnnotationLayer, 0, n, [1], element(Root)).
```

where `n` refers to the last character in the textual data. The XML attributes of the root element `att1` and `att2` and their values `val1` and `val2` are represented as two facts:

```
attr(AnnotationLayer, 0, n, [1], 'att1', 'val1').
attr(AnnotationLayer, 0, n, [1], 'att2', 'val2').
```

This representation contains some redundant information, because the pointers to the character (0 and n) could be inferred automatically by means of the

[4] This program is mainly written and maintained by Daniel Naber and Oliver Schonefeld. It is available via the project Web pages (http://www.text-technology.de; 'Projekt Sekimo').

information of the respective element, but the explicit indication of this information can speed up processing.

Some options for the transformation process are:

compare: the primary data, i.e. the `PCDATA` content of the elements of the XML files is compared; if the primary data is not identical, the first different character is shown;

pcdata/pcdatanodes: character data is included;

aggressive: whitespace is added or removed anywhere in the document if whitespace is the reason for differences of the primary data;

filter: some elements in some files should be filtered (including their textual content), e.g. `<script>` within HTML-documents.

That way it is possible to convert any number of identical but differently marked up texts into a collection of Prolog facts.

For the conversion of text which is annotated in multiple forms according to the NITE-format, another program has been developed.[5] This program is called `nexus.pl` and is implemented in the Perl programming language. The functionalities are similar to `xml2prolog.py`. The input is *n* annotations of the same text. The program outputs a NITE-corpus that consists of the *n+2* files described above.

2.2.4 Discussion

It has been shown that the technique of annotating the same text in multiple forms has many advantages and that its main drawback can be avoided. However, exactly the same data has to be annotated several times. With this prerequisite the multiply annotated files can be regarded as a unit which is heavily interrelated, because the text serves as the implicit link.

---

[5] This program has been developed by Jan Frederik Maas. Also this program is available via the project Web pages.

