Proceedings of Extreme Markup Languages®

Towards validation of concurrent markup

Oliver Schonefeld
Andreas Witt

Abstract

XCONCUR[1] allows for the annotations of multiple concurrent hierarchies, but lacks cross-layer validation. This paper explores the requirements for a constraint-based approach for such a validation process.

Keywords: Modeling; Concurrent Markup/Overlap; Validating

Table of Contents

Introduction
Multi-hierarchical annotation: XCONCUR
Constraints between annotation layers
    Building blocks for expressing constraints
    Building constraint expressions
    Constraint expressions and the XML world
Discussion

Oliver Schonefeld

Oliver Schonefeld studied Computer Science and Language Technology at Bielefeld University. Parts of the paper deal with aspects of his PhD thesis.

Andreas Witt

1996 - 2006, Andreas Witt has taught at Bielefeld University, Germany in the field of 'text technology'. His research interests include the combination of computational linguistics and markup technologies, schema languages, and corpus annotation. Since April 2006 he is engaged in a project on "Sustainability of linguistic data" of the Universities Hamburg, Potsdam/Berlin and Tübingen.

XML Source PDF (for print) Author Package Typeset PDF

Towards validation of concurrent markup

Oliver Schonefeld [Bielefeld University]
Andreas Witt [Bielefeld University]

Extreme Markup Languages 2006® (Montréal, Québec)

Introduction

More and more complex annotations of resources often yield documents, which are annotated on more than one annotation layer. These annotation layers form concurrent hierarchies with or without overlapping elements.

A transcription of a dialog between Peter and Paul (figure 1) could be annotated on two annotation layers: in layer 1 it is coded who produced what utterance and layer 2 captures the sentence structure. Both layers are independent from each other. In our example Peter is interrupted by Paul who completes the sentence, which was begun by Peter. The annotation of the utterance and that of the sentence structure produces an overlap.

Several techniques have been developed to handle overlap in XML. Methods like fragmentation and milestones have been discussed extensively (see **[Barnard et al. (1995)]**, **[Sperberg-McQueen & Burnard (1994)]**, **[Witt (2004)]**, **[DeRose (2004)]**, **[Bauman 2005]**).

Figure 1

Peter: Hey Paul! Would you give me

Paul: the hammer?

An example dialog.

This late breaking paper examines the requirements for a constraint based validation method of multiple concurrent hierarchies. A constrained based approach is used, since finding a generalized method for converting any arbitrary set of annotation schemas into one multi-hierarchical schema may involve enormous complexity problems.

The recently started Rabbit/duck grammers (see **[Sperberg-McQueen (2006)]**) are an approach to develop grammars and therefore a foundation for validation for overlapping hierarchies.

Multi-hierarchical annotation: XCONCUR

At Extreme 2005 MuLaX [Multi-layered XML] **[Hilbert et al.(2005)]** was introduced to bring back SGML's CONCUR OPTION to XML and allow users to intuitively write overlapping markup in a standardized way. After the conference MuLaX was renamed to XCONCUR[2].

The XCONCUR Syntax is similar to SGML with the CONCUR option set to 'YES'. Each element has to be prefixed with an annotation layer id and therefore is assigned to an annotation layer.

Similar to XML, XCONCUR requires a well-formed structure on each annotation layer. To check for well-formedness on a selected annotation layer, all annotations (tags) that do not belong to the selected annotation layer must be deleted. Then all annotation layer id prefixes must be removed from the remaining tags. An analogous method must be applied to any possibly existing DOCTYPE declarations. The resulting document must meed XML's well-formedness critaeria.

An annotation schema, such as DTD, XML Schema or RelaxNG, can optionally be assigned to each annotation layer in a XCONCUR document using DOCTYPE-like declarations. To yield a valid XCONCUR document each annotation layer must be valid, i.e. must meet the layer's document type definition. If no annotation schema is assigned, only a well-formedness check can be done on that layer. The presentation will provide a more detailed description of XCONCUR.

Figure 2 shows a concurrent annotation of the dialog from figure 1 on two annotation layers using XCONCUR. The first annotation layer with the prefix *l1* provides the speaker turns and the second annotation layer with the prefix *l2* contains the linguistic markup. Each layer is associated to a DTD using a DOCTYPE declaration.