After that, two different formats have been described. One format is an interrelated Prolog representation of the information contained in the multiple files. The other format is based on XML and was developed for the processing and the exchange of linguistic corpora annotated on several levels of description.

Furthermore, programs for the automatic transformation of multiply annotated text to the integrated formats have been introduced.

3 Aspects of Processing Multiply Annotated Text

In this section, techniques and software implementations for editing, inferring and unifying separately annotated texts are presented. Moreover, a technique of unifying the multiple forms will be discussed.

3.1 Editing

The editing of copies of text, each annotated separately, definitely is not an easy task. One way to do this is annotating each file with the help of a standard XML editor. Since, at least in some scenarios, the text is given and need not be changed, this approach offers at least two advantages: standard XML-editing software is available and the automatic comparison of the textual content (e.g. by the option ‘compare’ of the transformation program `xml2prolog` described above) allows quality assurance, since it is highly unlikely that exactly the same modification of the textual data occurred twice (or even more times) in different files. Unfortunately, this has also several drawbacks. One of these is connected with the comparison of whitespace. Since sometimes whitespace matters, it makes no sense to collapse all whitespace. On the other hand, most often this difference should be ignored. Therefore a special whitespace normalization

program has been implemented.[6] But if textual data must be changed, textual content must be changed in different files. This task requires special editing software.

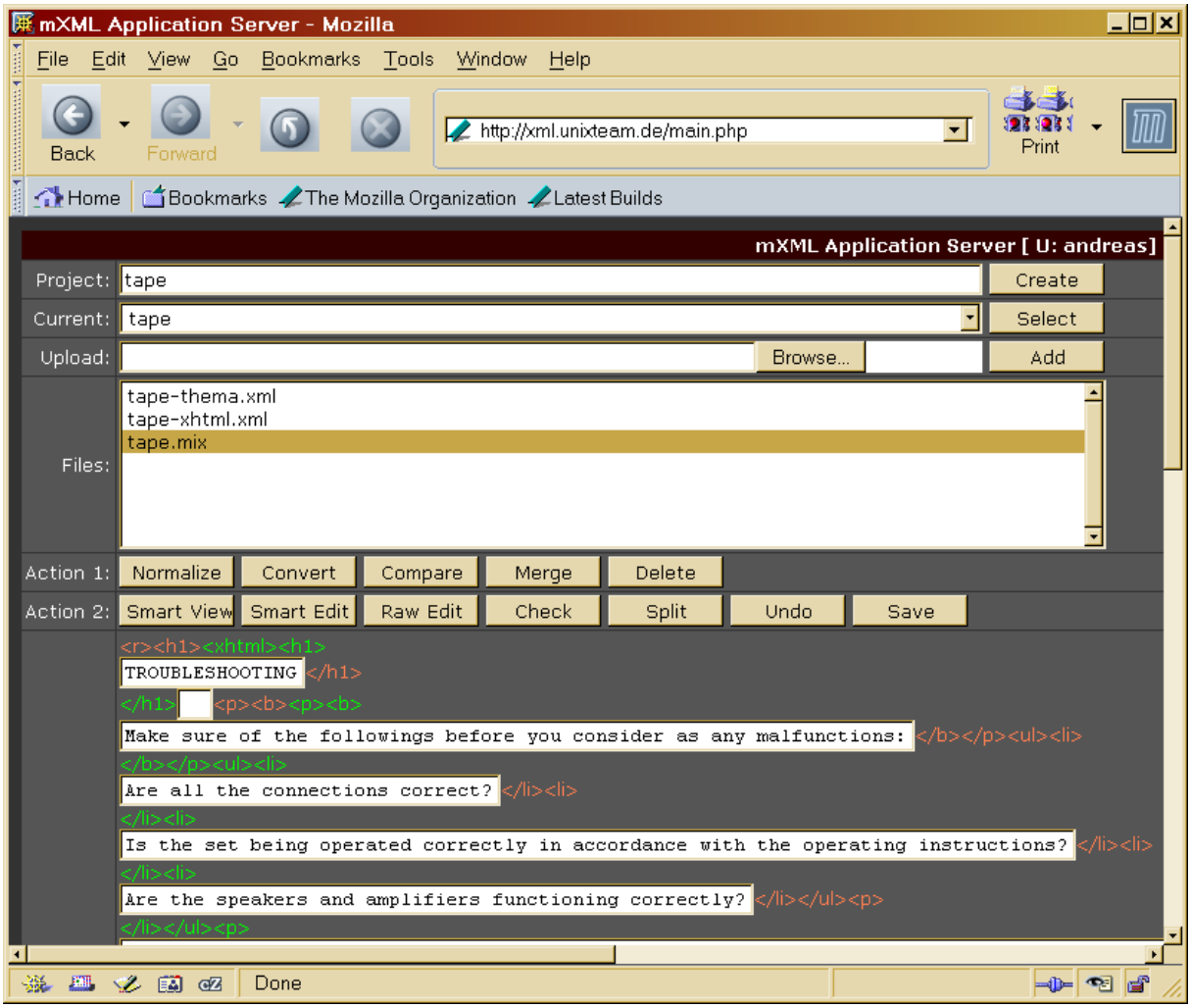

Fig. 2: Editor mode for changing textual content

At the time of writing this paper two master's thesis projects are concerned with implementing special editing software for this task.

One editor is web-based (implemented in PHP) and allows for typing and changing the textual content of multiply annotated files. The two screenshots

---

[6] This program is written and maintained by Oliver Schonefeld. It is available via the project Web pages (http://www.text-technology.de; 'Projekt Sekimo').