Figure 2

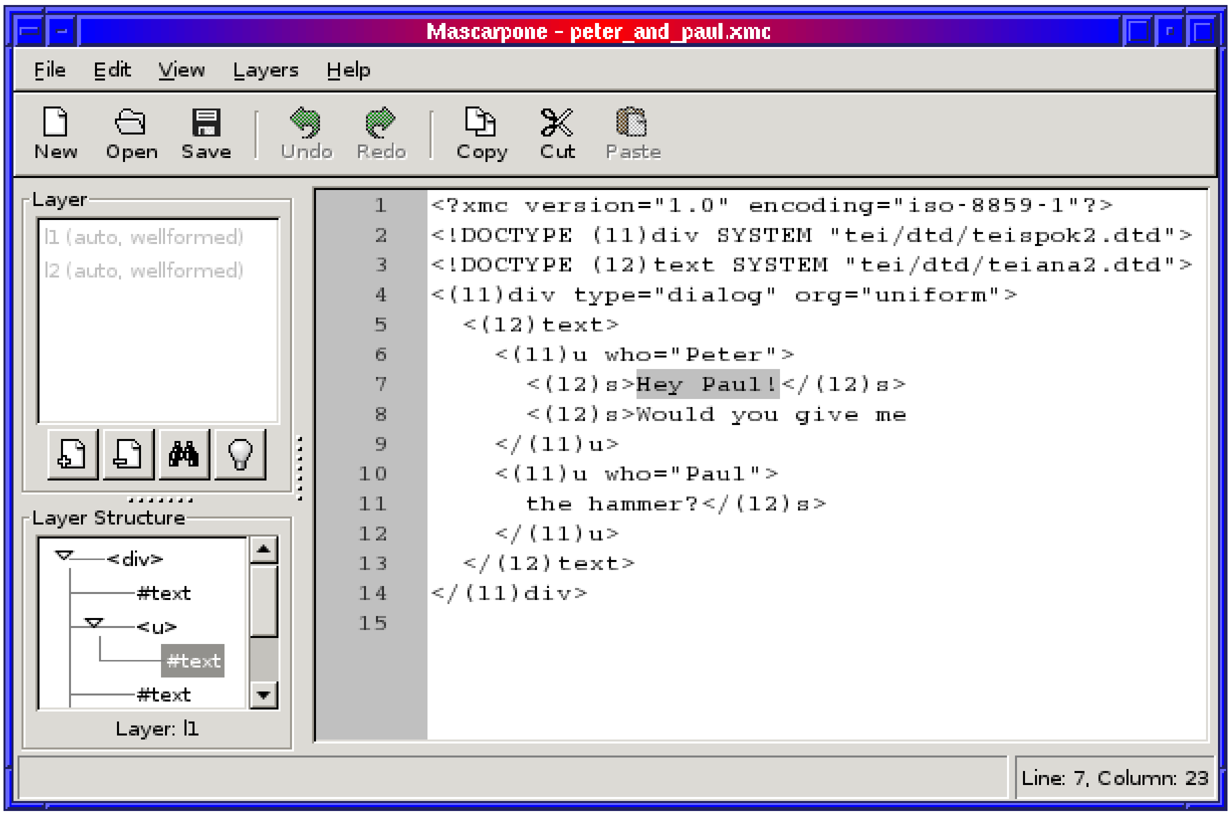
```
<?xconcur version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE (l1)div SYSTEM "tei/dtd/teispok2.dtd">
<!DOCTYPE (l2)text SYSTEM "tei/dtd/teiana2.dtd">
<(l1)div type="dialog" org="uniform">
  <(l2)text>
    <(l1)u who="Peter">
      <(l2)s>Hey Paul!</(l2)s>
      <(l2)s>Would you give me
    </(l1)u>
    <(l1)u who="Paul">
      the hammer?</(l2)s>
    </(l1)u>
  </(l2)text>
</(l1)div>
```

The example dialog annotated with XCONCUR.

Constraints between annotation layers

The current prototype XCONCUR editor (figure 3) allows users to work with XCONCUR documents, but the software lacks validation. There is ongoing work to add validation on a single annotation layer by implementing support for DTDs, XML Schema and RelaxNG. But even after that a further problem remains: How is it possible to formally state possible relations of the annotation layers? The answer of this question is a prerequisite for achieving a cross-layer validation.

Figure 3



```
<?xmc version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE (l1)div SYSTEM "tei/dtd/teispok2.dtd">
<!DOCTYPE (l2)text SYSTEM "tei/dtd/teiana2.dtd">
<(l1)div type="dialog" org="uniform">
  <(l2)text>
    <(l1)u who="Peter">
      <(l2)s>Hey Paul!</(l2)s>
      <(l2)s>Would you give me
    </(l1)u>
    <(l1)u who="Paul">
      the hammer?</(l2)s>
    </(l1)u>
  </(l2)text>
</(l1)div>
```

[Link to open this graphic in a separate page]

A screenshot of the mascarpone XCONCUR editor

The need of a cross layer annotation exists for instance in the field of linguistics. In a linguistic annotation one may want to annotate words, syllables and morphemes. On the one hand syllables and morphemes occur within words and must not overlap words. On the other hand syllables and morphemes may or may not overlap. An author of such an annotation may want to ensure that those assertions hold and therefore may use a set of constraints to restrict the occurrences of syllable (annotated in a phonological annotation layer) and morpheme elements (in an annotation layer 'morphology') to text ranges annotated as word elements in an annotation layer 'lexical items'.

For example the word *Paul* contains just one syllable and one morpheme. In an annotation all elements would span over the same range of characters in the primary data and therefore be equal, since the start of the word is also the start of the syllable and the morpheme. Analogously, the same holds for the end. The word *tables* has two syllables (*ta* and *bles*) and two morphemes (the stem *table* and the plural morpheme *s*). Again, both syllables and morphemes are contained in the word. But the morphemes and syllables result in an overlapping structure.

A set of constraints crafted for this example has to allow the user to express these relations, by allowing the user to formulate rules like: *"A syllable must be contained in a word or a syllable must be equal to a word."* or *"A morpheme must be contained in a word or it must be equal to a word."*. Thus a validation process must be able to handle such constraints.

If there is no overlap between two elements, two elements can relate to each other in the following three ways:

- One element may contain another element. This can be the case if the element is completely included in another element and their start and end elements are not shared in any way, or if the two elements either share their start or end tag.
- One element equals the other element, that is both elements share their start and end tag.
- Both elements have nothing in common, i.e. an element on an annotation layer precedes or succeeds an other element on another annotation layer.

Regarding the word/morpheme example from above it is clear that if we focus on a single morpheme, it is contained in a word. But since the text usually contains more than one word, this morpheme is outside the borders of all but one word. A validation process must take this into account. The validation process can only apply constraints to a single instance of an affected element. If and only if this combination will allow the constraint to evaluate to true, all other instances of elements must not be regarded. I.e. if the morpheme *table* is found inside the word *tables* the constraint will be satisfied and all other instances of word elements will not be considered[3].

In the following we want to examine how to define constraint expressions which allow to express relations between elements on different annotation layers. First, we want to define an abstract syntax to have a notation for constraint expressions. Second, we examine, how to use the constraint expressions to define all different relations which may occur between two elements. And finally, we describe how to bring the constraint expressions closer to the XML world.

Building blocks for expressing constraints

The general scheme for building constraint rules is a expression like "***operand operator operand***". One rule must contain at least one of those expressions, but may contain more. The operands take a *layer-id* and *element-name* as parameters.

In the following the *layer-id* refers to an annotation layer id prefix, which is given in the DOCTYPE declarations of a XCONCUR document. In the XCONCUR example document (figure 2) the sentence layer has the annotation layer id *l1* and the sentence layer *l2*. Therefore the strings `l1` or `l2` can be used for *layer-id* in the constraint rule expressions. The *element-name* is the name of an element, which belongs to the annotation layer referenced by the *layer-id*. In repect to the XCONCUR document example one could choose `div` or `u` on annotation layer *l1* or `text` or `s` on annotation layer *l2*. Obviously all names of elements, which are defined in the annotation layer's document schema can be used for *element-name* [4].

The following entities can be used as operands and operators:

Operands

start(*layer-id*, *element-name*)	Denotes the start tag of an element with the name *element-name* on annotation layer *layer-id*.
end(*layer-id*, *element-name*)	Denotes the end tag of an element with the name *element-name* on annotation layer *layer-id*.
element(*layer-id*, *element-name*)	Denotes an element with the name *element-name* on annotation layer *layer-id*.