give an impression of this program. Fig. 2 shows how text can be modified. As can be seen, the markup cannot be changed in this mode.

Fig. 3 shows the non-XML-based markup employed internally by the editor. This format can be used by experts to modify not only the textual content but also the markup.

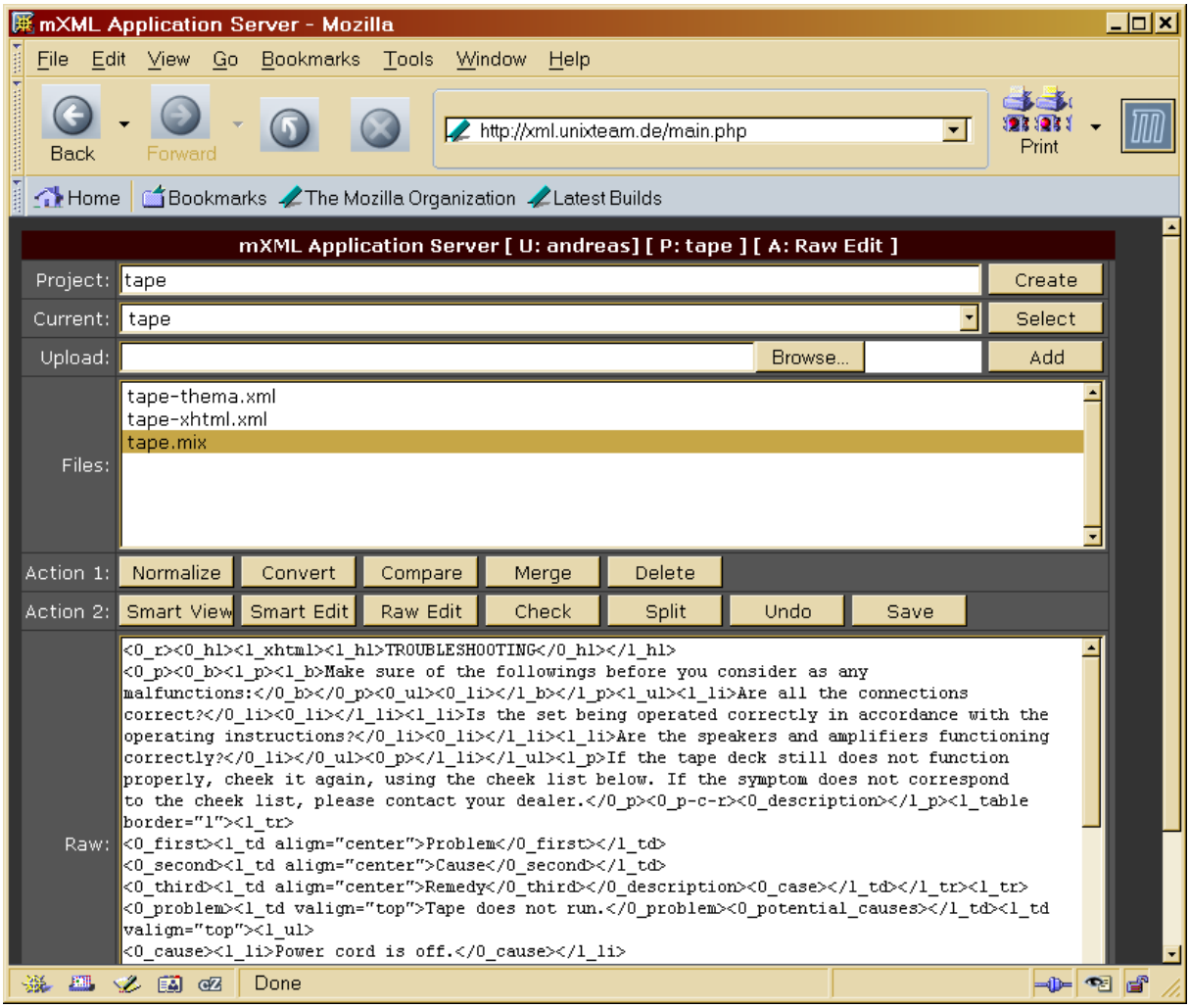

Fig. 3: Editor mode editing textual content and markup

The existence of such an editor is important for this approach. Otherwise, it is very difficult to change multiply annotated text, because each modification of the text must be done in each layer.

In a second master's thesis an editor will be implemented in the Java programming language, using the Eclipse platform. The aim of this master's

project is the implementation of an editor capable of associating several document grammars with one text. The insertion of elements is a two step process: first, the annotator refers to a document grammar the element belongs to and, second, (s)he can choose an element out of a list of elements that are allowed at this point according to the schema. When saving the document, for each associated schema one file will be saved. The validation will take place for each of these files.

3.1.1 Relations Between Annotations

The markup within a single document is hierarchically structured. The structure, leaving aside cross-references, can be represented as a tree. Between the nodes of the tree there exist certain relations, i.e. subordination, (direct) neighborhood, etc. These relations can be used for queries for structural characteristics in one layer. Such queries can be formulated in several ways, as e.g. with XSLT, in query languages like XQuery or (when using the appropriate library) in Prolog (see Sperberg-McQueen et al., 2002).

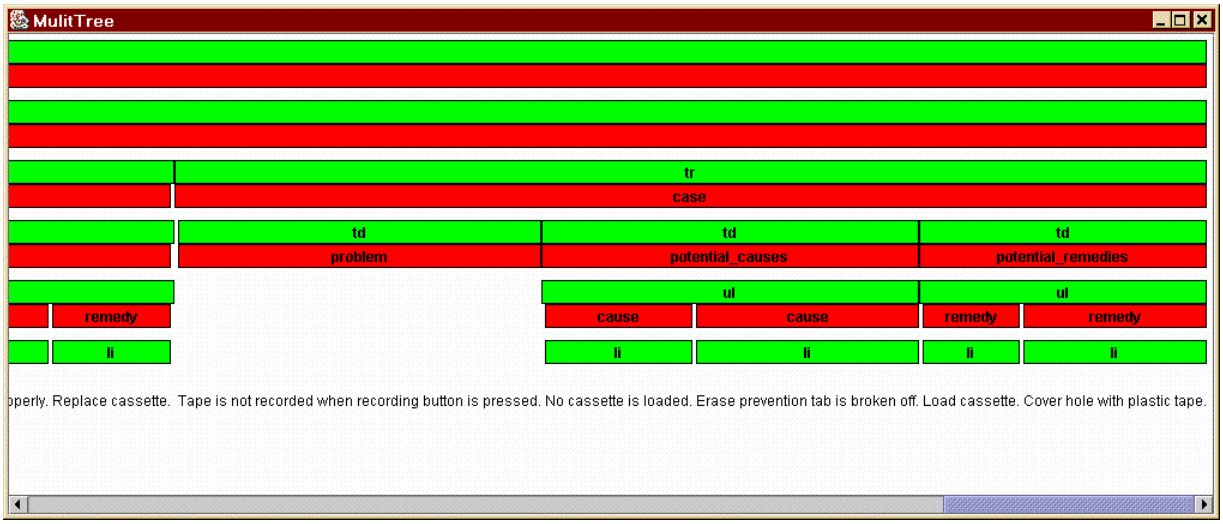

Fig. 4: Example annotation with two layers

When regarding more than one annotated layer more relations can be found. The figure above depicts the two layers of the example annotation, the XHTML

layer and the content-oriented annotation layer. This visualization shows some of these relations.

An aligned representation of both layers shows that an identical range in the primary data is marked up with different elements.