Operators

op1 << *op2*	"precedes operator": entity *op1* precedes entity *op2*. *op1* and *op2* may only be a **start** or **end** operand.
op1 == *op2*	"equals operator": The entity defined by *op1* and the entity defined by *op2* appear at the same position[5] in the corresponding annotation layers. *op1* and *op2* may only be a **start** or **end** operand.
op1][*op2*	"outside operator": entity *op1* is not enclosed by *op2*, that is entity op1 is not enclosed between the start and end tag of the denoted element. *op1* may be a **start** or **end** operand

and *op2* must be an **element** operand.

Figure 4

```
start(l1, div) << start(l2, s)
```

A simple constraint expression rule.

Using the example given in figure 4, we will illustrate the syntax of the constraint expressions. The Rule may be applied to the XCONCUR document in figure 2. It consists of just one expression and both operands are connected using the precedes operator. The left hand side operand is the start tag of an element `div` on annotation layer *l1*; the right hand side operand is the start tag of an element `s` on annotation layer *l2*.

Using natural language we can paraphrase the rule in figure 4 as "*the start tag for an element div on annotation layer l1 precedes the start tag for an elements on annotation layer l2*". If this rule is applied to the example in figure 2, it will evaluate to 'true', since all occurrences of the start tag of element `div` on annotation layer *l1* precede the start tags of element `s` on annotation layer *l2*.

For more compact and "human-friendly" rules, we could introduce three more operators:

op1 >> *op2*	"succeeds operator": the operator is similar to the precedes operator, but entity *op1* succeeds entity *op2*.
op1 [] *op2*	"inside operator": the operator is similar to the outside operator, but entity *op1* must be enclosed by the start and end tag of element denoted by *op2*.
op1 <= *op2*	"precedes-or-equals operator": this operator allows to combine the precedes or equals operator in one rule for more compact rules. The entities *op1* and *op2* may only be a **start** or **end** operand.

The first two operators can be defined in terms of the already existing, if we use the inverse of the corresponding operators and therefore they can be seen as "syntactic sugar"[6]. For the third one, we must allow the use of alternatives in the constraint rules.

Building constraint expressions

To be able to describe a set of constraints between two annotation layers, we first need know which kind of relations may occur between two elements in different annotation layers. **[Durand (1999)]** and **[Durusau & O'Donnell (2002)]** compiled a list of these possibilities. The visualization is oriented on **[Durusau & O'Donnell (2002)]**.

```
1. No-overlap

 l1:                      <a>...........</a>
 l2: <b>..........</b>

2. Elements share one end/start point

 l1:                  <a>.............</a>
 l2: <b>.............</b>

3. 'Classic' overlap

 l1:              <a>...................</a>
 l2: <b>....................</b>

4. Elements share end point
```

```
 l1:             <a>...................</a>
 l2: <b>...............................</b>
```

```
5. One element contained within the other
```

```
 l1:         <a>..............</a>
 l2: <b>...............................</b>
```

```
6. Elements share start point
```

```
 l1: <a>...............................</a>
 l2: <b>.............</b>
```

```
7. Elements share both start and end points
```

```
 l1: <a>...............................</a>
 l2: <b>...............................</b>
```

```
8. Elements share start point
```

```
 l1: <a>..............</a>
 l2: <b>..............................</b>
```

```
9. One element contained within the other
```

```
 l1: <a>..............................</a>
 l2:         <b>..............</b>
```

```
10. Elements share end point
```

```
 l1: <a>..............................</a>
 l2:                  <b>.............</b>
```

```
11. 'Classic' overlap
```

```
 l1: <a>...................</a>
 l2:       <b>........................</b>
```

```
12.  Elements share one end/start point
```

```
 l1: <a>.............</a>
 l2:                   <b>............</b>
```

```
13. No overlap
```

```
 l1: <a>...........</a>
 l2:                     <b>..........</b>
```

The different relations can be modeled with the following constraints. All expressions are connected by an implicit **and** connector, so all expressions must evaluate to 'true' for the constraint to hold.