```
...<potential_causes><cause>No cassette is loaded.</cause>...
...<td valign="top"><ul><li>No cassette is loaded.</li>...
```

Durand (1999) and Durusau & O'Donnell (2002) assembled all the possible relations between elements of different layers. The visualization is based on the presentation of Durusau & O'Donnell (2002).

```
Start-tag identity
<a>.................................</a>
<b>............</b>

Full inclusion
<a>.................................</a>
           <b>.........</b>

Total identity
<a>.................................</a>
<b>.................................</b>


End-point identity
            <a>.....................</a>
<b>.................................</b>


Ranges annotated by different elements overlap
<a>.....................</a>
              <b>.............................</b>


The end-position of one element is shared by the start-tag of
another element
<a>.................</a>
                    <b>................</b>

etc.
```

Within our project, the Prolog fact base is used as a base for inferencing these relations. For this task, special Prolog predicates have been implemented.[7]

Alternatively, the NITE XML search tools[8] could be used for representations conforming to the NITE representation.

3.2 Relations Between Annotation Layers

More general information on the relations between element classes, i.e. the set of all instances of an element, is more interesting than a comparison of relations between single element instances. To do this, a set of meta relations have been defined. A meta relation holds under certain conditions.

The meta relation `identity` between the element classes `a` and `b` holds, if for every occurrence of an element instance `a` the same range of text is annotated by an element instance `b` and vice versa.

```
Meta-relation identity:
<a>.....................</a>
<b>.....................</b>
```

The meta relation `inclusion` between the element classes `a` and `b` holds, if for every occurrence of an element instance `a` the same range of text is annotated by an element instance `b`, and if the meta-relation `identity` does not hold, i.e. for all occurrences, one of the following configurations can be found:

```
<a>...................</a>
<b>..............................</b>

                          <a>.....................</a>
<b>.........................................</b>
```

---

[7] This program was mainly written by Daniela Goecke. It is available via the project Web pages.

[8] NXT Search is freely available (binaries, documentation, and source code) via http://www.ims.uni-stuttgart.de/projekte/nite/download.shtml.