1. *"element b precedes element a"*

   ```
   end(l2, b) << start(l1, a)
   end(l2, b) ][ element(l1, a)
   ```

2. *"element a shares start tag with end tag of element b"*

   ```
   end(l2, b) == start(l1, a)
   ```

3. *"elements a and b overlap and start tag of element b precedes start tag of element a and end tag of element b precedes end tag of element a"*

   ```
   start(l2, b) << start(l1, a)
   start(l1, a) << end(l2, b)
     end(l2, b) << end(l1, a)
   ```

4. *"element a and b share end tags, but start tag of element b precedes start tag of element a"*

```
start(l2, b) << start(l1, a)
  end(l2, b) == end(l1, a)
```

5. *"element a is contained inside element b"*

```
start(l2, b) << start(l1, a)
  end(l1, a) << end(l2, b)
```

6. *"element a and element b share start tags and end tag of element b precedes end tag of element a"*

```
start(l2, b) == start(l1, a)
  end(l2, b) << end(l1, a)
```

7. *"elements a and b share start and end tags"*

```
start(l2, b) == start(l1, a)
  end(l2, b) == end(l1, a)
```

8. *"element a and element b share start tags and end tag of element a precedes end tag of element b"*

```
start(l2, b) == start(l1, a)
  end(l1, a) << end(l2, b)
```

9. *"element b is contained inside element a"*

```
start(l1, a) << start(l2, b)
  end(l2, b) << end(l1, a)
```

10. *"element a and b share end tags, but the start tag of element a precedes the start tag of element b"*

```
start(l1, a) << start(l2, b)
  end(l2, b) == end(l1, a)
```

11. *"elements a and b overlap and start tag of element b is contained inside element a, but end tag of element b is not contained inside element a"*

```
start(l1, a) << start(l2, b)
start(l2, b) << end(l1, a)
  end(l1, a) << end(l2, b)
```

12. *"element a shares end tag with start tag of element b"*

```
  end(l1, a) == start(l2, b)
```

13. *"element a precedes element b"*

```
  end(l1, a) << start(l2, b)
start(l2, b) ][ element(l1, a)
```

Constraint expressions and the XML world

The DTD in figure 5 can be used to define a set of constraint expression rules in XML. The constraint expression rules are modeled with a focus on elements. A relation between two elements can be expressed using the `<element>` element. The `relation` attribute sets the relation type between the two elements. It's value corresponds the different operators defined in section "Building blocks for expressing constraints" and may be `precedes`, `equals`, `outside`, `succeeds`, `inside` or `precedes-or-equals`. The left hand side of the constraint expression is defined using the `left-layer`, `left-name` and `left-type` attributes. The name of the annotation layer id for the element is defined in the `left-layer` attribute; `left-name` is the name of the element. The `left-type` attribute denotes the referred element type, which may be `start` or `end`. The right hand side is defined in the same way using the `right-type`, `right-name` and `right-type` attributes. The `right-type` attribute additionally allows a type of `any` which is only valid for the outside operator.[7]

A single rule consists of one or more expressions. Thus the `rule` elements allows to group one or more element elements which are connected by an implicit logical **and**. A rule may also contain a choose element, which includes alternative elements. This can be used to use something like a logical **or** to choose between a set of alternative constraint expressions. The alternative elements contains one or more element elements.

Each rule may has an optional priority attribute, which can be used when choosing between two possible rules during processing.[8] The root element of the XML syntax is the constraints element, therefore all rule elements are inclosed in a single constraints element.

Figure 6 shows the example rule from figure 4 in the XML syntax representation. Using the XML syntax for constraint expressions, a validation process could be implemented using XSLT.

Figure 5

```
<!ELEMENT constraints (rule+)>

<!ELEMENT rule (choose | element+)>
<!ATTLIST rule
  priority CDATA #IMPLIED>

<!ELEMENT element EMPTY>
<!ATTLIST element
  relation (precedes|equals|outside|succeeds|inside|precedes-or-equals) #REQUIRED
  left-layer NMTOKEN #REQUIRED
  left-name NMTOKEN #REQUIRED
  left-type (start|end) #REQUIRED right-layer NMTOKEN #REQUIRED
  right-name NMTOKEN #REQUIRED
  right-type (start|end|any) #REQUIRED>

<!ELEMENT choose (alternative+)>

<!ELEMENT alternative (element+)>
```

A DTD for encoding constraint expressions in XML.