```
          <a>.....................</a>
<b>.....................................</b>

<a>.....................</a>
<b>.....................</b>
```

The meta-relation `overlap` between the element classes `a` and `b` holds, if for every occurrence of an element instance `a` the range annotated by `a` overlaps with the range annotated by an element instance `b`. For all occurrences of `a`, the following configuration can be found:

```
<a>.....................</a>
            <b>.....................</b>
```

The inferred meta-relations indicate whether theoretical constructs modeled by (certain elements of) two document grammars are in some relation to each other. So it might be investigated whether certain constructs used by different linguistic theories (e.g. in traditional Japanese grammar and in 'modern' phrase structure grammars) are alphabetical variants of each other. Moreover, with these meta-relations, generalizations stated by researchers or inferred automatically on a small empirical basis can be falsified.

Unfortunately, however, the research conducted by the projects of the DFG research group mentioned above showed that these meta-relations do not hold very often. The reason for this lies in the way they are defined: a meta relation between two elements holds if certain conditions hold for *all* occurrences of these elements. It would be interesting to explore whether certain meta relations exist under certain conditions.

One possibility for a refinement of the meta relations is a description of specific contexts where these relations do hold. Context specifications allow for expressing such a condition.

A context specification could be expressed by a set of XPath expressions, but XPath seems to be a language that is too powerful for context specifications.

Therefore, an alternative format to express structural properties, called "Context Specification Document" (CSD), has been developed (Sasaki and Pönninghaus, 2003).

3.3 Unification of Annotation Layers

Of course, sometimes an integrated XML representation is necessary. Therefore a program for the unification of multiply annotated documents has been developed.[9] With this Prolog program two document layers can be merged. The architecture of this program is visualized in the next figure.

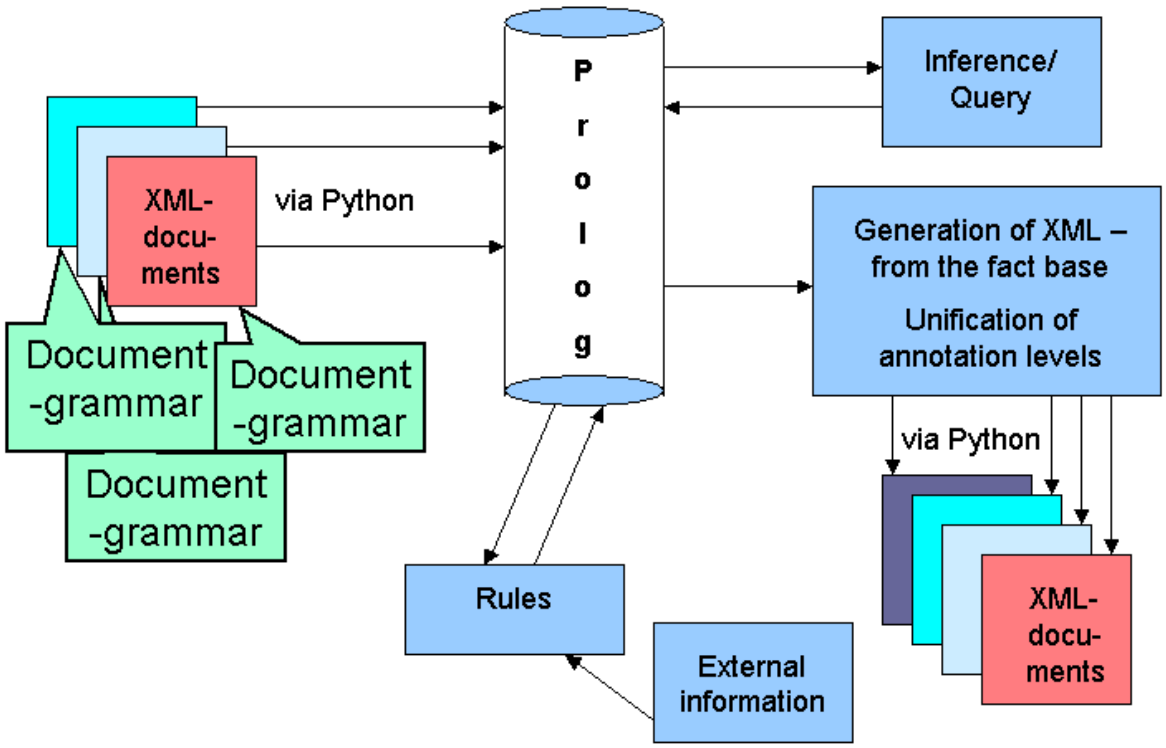

Fig. 5: Unification of annotation layers

[9] This program was mainly written by Daniela Goecke and is maintained by Harald Lüngen. It is called `semt.pl` and it is also available via the project web pages. It is also described by Witt et al. (2004).

The Prolog predicate (`semt`) receives four arguments:

- layer1 (to be unified)
- layer2 (to be unified)
- list of elements which should be deleted in the process of unification

The result of the merger (again a collection of Prolog facts) is written to a new file specified in the fourth argument. The new database contains a copy of all layers in the input database plus the result layer.

In case the unification results in a layer where the elements are not properly nested, a second result layer (a difference list) is created. The resulting database is re-converted to XML, again using a Python program.