Figure 6

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE constraints SYSTEM "constraints.dtd">
<constraints>
  <rule>
    <element relation="precedes"
      left-layer="l1" left-name="div" left-type="start"
      right-layer="l2" right-name="l2" right-type="start"/>
  </rule>
</constraints>
```

Example rule from figure 4 expression in XML syntax.

Discussion

To define one grammar for an arbitrary set of overlapping XML documents can be a very complex task. It may not even be possible at all. But since we want to be able to define relations between different XML documents another solution must be found. This late breaking paper shows an approach similar to Schematron (cf. **[Jelliffe (2002)]**), which allows to define a set of constraints to describe a valid document. We applied that idea to XCONCUR. Using the constraint expression and rules we can define relations between the different annotation layers.

Constraint expression rules are easy to create and in contrast to carefully hand-crafted grammar constraint rules are also easily modified in the case that the schema for one of the annotation layers changes. Using constraint rules allows one to reuse already existing grammars, since the grammar itself is not affected by the constraint rules. The constraints just define in which way elements from different annotation layers my interleave with each other. The original grammers can, to a certain degree, be modified without changing the set of constraints, e.g. by adding word elements to sentence elements. If no elements are modified which affect the constraints, the constraint and the other grammars must not be adapted.

On the other hand, one cannot use constraint expression to generate documents, they are only useful for validation.

Notes

1. At the Extreme Markup Languages 2005 conference XCONCUR was presented as MuLaX.
2. Our approach has also been called "Bielefeld CONCUR" by other authors ([Sperberg-McQueen (2006)]).
3. Rules are applied on the whole document. It is surely desirable to be able to limit the scope of a specific rule. For example one might want to make sure that a suffix morpheme is never the first morpheme in a word. Future work will explore methods for limiting constraint expression rules to a specific scope. The modification of approaches similar to CSS selectors or XSLT template selection are being evaluated.
4. One can certainly use other element-names, but such an constraint rule expression will never evaluate to a positive result, therefore such a rule will not make much sense.
5. The position is to be understood as the same position in regard to the character stream of a document.
6. Strictly speaking even the outside operator could also be defined just in terms of the precedes operator. If the rules are applied universal qualified the operator could be defined as:

```
start(l1, a) << start(l2, b)
  end(l1, a) << start(l2, b)
```

7. The restriction between the relation-type 'outside' and the element type 'any' can unfortunately not be modeled with DTD directly. If we used a different document structure which is centered on the expression type, we could have modeled the restriction using a RelaxNG document grammar. But since a broad variety of XSLT processors may not work correctly with RelaxNG, we chose the described document structure.
8. This might chance in the future. Rule seletion during validation needs refinement.

Bibliography

[Barnard et al. (1995)] Barnard, David; Burnard, Lou; Gaspart, Jean-Pierre; Price, Lynne A.; Sperberg-McQueen, C. M.; Varile, Giovanni Battista: *"Hierarchical Encoding of Text: Technical Problems and SGML Solutions." The Text Encoding Initiative: Background and Contents*, Guest Editors Nancy Ide and Jean Vèronis = Computers and the Humanities 29/3 (1995) 211-231.

[Bauman 2005] Bauman, Syd: *TEI HORSEing Around*, Extreme Markup Languages 2005 Conference Proceedings, Montreal, 2005

[DeRose (2004)] DeRose, Steven: *Markup Overlap: A Review and a Horse*, Extreme Markup Languages 2004 Conference Proceedings, Montreal, 2004

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

[Durusau & O'Donnell (2002)] Durusau, Patrick and Matthew Brook O'Donnell: *Concurrent Markup for XML Documents*, XML Europe 2002 Proceedings, 2002

[Hilbert et al.(2005)] Hilbert, Mirco; Schonefeld, Oliver and Andreas Witt: *Making CONCUR work*, Extreme Markup Languages 2005 Conference Proceedings, Montreal, 2005

[Jelliffe (2002)] Jelliffe, Rick: *The Schematron Assertion Language 1.5*, 2002

[Sperberg-McQueen & Burnard (1994)] *Guidelines for Electronic Text Encoding and Interchange (TEI P3)*. Eds. C. M. Sperberg-McQueen and Lou Burnard. Chicago, Oxford: Text Encoding Initiative, 1994.

[Sperberg-McQueen (2006)] Sperberg-McQueen, C. M.: Rabbit/duck grammars: a validation method for overlapping structures. To appear in: Proceedings of Extreme Markup Languages. Montreal, 2006

[Witt (2004)] Witt, Andreas: *Multiple hierarchies: new aspects of an old solution*. Proceedings of Extreme Markup Languages. Montreal, 2004

Towards validation of concurrent markup

Oliver Schonefeld [Bielefeld University]
Andreas Witt [Bielefeld University]