If no difference list exists, the result of the merging of two layers can be linearised as an XML document straightforwardly. In case the resulting fact base contains a difference list, two different linearizations can be generated. The default processing uses milestone elements to mark the borders of incompatible elements. Alternatively, the technique of fragmentation of elements can be invoked.

4 Conclusion

In this paper it was argued that the problem of representing and processing multiply structured data should be subdivided into two separate problems. First, it is necessary to declare and/or apply to this data elements and attributes defined by different document grammars or belonging to different tag sets. It is desirable to be able to distinguish these elements according to their origins. Furthermore it can happen that the elements belonging to different tag sets mark overlapping regions, which would result in structures that are difficult to handle with SGML-based markup languages. Several proposed solutions for both problems have been discussed. It was argued that the most simple solution, i.e.

annotation of multiple structures or hierarchies in multiple files, can be a way to overcome both problems and that this approach offers many benefits. However, it is necessary to ensure that the multiple files can be represented as a single unit. For doing this, some preconditions have to be accepted by the users of this approach.

5 References

ACH/ACL/ALLC (1994). *Guidelines for Electronic Text Encoding and Interchange (TEI P3)*. C. M. Sperberg-McQueen and L. Burnard (eds.). Chicago, Oxford: Text Encoding Initiative.

Barnard, David, Lou Burnard, Jean-Pierre Gaspart, Lynne A. Price, C. M. Sperberg-McQueen, and Giovanni Battista Varile (1995). *Hierarchical Encoding of Text: Technical Problems and SGML Solutions.* In: N. Ide and J. Véronis (eds.). The Text Encoding Initiative: Background and Context, Special Issue of Computers and the Humanities, 29(3), pp. 211-231.

Bayerl, Petra Saskia, Harald Lüngen, Daniela Goecke, Andreas Witt, and Daniel Naber (2003). *Methods for the Semantic Analysis of Document Markup.* In: C. Roisin, E. Munson, and C. Vanoirbeek (eds.). Proceedings of the ACM Symposium on Document Engineering (DocEng 2003), pp. 161-170.

Bray, Tim, Dave Hollander, and Andrew Layman (eds.) (1999). *Namespaces in XML*. W3C Recommendation, World Wide Web Consortium.

Carletta, Jean, Jonathan Kilgour, Tim O'Donnell, Stefan Evert, and Holger Voormann (2003). *The NITE Object Model Library for Handling Structured Linguistic Annotation on Multimodal Data Sets.* In:

Proceedings of the EACL Workshop on Language Technology and the Semantic Web (3rd Workshop on NLP and XML, NLPXML-2003).

Czmiel, Alexander (2004). *XML for Overlapping Structures (XfOS) using a non XML Data Model.* ALLC/ACH 2004, Joint Conference of the ALLC and ACH, Göteborg.

DeRose, Steve, David Durand, Elli Mylonas, and Allen Renear (1990). *What is Text, Really?* Journal of Computing in Higher Education, 1(2), pp. 3-26.

Durand, David G. (1999). *Palimpest: Change-Oriented Concurrency Control for the Support of Collaborative Applications*. PhD Thesis, Boston University.

Durusau, Patrick and Matthew Brook O'Donnell (2002). *Concurrent Markup for XML Documents*. XML Europe 2002.

Haugen, Odd Einar (2004). *Parallel Views: Multi-level Encoding of Medieval Nordic Primary Sources.* Literary and Linguistic Computing, 19(1), pp. 73-91.

Huitfeldt, Claus and C. M. Sperberg-McQueen (2001). *TexMECS: An Experimental Markup Meta-Language for Complex Documents*. `http://www.hit.uib.no/claus/mlcd/papers/texmecs.html`.

Pianta, Emanuele and Luisa Bentivogli (2004). *Annotating Discontinuous Structures in XML: the Multiword Case.* In: A. Witt, U. Heid, H. S. Thompson, J. Carletta, and P. Wittenburg (eds.). Proceedings of the LREC-Satellite-Workshop on XML-based Richly Annotated Corpora, Lisbon, pp. 30-37.

Renear, Allen, Elli Mylonas, and David Durand (1996). *Refining Our Notion of What Text Really Is: The Problem of Overlapping Hierarchies.* In: International Association for Literary and Linguistic Computing: Selected papers from the ALLC/ACH Conference: Christ Church, Oxford, April 1992. Oxford: Clarendon Press.

Sasaki, Felix and Jens Pönninghaus (2003). *Testing Structural Properties in Textual Data: Beyond Document Grammars.* Literary and Linguistic Computing, 18(1), pp. 89-100.

Sasaki, Felix, Andreas Witt, and Dieter Metzing (2003). *Declarations of Relations, Differences and Transformations between Theory-specific Treebanks: A New Methodology*. In: J. Nivre (ed.). Proceedings of the Second Workshop on Treebanks and Linguistic Theories, Växjö, pp. 141-152.

SGML ISO 8879:1986. *Information processing – Text and office systems – Standard Generalized Markup Language (SGML).*

Sperberg-McQueen, C. M. and Claus Huitfeldt (1999). *Concurrent Document Hierarchies in MECS and SGML*. Literary and Linguistic Computing, 14(1), pp. 29-42.

Sperberg-McQueen, C. M., Claus Huitfeldt, and Allen Renear (2001). *Meaning and Interpretation of Markup*. Markup Languages: Theory & Practice 2(3), pp. 215-234.

Sperberg-McQueen, C. M., David Dubin, Claus Huitfeldt, and Allan Renear (2002). *Drawing Inferences on the Basis of Markup*. In: Proceedings of Extreme Markup Languages 2002.

Tennison, Jeni and Wendell Piez (2002). *The Layered Markup and Annotation Language*. In: Proceedings of Extreme Markup Languages 2002.

Thompson, Henry S. and David McKelvie. *Hyperlink Semantics for Standoff Markup of Read-Only Documents*. In: Proceedings of SGML Europe ’97.

Witt, Andreas, *Meaning and Interpretation of Concurrent Markup.* In: ALLC/ACH 2002, Joint Conference of the ALLC and ACH, Tübingen.

Witt, Andreas, Harald Lüngen, Felix Sasaki, and Daniela Goecke (2004). *Unification of XML Documents with Concurrent Markup.* In: ALLC/ACH 2004, Joint Conference of the ALLC and ACH. Göteborg.

XQuery (2004) *XQuery 1.0: An XML Query Language*. S. Boag, D. Chamberlin, M. F. Fernàndez, D. Florescu, J. Robie, and J. Siméon (eds.). W3C Working Draft, 23 July 2004.

XSL Transformations (1999). *XSL Transformations (XSLT) Version 1.0*. J. Clark (ed.). W3C Recommendation, 16 November 1999.

Andreas Witt
Universität Bielefeld
Fakultät für Linguistik und Literaturwissenschaft
Arbeitsbereich Computerlinguistik und Texttechnologie
Postfach 10 01 31
33501 Bielefeld
Germany
andreas.witt@uni-bielefeld.de
http://coli.lili.uni-bielefeld.de/~andreas/